THE STRUCTURE AND DYNAMICS OF LARGE SOCIAL NETWORKS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Liaoruo Wang

January 2013

THE STRUCTURE AND DYNAMICS OF LARGE SOCIAL NETWORKS

Liaoruo Wang, Ph.D.

Cornell University 2013

In this thesis, we first explore two different approaches to efficient community detection that address different aspects of community structure. We establish the definition of community fundamentally different from previous literature, where communities were typically assumed to be densely connected internally but sparsely connected to the rest of the network. A community should be considered as a densely connected subgraph in which the probability of an edge between any two vertices is higher than average. Further, a community should also be well connected to the remaining network, that is, the number of edges connecting a community to the rest of the graph should be significant. In order to identify a well-defined community, we provide rigorous definitions of two terms: "whiskers" and the "core". Whiskers correspond to subsets of vertices that are barely connected to the rest of the network, while the core exclusively contains the type of community we are interested in. We prove that detecting whiskers, or equivalently, extracting the core, is an NP-complete problem for both weighted and unweighted graphs. Then, three heuristic algorithms are proposed for finding an approximate core and are evaluated for their performance on large networks, which reveals the common existence of the core structure in both random and real-world graphs. Well-defined communities can be extracted from the core using a number of techniques, and the experimental results not only justify our intuitive notion of community, but also demonstrate the existence of large-scale communities in various networks.

An (α, β)-community is a connected subgraph C with each vertex in C connected to at least β vertices of C (self-loops counted) and each vertex outside of C connected to at most α vertices of C ($\alpha<\beta$). We present a heuristic algorithm that in practice successfully finds a fundamental community structure. We also explore the structure of (α, β)-communities in various social networks. (α, β)-communities are well clustered into a small number of disjoint groups, and there are no isolated (α, β)-communities scattered between these groups. Two (α, β)-communities in the same group have significant overlap, while those in different groups have extremely small pairwise resemblance. A surprising core structure is discovered by taking the intersection of each group of massively overlapping (α, β)-communities. Similar experiments on random graphs demonstrate that the core structure found in many social networks is due to their underlying social structure, rather than due to high-degree vertices or a particular degree distribution.

In many social networks, there exist two types of users that exhibit different influence and different behavior. For instance, statistics have shown that less than 1% of the Twitter users (e.g. entertainers, politicians, writers) produce 50% of its content [1], while the others (e.g. fans, followers, readers) have much less influence and completely different social behavior. In this thesis, we define and explore a novel problem called community kernel detection in order to uncover the hidden community structure in large social networks. We discover that influential users pay closer attention to those who are more similar to them, which leads to a natural partition into different community kernels. We propose GREEDY and WEBA, two efficient algorithms for finding community kernels in large social networks. GREEDY is based on maximum cardinality search, while WEBA formalizes the problem in an optimization framework. We

conduct experiments on three large social networks: Twitter, Wikipedia, and Coauthor, which show that WEBA achieves an average 15–50% performance improvement over the other state-of-the-art algorithms, and WEBA is 6–2,000 times faster on average in detecting community kernels.

Cascading processes, such as disease contagion, information diffusion, and viral marketing, are a pervasive phenomenon in many types of networks. The problem of devising intervention strategies to facilitate or inhibit such processes has recently received considerable attention. However, a major challenge is that the underlying network is often unknown. In this thesis, we revisit the problem of inferring latent network structure given observations from a diffusion process, such as the spread of trending topics in social media. We define a family of novel probabilistic models that can explain recurrent cascading behavior, and take into account not only the time differences between events but also a richer set of additional features. We show that MAP inference is tractable and can therefore scale to very large real-world networks. Further, we demonstrate the effectiveness of our approach by inferring the underlying network structure of a subset of the popular Twitter following network by analyzing the topics of a large number of messages posted by users over a 10-month period. Experimental results show that our models accurately recover the links of the Twitter network, and significantly improve the performance over previous models based entirely on time.

BIOGRAPHICAL SKETCH

Liaoruo (Laura) Wang was born on April 3, 1983 in Harbin, China. She moved to Beijing, China with her family in 1995. After graduating with her high school diploma in 2001 from Beijing No. 4 High School, she was admitted to Tsinghua University in Beijing, China for her undergraduate study. She graduated with the Bachelor of Science degree in Electronic Engineering in July 2005. In September 2005, she enrolled as a graduate student in the Electrical and Computer Engineering department at the University of Massachusetts in Amherst, Massachusetts, where she received full financial support. In June 2007, she passed the thesis defense and obtained the Master of Science degree, focusing on connectivity in cooperative wireless ad hoc networks. She was co-advised by Dr. Dennis Goeckel and Dr. Don Towsley. In August 2007, she enrolled as a graduate student in the Electrical and Computer Engineering department at Cornell University in Ithaca, New York. Her doctoral research, under the direction of Dr. John E. Hopcroft, focused on the structure and dynamics of large social networks. After graduation, she will pursue an industrial career in California.

I would like to dedicate this Doctoral Dissertation to my family, especially to my dad and mom who open my eyes to the world, for their unconditional support, encouragement, and love through these many years of research.

ACKNOWLEDGEMENTS

I would never have been able to finish this dissertation without the support of many people. I would like to give my heartfelt acknowledgement to my advisor, Dr. John E. Hopcroft, for giving me the opportunity to work on this unique topic that I love very much, and for his patience, encouragement, many great inspirations and regular, constructive feedback on the progress throughout my graduate studies. I would also like to thank my committee members, Dr. Tsuhan Chen and Dr. Sidney Resnick, for their valuable suggestions and fruitful discussions during this endeavor.

Finally, I wish to specially thank my parents, my loved ones, and numerous friends for their unwavering support and understanding during the many hours I dedicated to achieving this milestone in my life and career. They were always supporting me and encouraging me with their best wishes through the good times and bad.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Computer science is undergoing a fundamental change and is reshaping our understanding of the world. An important aspect of this change is the theory and applications dealing with the gathering and analyzing of large real-world datasets. Innovative ways of analyzing such datasets allow us to extract useful information that we would never have obtained from small or synthetic datasets, thus providing us with new insights into the real world. In recent years, we have witnessed the rapid growth of social media (e.g., friendship network Facebook and Twitter, media-sharing website YouTube and Flickr, professional network LinkedIn). Such networks usually have hundreds of millions of users all over the world, thus require the support of gigantic data storage facilities. With the development of storage systems and database technologies, we have access to a much larger amount of real-world data than ever before, motivating the social network research to answer many open social science questions. Understanding the structure and dynamics of social networks can help us improve the quality of social media, the design of system infrastructure, and many other real-world applications, such as recommendation systems, viral marketing, and financial services.

Traditional research in computer science has focused primarily on problems with small input size. For instance, before any form of search was available, internet users were required to provide the IP address of the website to which they seek to connect. One of the first internet search providers, Ask Jeeves (later known as `Ask.com`), was founded in 1996. The service partially relied on professional editors to manually select the best answers for specific search queries.

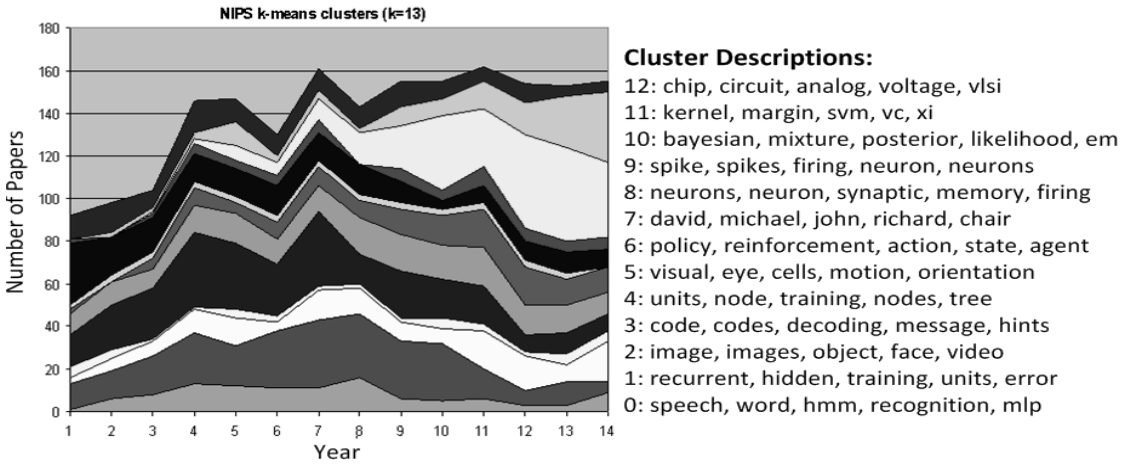


Figure 1.1 The distribution of k=13 clusters of NIPS papers [3]. The histograms of each cluster are stacked on top of each other to show the influence of cluster popularity over time.

Given the large scale of internet and the sheer number of websites nowadays, such a strategy is apparently no longer feasible. For another example, in the past, stores tracked the items purchased by each individual customer and gave that customer discounts for future purchases of those items. However, with the help of modern algorithms, service providers such as Netflix are now able to, not only make predictions based on a customer's past preferences, but also amalgamate preferences from millions of customers to make intelligent suggestions to effectively increase sales revenue.

The availability of real-world data over a period of time has made it possible to explore the current dynamics and to infer the future trend. For example, with thousands of scientific publications released every year, information on which research areas are growing or declining would be of great help to rank their popularity and to predict their evolutionary trend. As shown in Fig. 1.1, Shaparenko et al. [3] used sophisticated artificial intelligence techniques to cluster papers from the Neural Information Processing Systems (NIPS) conference between 1987 and 2000 into several groups. Clusters 10 on Bayesian methods

and Cluster 11 on kernel methods clearly show the two growing research areas. The graph correctly indicates that Cluster 10 emerged before Cluster 11, both starting to dominate the conference by 2000. Meanwhile, Cluster 1 on neural networks, Cluster 4 on supervised neural network training, and Cluster 8 on biologically-inspired neural memories were popular in the early years of NIPS, but almost disappeared from the conference by 2000. With the help of advanced computer science techniques, we are able to accurately predict how important a publication will be and how a research area will evolve.

With the recent explosion of online social media producing an unprecedented volume of new data, the research on social networks has attracted a great amount of attention, with scientists envisioning an opportunity to answer many open social science questions. Combining graph theory tools with data mining techniques, we study problems related to **large social networks** in this thesis, such as detecting community structure, identifying different levels of social influence, and inferring latent networks of diffusion. In our research, we design, implement, and evaluate efficient algorithms that scale to large real-world networks with millions of users for finding different types of communities. By exploring the obtained community structure, we categorize users according to their social influence. By tracking community structure over time, we analyze and predict its dynamic behavior, including how communities emerge, evolve, and disappear. Further, we model diffusion processes and infer the structure of unknown networks. These research challenges offer the potential to revolutionize many aspects of our lives, such as *search result customization*, *friend and product recommendation*, *business and financial decision making*, *resource planning*, *predictive marketing and advertising*. By integrating knowledge and tools from both computer science and social science disciplines, we develop computation-

ally efficient and mathematically rigorous models and algorithms to understand the fundamental structure, network dynamics, and behavioral patterns hidden in social interactions.

1.1 How to detect community structure in social networks?

Motivation and Challenge.

With the widespread proliferation of social websites such as Twitter and Facebook, understanding community structure is not only essential to study social interactions, but also crucial to improve the quality of online services, including *search engines*, *business decision making*, *resource planning*, and *predictive marketing*. Previous work was largely focused on graph partitioning and often considered communities as subsets barely connected to the rest of the network by only a few links. However, an individual may belong to multiple communities at the same time. Further, in our view, communities are not only better connected than expected from chance, but also well connected to the rest of the network via a significant number of edges, which is possibly even larger than the number of its internal edges. This intuition fundamentally redefines the problem of community detection, and leads to the discovery of community structure that most existing methods fail to detect.

Contributions.

We develop mathematical definitions of community and several heuristic algorithms that in practice efficiently find a fundamental **core structure**. These algorithms can eliminate unimportant periphery as well as random factors from the

network. Their performance can be verified by experiments on random graphs and real-world datasets. Surprisingly, in a Twitter network with 112,957 vertices and 481,591 edges, we find 6,912 distinct communities of size 200. These communities are densely connected internally while having a significant number of outgoing edges. Further, they are neatly categorized into a small number of massively overlapping clusters. Two communities in the same cluster have significant overlap ($>$90%), while two communities in different clusters have extremely small ($<$5%) overlap. Thus, we discover a core structure in many social networks by taking the intersection of a group of massively overlapping communities. The communities are well clustered into a small number of disjoint cores, and there are no isolated communities scattered between these cores. By contrast, the cores found in random graphs usually have significant overlap among them. Our experiments demonstrate that the core structure found in many social networks is indeed due to their underlying social structure, rather than due to high-degree vertices or a particular degree distribution.

Detailed Introduction.

We give a definition of (α, β)-community slightly different from that of Mishra et al. [10]. Without fixing the values of α and β, our definition highlights the contrast of internal and external connectivity. We develop a heuristic algorithm based on (α, β)-community that in practice efficiently finds a fundamental community structure. Our algorithm is focused on the difference $\beta - \alpha$ and is thus robust to the specific values of α and β. Further, we thoroughly explore the structure of (α, β)-communities in various large social networks. In a Twitter following network with 112,957 vertices and 481,591 edges, there are 6,912 dis-

tinct (α, β)-communities of size 200 with 45,361 runs of the algorithm. These (α, β)-communities are neatly categorized into a small number of massively overlapping clusters. Two (α, β)-communities in the same cluster have significant overlap (>90%), while two (α, β)-communities in different clusters have extremely small (<5%) pairwise similarity. This leads to the notion of *core*, which is the intersection of a group of massively overlapping (α, β)-communities.

Our definition provides an intuitive criterion as to whether to classify a subgraph as a community. The edges connecting each vertex in the community to vertices of the community should be strictly more than those connecting any vertex outside the community to vertices of the community. Further, by taking the intersection of a number of massively overlapping (α, β)-communities, the set of (α, β)-communities which differ by only a few vertices is reduced to an underlying core. Thus, each (α, β)-community contains one of the few cores and some peripheral vertices, and these peripheral vertices are what gives rise to such a large number of (α, β)-communities.

We can extract the core structure by taking the intersection of a group of massively overlapping (α, β)-communities with multiple runs of the algorithm. The number of cores decreases as k increases. For large k, the (α, β)-communities are well clustered into a small number of disjoint cores, and there are no isolated (α, β)-communities scattered between these cores. The cores obtained for a small k either disappear or merge into the cores obtained for a larger k. Further, the cores correspond to dense regions of the graph, and there are no bridges of intermediate (α, β)-communities connecting one core to another. By contrast, the cores found in various random graphs usually have significant overlap among them, and the number of cores does not necessarily decrease as

k increases. Extensive experiments demonstrate that the core structure found in various social networks is indeed due to their underlying social structure, rather than due to high-degree vertices or a particular degree distribution.

The number and average size of cores in the Twitter graph with respect to the community size k are given in Table 2.3. As k increases, some cores disappear due to their small neighborhood (Definition 2.2.4), while others merge into larger ones due to their high closeness (Definition 2.2.5). We explore some interesting questions in this section, for example, what causes many social networks to display the core structure, why (α, β)-communities correspond to well-defined clusters, and why there are no bridges of (α, β)-communities connecting one core to another. A bridge is a sequence of intermediate (α, β)-communities that connect two cores with substantial overlap between adjacent pairs.

1.2 How to identify different levels of social influence?

Problem Formulation.

The Pareto principle (a.k.a. 80-20 rule) can be observed in many social networks. In these networks, there exist two types of users that exhibit different social influence and social behavior. For example, less than 1% of the Twitter users (e.g., entertainers, politicians, writers) produce more than 50% of the content on the micro-blogging site, while the other 99% (e.g., fans, followers, readers) are much less influential. We also know that, in social networks, homophily accelerates the diffusion process but inhibits the spread of innovative ideas. Then, one might ask interesting questions such as: "do influential users form commu-

nities with significant homophily?" and "what is the underlying structure between influential users and their numerous followers?" To answer these questions, we formulate the problem of **community kernel detection** that includes two subtasks: (1) distinguish influential users from others, and (2) identify the community structure among influential users and their followers.

Motivation and Challenge.

Community kernel detection has many direct applications, such as *marketing strategies*, *recommendation systems*, and *network visualizations*. However, this problem is non-trivial and poses a set of challenges. First, it is difficult to determine truly influential users without any prior information. Second, it is unclear how influential users interact with each other. Finally, real-world social networks are growing fast with millions of users, and it is important to design an algorithm with high scalability.

Contributions.

We formulate the problem of community kernel detection in large social networks as two subtasks: identifying influential (kernel) members and uncovering the structure of community kernels. We propose two algorithms to solve these two subtasks in a unified approach. The first algorithm is a greedy algorithm based on maximum cardinality search. It can efficiently obtain an approximate solution, but does not have a bounded error. The second algorithm is a weight-balanced algorithm based on a multi-dimensional objective function that explicitly quantifies the community kernels. It can efficiently obtain an ap-

proximate solution with a small error bound. We validate their effectiveness and efficiency on three representative large social networks: a Twitter following network, an arXiv co-authorship network, and a Wikipedia co-editorship network. Our algorithms achieve an average 60–300% performance improvement over the other state-of-the-art algorithms, and are on average 6–2,000 times faster in detecting community kernels.

Detailed Introduction.

The problem of community detection has been extensively studied and many algorithms have been proposed, such as cut- and conductance-based methods [11–14], spectral clustering [4, 15, 16], (α, β)-clustering [10, 17], and topic modeling methods [18]. The cut- and conductance-based and spectral clustering methods are usually based on a fundamental assumption that communities have dense internal connections and sparse external connections. (α, β)-clustering methods relax this assumption by allowing communities to have dense external connections. Topic modeling methods are based on statistical analysis of the content information associated with each vertex. However, these methods ignore an important fact that the community structure of influential users is quite different from that of others. Our preliminary statistical analysis shows that the average degree of influential users is almost 10 times more than that of the others in the Twitter network.

To clearly demonstrate this, we present an example from the Twitter network as shown in Fig. 3.1. Refer to Section 3.4 for more details on the Twitter network. The left figure is an input of the Twitter following network with three entertainers (Oprah Winfrey, Ashton Kutcher, and Demi Moore) and two politi-

cians (Barack Obama and Al Gore) as well as some of their followers. This input represents a typical network structure with a few influential users connected with the rest of the network via a large number of links. To detect the community structure of this network, we consider Newman's algorithm [4], a state-of-the-art method based on modularity. The middle figure shows the community structure obtained by Newman's algorithm. We observe that, since there are a large number of connections between each influential user and its followers, Newman's algorithm tends to partition the influential users into different communities and to group them with their respective followers. The lack of ability to distinguish influential users from their numerous followers is a key problem with this method. The right figure shows the community structure obtained by our algorithm WEBA later introduced in Section 3.3. By contrast, this is exactly what one would expect a community detection algorithm to discover: two community kernels (one of entertainers and one of politicians) consist of influential users and two auxiliary communities associated with the kernels. Thus, in Chapter 3, we refer to this problem as community kernel detection, which includes two parts: (1) how to distinguish influential users (kernel members) from others, and (2) how to detect the community structure (community kernels) among influential users and their respective auxiliary communities.

The problem of community kernel detection has many practical applications, including representative user finding, friend recommendation, network visualization, and marketing. However, this problem is non-trivial and poses a set of challenges. First, it is difficult to identify the truly influential users. One may consider to use the number of followers as an indicator. Unfortunately, the follower count gives no information about who follows them. Second, it is unclear how influential users interact with each other. Would a politician tend to follow

another politician or an actress? Finally, real-world social networks are growing fast with thousands or millions of vertices. It is important to develop an algorithm with high scalability.

1.3 How to infer networks of diffusion?

Problem Formulation.

Given a network with a fixed population, we make observations on a diffusion process (e.g., the spread of a disease), which consist of a set of cascades. Each cascade is a record of observed infection times within the population during a given time interval. We assume that diffusion processes occur in static but unknown networks, and infections along edges occur independently of each other. With a generative probabilistic model to describe how infections propagate over time, we formulate the problem of **social network inference** that includes three subtasks: (1) compute the marginal probability density of each possible edge, (2) reconstruct the connectivity of the network, and (3) infer the likelihood of possible infections.

Motivation and Challenge.

Due to limited access to massive social network data, there are many cases where the complete network must be inferred from other measurements such as information diffusion. The process of diffusion has received considerable attention in a broad range of applications: *information propagation*, *viral marketing*, and *epidemiology*. Uncovering the hidden dynamics of diffusion networks is impor-

tant to control infections, increase sales, and forecast propagations. However, as is often the case, we know when an infection occurs but we do not know why the infection occurs, where the infection comes from, or how long it has taken for the infection to occur. Without these "why", "where", and "how" information, understanding the mechanism underlying the diffusion process can be rather difficult.

Contributions.

We define a family of novel probabilistic models that generalize prior models based solely on time. We propose a primary approach MONET that can handle recurrent diffusion processes. Further, we consider a richer set of additional features for infection events, defining novel feature-enhanced models that can better explain the observed data. With distributed optimization and convex objective functions, we can efficiently solve the problem of inferring the most probable latent network structure. Using additional features such as the languages of the messages and Jaccard indexes between the messages, we can accurately recover the links of the Twitter network by analyzing the topics of a large number of messages posted by a subset of the Twitter users over a 10-month period. Experimental results show that our models significantly improve the accuracy of the estimates over previous models by as much as 78.7%.

Detailed Introduction.

We revisit the problem of inferring latent network structure given observations of a diffusion process. For example, by observing a disease epidemic, we want

to infer the underlying social contact network, or by observing the spread of trending topics, we want to estimate the connectivity of the social media. Fig. 4.1 illustrates a case of information diffusion in the popular Twitter network. The nodes represent a subset of the Twitter users that have posted about a common trending topic, and the directed edges represent the "following" relation between the users. There is a clear pattern in the figure. The bigger and darker nodes, followed by the smaller and lighter nodes, form the hubs of the diffusion process. By looking at the time-stamps of the messages and at the underlying network structure, we observe that most information flows initiate at a hub node and spread across the network to reach other hub nodes and their followers. However, it is non-trivial to come up with such a picture simply by looking at the time-stamps of the messages, since without knowing the underlying network structure, we cannot decide from whom a node copied the information from. Intuitively, messages carry implicit information about the social relations among users. For instance, users who repeatedly post messages about the same topic within a short period of time, are more likely to be connected. Thus, a motivating application is to what extent we can estimate the relations in social networks by analyzing the messages published by users over time. Tweets that contain trending topics often start with one or more users and then spread through the network. We can sometimes identify the implicit social relations between users if their tweets show apparent connections (such as the "retweet" or "@" symbol). But, does the context of tweets contain enough information to fully recover the network structure?

This type of latent network inference problem based on the time-stamps of infection (or, information-reproduction) events has received increasing interest over the past few years [2, 19, 20]. Previous work was largely based on two ma-

jor assumptions: (1) the diffusion process is causal (i.e., not affected by events in the future), and (2) infection events closer in time are more likely to be causally related (e.g., according to an exponential, Rayleigh, or power-law distribution). While the causality assumption is indeed crucial and always satisfied in practice, we realize that there are many other factors that can be highly informative as far as the causality relations are concerned. For example, the time-stamps at which two users publish their tweets are important to decide whether they are related, but other factors such as the language or the content of the messages can be as important. Even if the two messages are close in time, they are unlikely to be related if the messages are written in different languages. Further, previous models in the literature are mostly focused on monotonic processes, while real-world processes are often recurrent. For instance, it is very common for one user to post about the same topic multiple times on Twitter, or purchase the same item regularly on Amazon.

The rest of this thesis is organized as follows. In Chapter 2, we discuss the problem of detecting community structure in social networks. We propose two heuristic approaches to efficiently solving this problem and provide the NP-hardness proofs. In Chapter 3, we discuss the problem of identifying different levels of social influence. We propose two heuristic algorithms and provide empirical results. In Chapter 4, we discuss the problem of inferring diffusion networks. We propose a family of feature-enhanced probabilistic models and conduct experiments on real-world datasets. Finally, we discuss several related research topics in Chapter 5 and conclude in Chapter 6 with comments on the problems considered and future work.

CHAPTER 2

HOW TO DETECT COMMUNITY STRUCTURE IN SOCIAL NETWORKS?

In this chapter, we introduce two community detection methods to efficiently detect community structure in large social networks.

2.1 Community Detection by the "Core" Structure

Ever since people started to realize the importance of comprehending how interactions initiate and develop, the research on complex networks has attracted a great amount of attention. A substantial quantity of work has been devoted to the task of identifying and evaluating close-knit communities in large complex networks, most of which is based on the premise that it is a matter of common experience that communities exist in these networks [21]. In particular, as the Internet has become an indispensable part of our life, understanding community structure is not only crucial for studying real-world societies, but also helpful to improve the accuracy and reliability of predicting online behaviors, which may greatly benefit the quality and effectiveness of online services, such as search engines, recommendation systems, and so on.

A complex network is usually modeled as a graph in which vertices represent entities and edges represent interactions between pairs of entities. In previous studies, a community was often assumed to be a subset of vertices that are densely connected internally but sparsely connected to the rest of the network [21–23]. Accordingly, numerous measures have been proposed to capture this feature, out of which conductance has become one of the most widely adopted metrics for evaluating how community-like a subset of vertices is. Par-

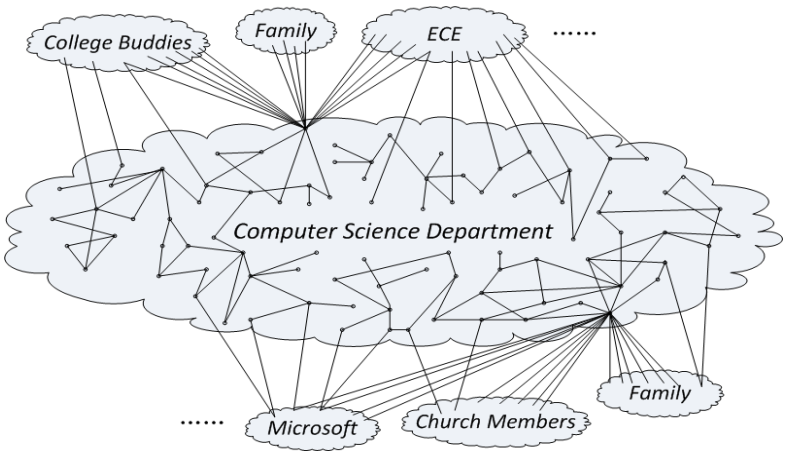


Figure 2.1 An example friendship network. Vertices typically have a significant number of cut edges.

ticularly, Leskovec et al. [21] conducted an extensive research on more than 100 large complex networks under the assumption that a community is more densely connected between its members than between its members and the remaining network. They carefully examined the relationship between conductance and community size, and discovered that the best community of the entire graph, i.e. the subset with the global minimum conductance, is usually a small set of vertices barely connected to the rest of the network by just a single edge.

However, it is our view that for real-world societies, communities are not only better connected than expected solely from chance, but are also well connected to the rest of the network. Actually, it is hard to imagine a small close-knit community, such as an academic department, with only one edge connecting it to the outside world. Empirically, a community displays a higher than average edge-vertex^2 ratio, which reflects the probability of an edge between two randomly-picked vertices, and it is also connected to the rest of the network via a significant number of edges, which is even possibly larger than the number of its internal edges, as depicted in Fig. 2.1.

Given a subset of vertices, an edge with only one endpoint inside the subset can be thought as a cut edge. A densely connected subset with a small number of cut edges, called a whisker, is not the type of community we are interested in. Since many previously-used measures simultaneously maximize internal connections and minimize external connections, leaving whiskers in the graph will interfere with the algorithms intended to extract the type of community we are interested in. Whiskers are peripheral rather than central, thus, the type of community we would like to identify is embedded in a special structure in which no whiskers exist, called the core. To get rid of the interference generated by whiskers, a community detection algorithm can be designed consisting of two steps: 1) identifying the core in which no whiskers exist, and 2) identifying communities in the core. Apparently, any subset of the core is connected to the rest of the graph by a moderate number of edges, and conductance can still be taken as a measure of community goodness. In this way, the best community is not only more densely connected than expected from chance but also well connected to the remaining network, which exactly corresponds to our intuitive notion of community.

We prove that extracting the exact core is NP-complete for both weighted and unweighted graphs. It is not difficult to see that, generally, the exact core cannot be obtained by removing whiskers one by one, but removing whiskers in a certain way can lead to an approximate core. We develop three heuristic algorithms, all of which are capable of finding an approximate core. Their performance can be verified by the experimental results obtained from random graphs and real-world graphs. In addition, we also discover that some algorithms are only suitable for a certain kind of networks but not for others. Further, the algorithms can be justified by the community profile of the core, in contrast to that of

the entire graph shown in [21], which plots the smallest possible conductances with respect to fixed community sizes. In various complex networks, the best communities have a relatively large conductance, which means the communities are densely connected internally while preserving a significant number of cut edges. Moreover, they also have a relatively large size, which demonstrates the existence of large-scale well-defined communities.

The rest of this section is organized as follows. In Section 2.1.1, we introduce some necessary background and present definitions of whiskers and the core. Then, in Section 2.1.2, we prove the NP-completeness of finding the exact core in weighted and unweighted graphs, and propose three heuristic algorithms for finding an approximate core. In Section 2.1.4, we apply the algorithms to real-world and random graphs to evaluate and compare their performance. Finally, we conclude in Section 2.1.5 with comments on the problems considered and future work.

2.1.1 The "Core" Structure

In this section, we first review some previous research on community discovery in large complex networks. Then, we establish the theoretical foundation by providing rigorous definitions of several terminologies related to "whiskers" and the "core". Finally, we prove two preliminary lemmas to explore the properties of "whiskers" and the "core".

Background

Given an undirected graph $G = (V, E)$ with adjacency matrix A, the conductance of a subset of vertices $S \subseteq V$ is defined as

$$\phi(S) = \frac{\sum_{i \in S, j \notin S} A_{ij}}{\min\{D(S), D(S^c)\}}. \tag{1}$$

Here, S^c denotes the complement of S and

$$D(S) = \sum_{i \in S} \sum_{j \in V} A_{ij} = \sum_{i \in S} d(i),$$

where $d(i)$ denotes the degree of vertex i in the graph G [21]. Clearly, the conductance of S provides a measure for the quality of the corresponding cut, which divides the graph into two subsets S and S^c. Out of numerous density-based measures, conductance has been extensively employed for community detection, which intends to maximize internal connectivity and minimize external connectivity [22, 24].

The concept of whiskers was informally introduced in [21] referring to weakly-connected subsets linked to the rest of the graph by just a single edge. Empirically, whiskers are peripheral and can be removed from the graph using the depth-first search to extract the giant biconnected component. Then, the union of whiskers is considered to form the periphery of the graph and the giant biconnected component is considered as the core. However, the biconnected component may still display a core-periphery structure with whiskers now referring to weakly-connected subsets linked to the rest of the graph by two edges, which inspires our generalized definitions of whiskers and the core.

In [21], a large number of real-world complex networks, such as friendship, citation, email and road networks, were thoroughly explored. Also, several approximation algorithms for community identification were implemented and

evaluated. These algorithms typically return a whisker or a union of disjoint whiskers as the best community, thus, whiskers are often interpreted as meaningful communities and are believed to have a significant influence on the community structure of the entire network. In addition, the network profile plot was also introduced to uncover the relationship between the lowest conductances and fixed community sizes, which usually achieves a global minimum at a small size scale of roughly 100 vertices.

A close-knit subset with just a single edge connecting it to the rest of the graph often corresponds to a low-conductance subset, which is more likely to be extracted as the optimal solution by the algorithms designed to minimize the conductance, and this may explain the reason why the best community is usually a whisker or a union of disjoint whiskers.

"Whiskers" and the "Core"

Let $G = (V, E)$ be an undirected graph with n vertices and m edges. A cut C is a collection of edges such that removing them from the graph G separates the vertex set V into two disjoint subsets S and S^c, where S^c denotes the complement of S and $C = \{(v, w) \in E \mid v \in S;\ w \in S^c\}$. Without loss of generality, we assume $|S| \leqslant |S^c|$ throughout this section, where $|S|$ and $|S^c|$ denote the cardinality of sets S and S^c, respectively. Note that both S and S^c are not necessarily connected. Then, an edge $(v, w) \in C$ is called a cut edge, and intuitively, the cut size is defined to be the cardinality of the set C. Further, a cut is considered to be *suitable* if its removal divides the vertices into two disjoint subsets such that both have cardinality greater than or equal to the cut size.

Definition 2.1.1. *A cut of size k is a* **suitable cut** *if its removal from the graph parti-*

tions the vertex set into two disjoint subsets S and S^c, where $k \leqslant |S| \leqslant |S^c|$.

Leskovec et al. [21] defined 1-whiskers to be maximal subgraphs that can be detached from the rest of the graph by removing a single edge, and they also use the term "whiskers" informally to refer to subsets of vertices barely connected to the rest of the graph. Whiskers are generally quite small compared to the whole graph while possessing a wide range of sizes and shapes. Moreover, they usually correspond to low-conductance sets that are more densely connected inside than connected to the outside. Whiskers and unions of disjoint whiskers are believed to exert a significant effect on the community structure of real-world networks, since they are extracted and interpreted as communities by the conductance measure, which, out of numerous density-based measures, has been extensively used for detecting communities and evaluating their quality [21, 22, 24].

However, as clarified in Section 2.1, this type of community neither corresponds to our intuitive notion of community nor widely exist in real-world societies, where it is a matter of common observation that communities are not only densely connected inside but also well connected to the outside. Therefore, it is of major interest to remove whiskers from the graph in order to provide insight into the community structure of the network core. For this purpose, we rigorously define whiskers and the corresponding core structure where barely-connected subsets have been removed.

Definition 2.1.2. *Given an undirected graph $G = (V, E)$ with n vertices, a* **k-whisker** *is defined as a connected subgraph $G_w(k) = (V_w(k), E_w(k))$ linked to the rest of the graph by k edges, where $k \leqslant |V_w(k)| \leqslant n/2$.*

Definition 2.1.3. *Given an undirected graph $G = (V, E)$ with n vertices, a* **maxi-**

mal k-whisker *is defined as a maximal connected subgraph* $G_w^*(k) = (V_w^*(k), E_w^*(k))$ *linked to the rest of the graph by* k *edges, where* $k \leqslant |V_w^*(k)| \leqslant n/2$.

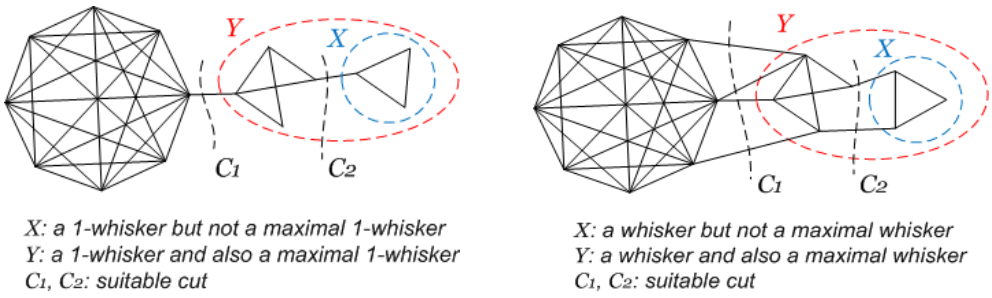
Small isolated components are frequently encountered in large complex networks, and they can simply be viewed as (maximal) 0-whiskers. Definition 2.1.2 and 2.1.3 are a direct extension of the definition of 1-whiskers given in [21]. Then, in a similar way, the definitions of whiskers and maximal whiskers can be formulated independent of the value of k referring to weakly-connected subsets attached to the remaining graph via a small number of edges.

Definition 2.1.4. *Given an undirected graph* $G = (V, E)$ *with* n *vertices, a* **whisker** *is defined as a connected subgraph* $G_w = (V_w, E_w)$ *linked to the rest of the graph by a suitable cut, where* $|V_w| \leqslant n/2$.

Definition 2.1.5. *Given an undirected graph* $G = (V, E)$ *with* n *vertices, a* **maximal whisker** *is defined as a maximal connected subgraph* $G_w^* = (V_w^*, E_w^*)$ *linked to the rest of the graph by a suitable cut, where* $|V_w^*| \leqslant n/2$.

See Fig. 2.2 for a detailed illustration of Definition 2.1.1 through Definition 2.1.5. A maximal whisker is obviously a whisker, but a whisker is not necessarily a maximal whisker, since it can be contained in a larger whisker. Besides, a 0-whisker is also a maximal whisker by Definition 2.1.5.

As discussed above, maximal whiskers, although argued by some to be community-like, are not what we are interested in here. Therefore, we define the core as the remaining structure after removing the union of all maximal whiskers from the graph. Meaningful communities can be further extracted from the core using a variety of algorithms, which, unlike whiskers, are not



(a) k-whisker vs. maximal k-whisker (b) whisker vs. maximal whisker

Figure 2.2 Schematic illustrations of Definition 2.1.1 through Definition 2.1.5.

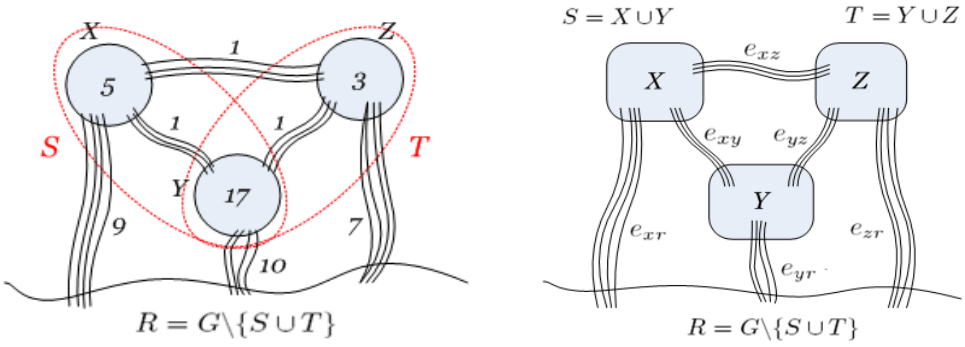
only better connected than expected from chance but also well connected to the rest of the graph.

Definition 2.1.6. *The* **core** *is a connected subgraph that is the complement of the union of all maximal whiskers.*

Clearly, there does not exist any suitable cut in the core subgraph. Before we move on to Section 2.1.2 to design and implement algorithms for finding the core structure and its underlying communities, we first examine some properties of whiskers. If all maximal whiskers are disjoint in the graph, it is straightforward that we can remove these disjoint whiskers one by one until we obtain the core. However, whiskers may overlap with each other, and unfortunately, their union is often no longer a whisker. In fact, a number of counterexamples can be constructed to justify this statement, and we conclude the following lemma:

Lemma 2.1.7. *Let G be an undirected graph with two overlapping maximal whiskers S and T. The subgraph $S \cup T$ is not necessarily a whisker.*

Proof. As shown in Fig. 2.3(a), for instance, $S = X \cup Y$ is a maximal whisker with 22 vertices and 21 outgoing edges. Similarly, $T = Y \cup Z$ is also a maximal



(a) Two overlapping maximal whiskers. (b) Two overlapping whiskers.

Figure 2.3 Each circled integer denotes the number of vertices in that subset.

whisker with 20 vertices and 19 outgoing edges. However, there are a total of 25 vertices in the set $X \cup Y \cup Z$ and 26 outgoing edges that connect this union to the rest of the graph, thus $S \cup T$ is not a whisker. □

In general, there are two reasons why a subset of vertices is not a whisker: 1) it contains more than half of the vertices, and 2) the number of edges connecting it to the rest of the graph is strictly greater than its cardinality. Thus, the union of two disjoint whiskers is still a whisker if and only if it is no larger than its complement. In addition, the union of two overlapping maximal whiskers is not a maximal whisker, since a maximal whisker cannot have any other maximal whisker as its subset. Based on Lemma 2.1.7, there is another observation we can make about whiskers:

Lemma 2.1.8. *Let G be an undirected graph with n vertices and two overlapping whiskers S and T, where the number of vertices in the subgraph $S \cup T$ is no more than $n/2$. If $S \cup T$ is not a whisker, then $S \cap T$ must be a whisker.*

Proof. Assuming that the subgraph $S \cup T$ is not a whisker, write $S = X \cup Y$ and

$T = Y \cup Z$ where $S \cap T = Y$, as shown in Fig. 2.3(b). Then, it follows that

$$e_{xr} + e_{xz} + e_{yr} + e_{yz} \leqslant v_x + v_y \tag{2.1}$$

$$e_{yr} + e_{xy} + e_{zr} + e_{xz} \leqslant v_y + v_z \tag{2.2}$$

$$e_{xr} + e_{yr} + e_{zr} > v_x + v_y + v_z \tag{2.3}$$

where v_x, v_y, and v_z denote the number of vertices in the sets X, Y, and Z, respectively. Adding Equation (2.1) and (2.2), we have that

$$\begin{aligned} e_{xr} + 2e_{yr} + e_{zr} + e_{xy} + e_{yz} + 2e_{xz} &\leqslant v_x + 2v_y + v_z \\ &< e_{xr} + e_{yr} + e_{zr} + v_y. \end{aligned}$$

Thus,

$$e_{yr} + e_{xy} + e_{yz} + 2e_{xz} < v_y. \tag{2.4}$$

Since e_{xz} is non-negative as the number of edges between the sets X and Z, by Equation (2.4),

$$e_{yr} + e_{xy} + e_{yz} < v_y,$$

and the subgraph $Y = S \cap T$ is clearly a whisker. □

2.1.2 Methodology

In this section, we propose an efficient approach for identifying the core in a given graph. Armed with the definitions provided in Section 2.1.1, we prove that detecting whiskers is NP-complete in both weighted and unweighted graphs, and thus is computationally intractable unless P=NP. This indicates that there is no feasible algorithm for finding the exact core, which is equivalent to finding the union of all maximal whiskers. Then, we propose three heuristic

algorithms for finding an approximate core, and will empirically evaluate their performance later in Section 2.1.4.

NP-Completeness

Define NAE-3-SAT as the problem of determining whether there exists a truth assignment for a 3-CNF Boolean formula such that each clause has at least one true literal and at least one false literal (i.e. literals in each clause are not all equal). Then, we have the following well-known theorem:

Theorem 2.1.9. NAE-3-SAT *is* NP-*complete* [25].

Now, define WHISKER as the problem of determining whether there exists a whisker in a given weighted undirected graph. We will formally prove that WHISKER is an NP-complete problem by constructing a polynomial-time reduction from NAE-3-SAT.

Theorem 2.1.10. WHISKER *is* NP-*complete.*

Proof Sketch. See Appendix for details. Given an instance of the WHISKER problem, we can guess a solution and verify in linear time whether it is indeed a whisker, thus WHISKER $\in$ NP.

Consider a given 3-CNF Boolean formula with c clauses and n variables. A weighted graph G^* can be constructed where $2n$ vertices are arranged in two columns of n vertices each, corresponding to the literals $\{x_i, \bar{x}_i | 1 \leqslant i \leqslant n\}$, and each vertex is connected to every other vertex by a weighted edge. Note in particular that the size of any cut in G^* has been generalized to the weighted sum of the cut edges. Clearly, this graph can be constructed in polynomial time.

Taking the edge weights of G^* as a function of ε and δ, where ε and δ are small positive numbers, it is guaranteed that all whiskers in G^* are of size n and come from selecting one vertex from each row. Then, with ε and δ satisfying

$$\frac{n(1-\varepsilon)}{(n-\varepsilon)\,(cn^2-2c+2)} < \delta \leqslant \frac{n(1-\varepsilon)}{(n-\varepsilon)\,(cn^2-2c)} \tag{3}$$

for the given c and n, the true literals of a not-all-equal assignment for the formula correspond to the vertices of a whisker in G^*, and the vertices of a whisker in G^* also correspond to the true literals of a not-all-equal assignment for the formula. Therefore, we have established a one-to-one correspondence between not-all-equal truth assignments and whiskers, that is, a weighted graph can be constructed for a given 3-CNF Boolean formula such that whiskers can be found in the graph if and only if the formula is not-all-equal satisfiable. Clearly, NAE-3-SAT reduces to WHISKER in polynomial time, thus, WHISKER is NP-complete. □

Define U-WHISKER as the problem of determining whether there exists a whisker in a unweighted undirected graph. Now, we prove that U-WHISKER is also an NP-complete problem by constructing a polynomial-time reduction from NAE-3-SAT.

Theorem 2.1.11. U-WHISKER *is* NP-*complete.*

Proof Sketch. See Appendix for details. Given an instance of the U-WHISKER problem, we can guess a solution and verify in linear time whether it is indeed a whisker, thus U-WHISKER $\in$ NP. As shown above in the proof of Theorem 2.1.10, after we get G^*, we can enlarge the bipartite graph such that all edge weights become integers, while each vertex is replaced by a clique. The weights of all edges, except those between the literals and their negations, are at least

$(k-1)(n-1)$, since the original weights are $1/(n-\varepsilon)$, where ε is a small positive number.

Then, it is guaranteed that every vertex of each clique is connected to at least one edge. In fact, almost every vertex is connected to two edges, and only a small portion of vertices is connected by only one edge. For simplicity, we assume that each clique is of size $k(n-1)$, where k is a large integer and n is the number of vertices of each column in the original graph.

We show that: (1) If a whisker contains only whole cliques, then it must be exactly one clique from each row, (2) If a whisker contains a partial clique, then including the rest of the clique will form another whisker, and (3) A whisker cannot contain partial cliques. Then, we establish a one-to-one correspondence between whiskers and NAE truth assignments. □

2.1.3 Heuristic Algorithms

An intuitive approach to identifying the core is simply to remove maximal whiskers one by one until no more whiskers exist. However, the following claim characterizes the non-exactness and non-uniqueness of this method, which indicate the generic difficulties associated with any algorithm using this approach to find the core structure.

Claim. *Removing maximal whiskers one by one leads to different subgraphs approximate to the exact core, depending on the order in which whiskers are removed.*

Proof. Here, we can still take Fig. 2.3(a) as an example. Assume that sets S and T are both maximal whiskers and that they do not intersect with other maximal

whiskers. If the set S is first removed, we will be left with the set Z of 3 vertices and 7 outgoing edges, which is apparently not a (maximal) whisker. However, if the set T is first removed instead, we will be left with the set X of 5 vertices and 9 outgoing edges, which is not a (maximal) whisker either. In this case, different sets of vertices remain as part of the ultimate subgraph, neither of which belongs to the exact core. Therefore, the approximate core subgraph depends rather crucially on the order in which we remove these maximal whiskers from the graph, which means that it is not necessarily unique. □

Identifying the exact core in both weighted and unweighted graphs has been proved to be NP-complete in Section 2.1.2. Now, we present three heuristic algorithms for finding an approximate core, whose performance on random and real-world graphs will be experimentally demonstrated in Section 2.1.4.

- ◇ **Algorithm 1** (brute-force search). For each ordered pair of vertices, find its minimum cut and remove the smaller component if the cut is suitable.
- ◇ **Algorithm 2**. Extract the giant component and then the giant biconnected component. Replace all degree-two vertices by a single edge and then test the existence of suitable cuts.
- ◇ **Algorithm 3** (flow-based algorithm). For a given threshold value λ, find the largest subgraph with the maximum edge-vertex ratio exceeding λ. Then, test the existence of suitable cuts.

There is no particular order in which whiskers are removed by Algorithm 1. According to the above claim, larger maximal whiskers could be destroyed and the resulting graph is not necessarily unique, depending rather crucially on the order in which Algorithm 1 removes whiskers. Since a series of degree-two

vertices could result in a whisker, Algorithm 2 contracts all degree-two vertices after obtaining the giant biconnected component. Although Algorithm 2 offers a better run-time performance compared to Algorithm 1, it actually encounters the same difficulties as Algorithm 1 does. The three algorithms are all capable of finding an approximate core, but we will focus on Algorithm 2 and Algorithm 3 since they require shorter running time. Empirically, Algorithm 2 works better for sparse networks, while Algorithm 3 works better for dense ones.

Algorithm 3 is also known as a flow-based algorithm, where the original graph is converted into a weighted directed graph. For the new graph, we create two designated nodes as source and sink, plus a vertex/edge node corresponding to each vertex/edge of the original graph. A directed edge connects the source to every edge node with capacity 1 and connects every vertex node to the sink with capacity λ. Also, a directed edge of infinite capacity connects every edge node to each of the two vertex nodes corresponding to the two vertices adjacent to that edge. As shown in Fig. 2.4, a maximum flow algorithm will return three types of minimum cut set: 1) all the edges going into the sink, 2) some edges coming from the source and some edges going into the sink, and 3) all the edges coming from the source. Note that the directed edges of infinite capacity are opposite from others such that they are never counted as cut edges.

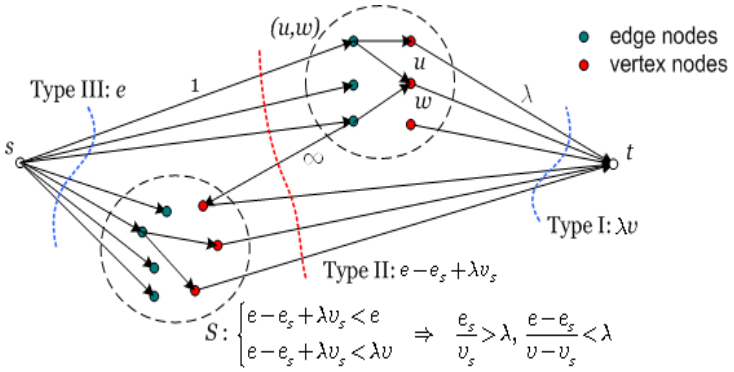


Figure 2.4 Algorithm 3 (flow-based algorithm).

Assume that the original graph has v vertices and e edges. By adjusting the value of λ, we would like to obtain the second type of minimum cut set, where the capacity of minimum cut is given by $e - e_s + \lambda v_s$. Here, e_s and v_s denote the number of edge nodes and vertex nodes in the set S, respectively. Hence, S contains a subset of edge nodes and vertex nodes that satisfy the conditions

$$\frac{e_s}{v_s} > \lambda \qquad \text{and} \qquad \frac{e - e_s}{v - v_s} < \lambda.$$

Clearly, S corresponds to a subgraph with the maximum edge-vertex ratio exceeding λ. Note that this algorithm always returns the largest subgraph meeting the above requirements.

2.1.4 Experimental Results

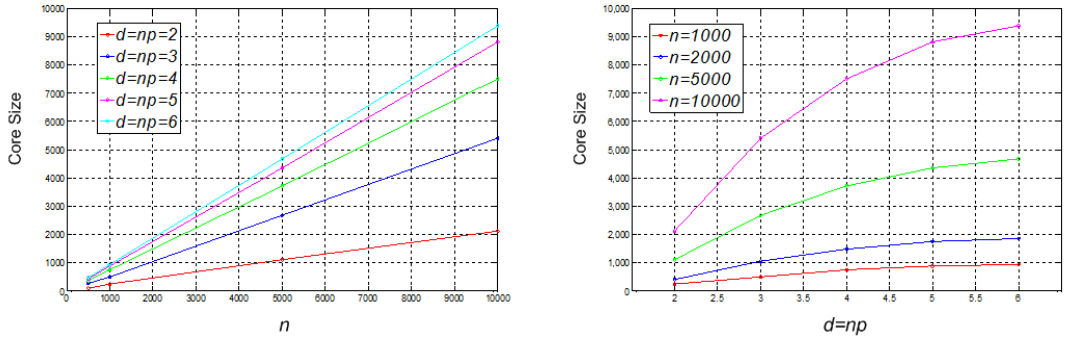
Random Graphs

A random graph $G(n, p)$ can be obtained by starting with a set of n vertices and adding (undirected) edges between them independently with probability $p \in (0, 1)$. Although a random graph does not display any community structure, we can still identify its core using the above algorithms. Table 2.1 shows the size of the cores found by Algorithm 2 and Algorithm 3 in selected random graphs. When p is relatively small, $G(n, p)$ is sparse with low edge-vertex ratio, where Algorithm 3 fails to find an approximate core. In this case, Algorithm 2 can positively identify an approximate core. When p is close to 1, both algorithms are successful in finding an approximate core. As illustrated in Fig. 2.5, the size of the core of $G(n, p)$ grows linearly with $d = np$ for fixed n and logarithmically with n for fixed d. In addition, we observe the existence of phase transition at

$p = 1/n$, above which the core emerges with high probability and below which it emerges with extremely low probability.

Table 2.1 Cores found by Algorithm 2 and 3 in random graphs.

Method	**Core Size**		
	$G(1000, 0.002)$	$G(1000, 0.004)$	$G(10000, 0.0002)$
Algorithm 2	200	761	2151
Algorithm 3	0 (fail)	868 ($\lambda = 1.7$)	0 (fail)



(a) core size as a function of n for fixed d (b) core size as a function of d for fixed n

Figure 2.5 Random Graphs

We conjecture that every $G(n, p)$ with $p > 1/n$ displays the core structure with high probability. For any fixed (large) n, $p = 1/n$ is the threshold for phase transition at which the core structure emerges. The probability and the average size of the core both increase as p grows. For any fixed p, the average size of the core increases as n grows, but the probability of the core remains the same.

Real-World Graphs

Textual Graph. A textual graph consists of vertices representing words and edges representing semantic correlations, which contains information about research topics and areas of interest. We crawl more than 10,000 scientific papers

of the KDD conference from 1992 to 2003 and collect the words of each abstract. A series of pre-processing steps are carried out to simplify the data, which include word stemming, stop-word filtering, and occurrence rate thresholding. Word stemming reduces inflected or derived words to their base form and combines multiple entries of the same word in different tenses. Stop-word filtering removes extremely common but meaningless words, such as and, can, the, will, etc. Occurrence rate thresholding removes extremely rare words occurring in only a small number of abstracts, which exert a trivial effect on the overall community structure.

Pointwise mutual information or log-likelihood ratio can be applied to determine whether there is an edge between each pair of vertices of the textual graph. In this section, we will only discuss the first approach. Pointwise mutual information quantifies the semantic correlation between two words, and we may choose a critical value α above which a strong correlation can be expected. In other words, if the mutual information of two words exceeds α, then an edge exists between them, which indicates a high probability for the two words to occur together. Otherwise, no edge exists between them, which indicates a low probability for the two words to occur together. For a pair of words (i, j) and the threshold value α, there exists an edge between vertex i and vertex j if

$$\log \frac{P(i,j)}{P(i)P(j)} > \alpha,$$

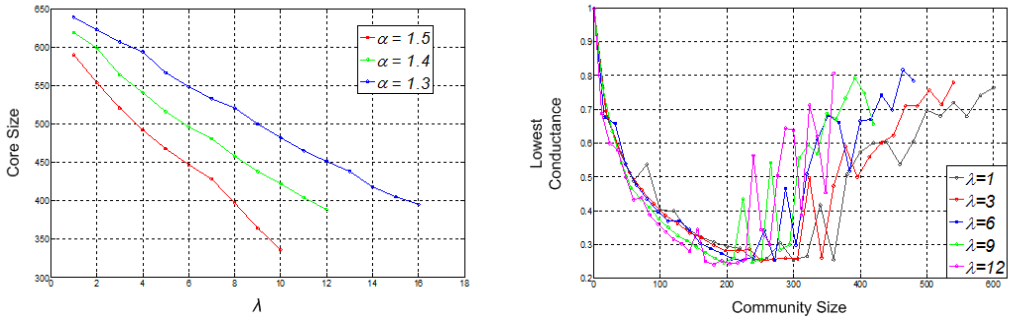
where $P(i)$ and $P(j)$ are the occurrence rate of i and j, respectively, and $P(i, j)$ is the probability of i and j occurring in the same abstract.

For example, the textual graph has 685 vertices and 6,432 edges when $\alpha = 1.4$. Both Algorithm 2 and Algorithm 3 are successful in identifying an approximate core, in which no whiskers exist. In particular, the core returned

by Algorithm 2 is almost identical to that returned by Algorithm 3 when λ is relatively small. Table 2.2 shows the size of the cores found by Algorithm 2 and Algorithm 3 in the textual graph with different values of the correlation threshold α. Higher values of λ will result in a smaller core, and intuitively, higher values of α will result in a graph with less edges and thus a smaller core, as shown in Fig. 2.6(a).

Table 2.2 Cores found by Algorithm 2 and 3 in the textual graph.

Method	**Core Size**		
	$\alpha = 1.3$	$\alpha = 1.4$	$\alpha = 1.5$
Algorithm 2	623	600	554
Algorithm 3	623 ($\lambda = 2$)	599 ($\lambda = 2$)	554 ($\lambda = 2$)



(a) core size as a function of λ (b) community profile of the core ($\alpha = 1.4$)

Figure 2.6 Textual Graph.

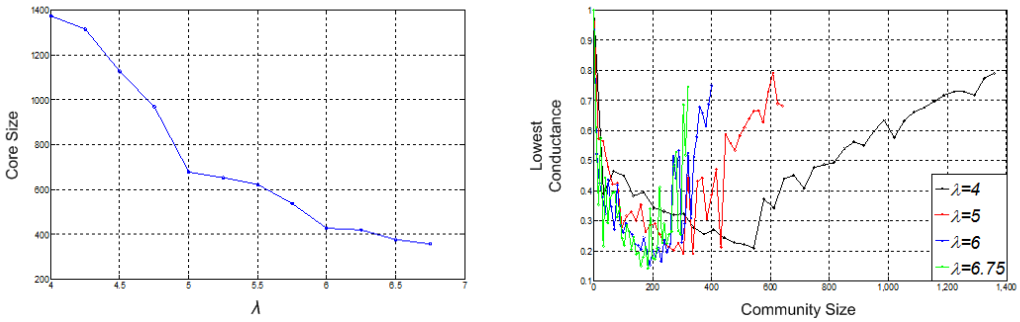
After the approximate core has been extracted from the graph, a simulated annealing algorithm can be performed on the core for finding a subset of a given size with the lowest conductance. As shown in Fig. 2.6(b), the best community of the textual graph possesses a quite large conductance around 0.3, which means the best community has only as many internal edges as cut edges. This exactly corresponds to our intuitive notion that a community should have a significant number of edges connecting it to the rest of the graph. Clearly, the community profile of the core is rather different from what was obtained in [21].

Recall that the best community of most networks examined in [21] displayed an extremely small conductance, typically at the order of 10^{-2}, which means the best community has almost 50 times as many internal edges as cut edges. Moreover, the best community of the textual graph is of size roughly 350 for $\alpha = 1.4$ and $\lambda = 1$, containing more than half of the vertices, which demonstrates the existence of large-scale well-defined communities. As expected, the best community extracted from the textual graph usually specifies a category of research topics or a flow of ideas, and Algorithm 3 is believed to be particularly useful when collaborative filtering is employed to improve the quality of search results.

Coauthor Graph. A coauthor graph reflects the common interests among researchers working in diverse fields, which contains information about authors' reputation and levels of activity. We collect more than 10,000 scientific papers of the KDD conference from 1992 to 2003 and refine the authors' information [1]. Different from the textual graph, the coauthor graph is deterministic with 7,943 vertices and 20,488 edges, where each vertex represents an author and each edge represents a co-authorship. Here, Algorithm 2 is not successful in finding an approximate core by pulling out the giant biconnected component and contracting degree-two vertices. In contrast, Algorithm 3 is able to identify an approximate core, and its size decreases as the threshold value λ increases, as shown in Fig. 2.7(a).

As depicted in Fig. 2.7(b), the community profile of the core of the coauthor graph is rather different from what was obtained in [21]. Recall that the best community of most networks examined in [21] displayed an extremely small conductance, typically at the order of 10^{-2}, which means the best community

[1] `http://www.cs.cornell.edu/projects/kddcup`



(a) size of the core as a function of λ (b) community profile of the core

Figure 2.7 Coauthor Graph

has almost 50 times as many internal edges as cut edges. Here, the best community of the coauthor graph possesses a large conductance about 0.2, which means the best community has only twice as many internal edges as cut edges. This, again, corresponds to our intuitive notion that a community should have a moderate number of edges connecting it to the rest of the graph. Moreover, the best community of the coauthor graph is of size roughly 500 for $\lambda = 4$, containing more than a third of the vertices, which again demonstrates the existence of large-scale well-defined communities.

2.1.5 Conclusion

We investigated large real-world complex networks and proposed an innovative definition of community as opposed to what was generally assumed in previous studies, where communities were thought to be better connected internally than connected with the rest of the network. In fact, a community is more densely connected internally than expected solely from chance, but it is also connected to the rest of the network by a significant number of edges. Further, we defined two auxiliary terms: whiskers and the core. Whiskers were

often interpreted as communities, but they are not the type of community we are interested in here. In contrast, the core exclusively contains the type of community we would like to identify.

Armed with these definitions, we designed a community detection algorithm consisting of two steps: 1) identifying the core in which no whiskers exist, and 2) identifying communities within the core. However, extracting the exact core is rigorously proved to be NP-complete for both weighted and unweighted graphs. The three heuristic algorithms demonstrate their capability of finding an approximate core, and a simulated annealing algorithm is performed on the approximate core to find its best community, i.e. the subset with the lowest conductance, for a given community size. As expected, the network community profile of the core justifies our definition of community and shows the existence of large-scale well-defined communities in various real-world complex networks.

Overlapping communities exist more commonly than disjoint ones in real-world networks, but the community definition provided here does not include overlapping situations explicitly. Thus, a rigorous definition for overlapping communities is required, based on which we can design community detection algorithms and evaluate their performance. In this case, we still have the same notion that a community should not only be more densely connected than expected from randomness, but also be well connected to the rest of the network. Personal communities are another interest of our future research. We expect to find the union of all communities containing a particular vertex, and by taking the intersection of personal communities of different vertices, we can thoroughly understand the network structure from a community perspective.

2.2 Community Detection by the (α, β)-community

Much of the early work on finding communities in social networks was focused on partitioning the corresponding graph into disjoint communities [4, 11, 15, 23, 26–30]. Conductance was often taken as the measure of the quality of community, and algorithms were sometimes restricted to dense graphs [13, 22, 24, 27]. However, to identify well-defined communities in social networks, one needs to realize that an individual may belong to multiple communities at the same time, and is likely to have more connections to individuals outside of his / her community than inside. For example, a person in the theoretical computer science community is likely to have many connections to individuals outside of this community, who may be his / her family and friends, or enroll in his / her institution, or attend his / her religious group. One approach to finding such overlapping communities is that of Mishra et al. [10], in which the concept of (α, β)-community was introduced and algorithms were given for finding an (α, β)-community in dense graphs, provided there exists a champion in the community. A champion of a community is an individual with a bounded number of neighbors outside of the community.

We present a case study on the Twitter network to evaluate the community structure found by many graph partitioning methods, as shown in Fig. 2.8. The left figure gives a fundamental structure with four meaningful communities (blue, red, green, and pink) extracted from their numerous followers (yellow nodes). Some community members are enlarged to highlight the details. Interestingly, the blue one consists of a group of well-known entertainers and the red one consists of a group of active politicians. The right figure shows the four communities obtained by Newman's modularity-based algorithm [4]. By con-

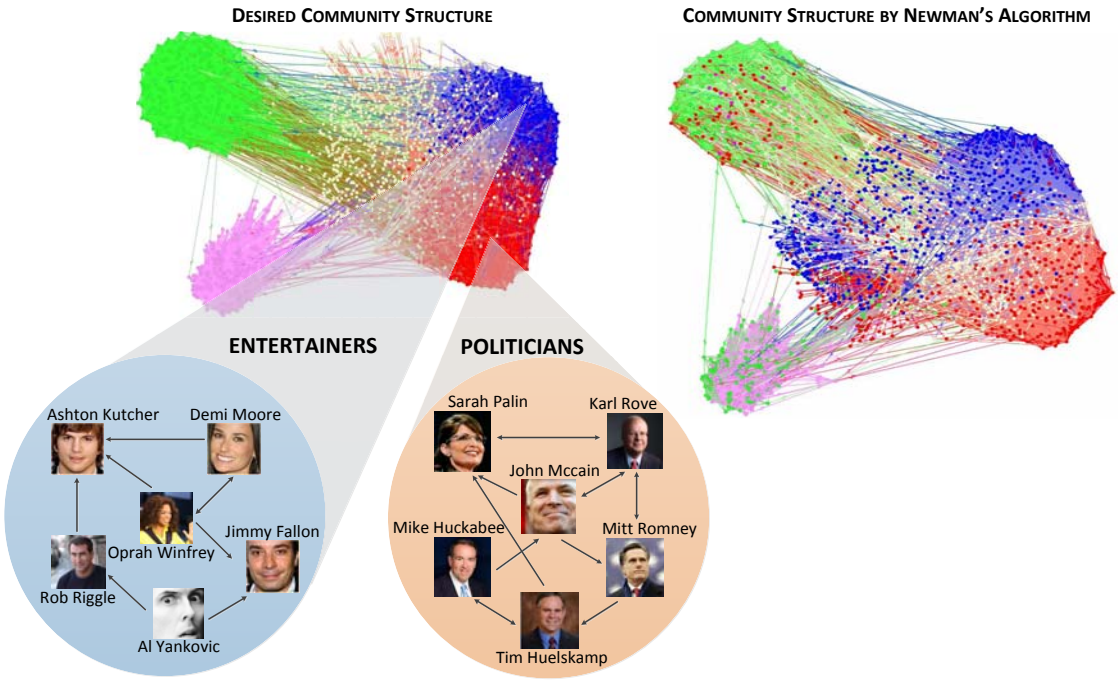


Figure 2.8 Case study on the Twitter network. Traditional community detection methods cannot extract the four meaningful communities from their numerous followers (colored yellow). The blue community consists of entertainers and the red community consists of politicians.

trast, most of the yellow nodes are grouped into one of the four communities, and the communities are heavily blended with each other. Thus, this example reveals that traditional community detection methods fail to discover the desired community structure in many cases.

In this section, we give a definition of (α, β)-community slightly different from that of Mishra et al. [10]. Without fixing the values of α and β, our definition highlights the contrast of internal and external connectivity. We develop a heuristic algorithm based on (α, β)-community that in practice efficiently finds a fundamental community structure. Our algorithm is focused on the difference $\beta - \alpha$ and is thus robust to the specific values of α and β. Further, we thoroughly explore the structure of (α, β)-communities in various large social networks. In a Twitter following network with 112,957 vertices and 481,591

edges, there are 6,912 distinct (α, β)-communities of size 200 with 45,361 runs of the algorithm. These (α, β)-communities are neatly categorized into a small number of massively overlapping clusters. Two (α, β)-communities in the same cluster have significant overlap (>90%), while two (α, β)-communities in different clusters have extremely small (<5%) pairwise resemblance. This leads to the notion of *core*, which is the intersection of a group of massively overlapping (α, β)-communities. Our definition provides an intuitive criterion as to whether to classify a subgraph as a community. The edges connecting each vertex in the community to vertices of the community should be strictly more than those connecting any vertex outside the community to vertices of the community. Further, by taking the intersection of a number of massively overlapping (α, β)-communities, the set of (α, β)-communities which differ by only a few vertices is reduced to an underlying core. Thus, each (α, β)-community contains one of the few cores and some peripheral vertices, and these peripheral vertices are what gives rise to such a large number of (α, β)-communities.

We can extract the core structure by taking the intersection of a group of massively overlapping (α, β)-communities with multiple runs of the algorithm. The number of cores decreases as k increases. For large k, the (α, β)-communities are well clustered into a small number of disjoint cores, and there are no isolated (α, β)-communities scattered between these cores. The cores obtained for a small k either disappear or merge into the cores obtained for a larger k. Further, the cores correspond to dense regions of the graph, and there are no bridges of intermediate (α, β)-communities connecting one core to another. By contrast, the cores found in various random graphs usually have significant overlap among them, and the number of cores does not necessarily decrease as k increases. Extensive experiments demonstrate that the core structure found

in various social networks is indeed due to their underlying social structure, rather than due to high-degree vertices or a particular degree distribution.

The number and average size of cores in the Twitter graph with respect to the community size k are given in Table 2.3. As k increases, some cores disappear due to their small neighborhood (Definition 2.2.4), while others merge into larger ones due to their high closeness (Definition 2.2.5). We explore some interesting questions in this section, for example, what causes many social networks to display the core structure, why (α, β)-communities correspond to well-defined clusters, and why there are no bridges of (α, β)-communities connecting one core to another. A bridge is a sequence of intermediate (α, β)-communities that connect two cores with substantial overlap between adjacent pairs.

The rest of this section is organized as follows. We introduce the definition of (α, β)-community in Section 2.2.1. Then, we prove the NP-hardness of finding an (α, β)-community and present the heuristic (α, β)-COMMUNITY algorithm in Section 2.2.2. In Section 2.2.3, we apply the algorithm to various social and random graphs to demonstrate, explore, and analyze the core structure found in many social networks. Finally, we conclude in Section 2.2.4 with comments on the problems considered and future work.

2.2.1 Preliminaries

The concept of (α, β)-community was proposed by Mishra et al. [10] as a powerful tool for graph clustering and community discovery. In [10], an (α, β)-community refers to a cluster of vertices with each vertex in the cluster adjacent to at least a β-fraction of the cluster and each vertex outside of the cluster adja-

cent to at most an α-fraction of the cluster. Without loss of generality, we adopt a slightly different definition in this section.

Given a subset of vertices $S \subseteq V$, for any $v \notin S$, $\alpha(v)$ is defined as the number of edges between v and vertices of S. Similarly, for any $w \in S$, $\beta(w)$ is defined as the number of edges between w and vertices of S (self-loop counted). Then, we define $\alpha(S) = \max\{\alpha(v)|v \notin S\}$ and $\beta(S) = \min\{\beta(w)|w \in S\}$.

Definition 2.2.1. *Given a graph $G = (V, E)$ with self-loops, a subset of vertices $C \subseteq V$ is called an (α, β)-*community *if each vertex in C is connected to at least β vertices of C (self-loop counted) and each vertex outside of C is connected to at most α vertices of C ($\alpha < \beta$). That is, $\alpha = \alpha(C) < \beta = \beta(C)$.*

Definition 2.2.1 is equivalent to that of [10] where C is a $(\alpha(C)/|C|, \beta(C)/|C|)$-cluster with $\alpha(C) < \beta(C)$. It acknowledges the importance of self-loops: although a maximal clique should intuitively be a community, this cannot be guaranteed without self-loops. An (α, β)-community in a graph G is called *proper* if it corresponds to a non-empty proper subgraph of G.

A maximal clique is guaranteed to be an (α, β)-community since self-loops are counted by Definition 2.2.1. Thus, every graph that is not a clique must contain an (α, β)-community (or, a maximal clique) as a proper subgraph. Starting with any vertex, either it is a proper (α, β)-community or there must be another vertex connected to it. Then, two vertices connected by an edge either form a proper (α, β)-community or there must be a third vertex connected to both. Continue this argument until a proper (α, β)-community is found or all vertices are included in a clique, contradicting the assumption that the graph is not a clique. Thus, we have the following theorem:

Theorem 2.2.2. *A non-complete graph must contain a proper (α, β)-community.*

Proof. Consider a graph $G = (V, E)$. Initially let a set $C = V$ and repeatedly remove a vertex from C with the lowest β value. We will show that either G is complete or C forms a proper (α, β)-community during some iteration.

Assume that C is not a clique. Let v_1 be the first vertex removed from C with $\beta(v_1) = \rho$. Once v_1 has been removed, $\alpha(v_1) = \rho - 1$ since its self-loop is no longer counted. Hence, $C = V - \{v_1\}$ and $\alpha(C) = \alpha(v_1) = \rho - 1$. Assume that C is still not an (α, β)-community at this point, i.e. $\alpha(C) = \rho - 1 \geqslant \beta(C)$. For each vertex $v \in \{u \in C | (u, v_1) \notin E\}$, $\beta(v)$ does not change, and for each vertex $v \in \{u \in C | (u, v_1) \in E\}$, $\beta(v)$ is reduced by one. Then, v_1 is connected to some vertex $v_2 \in C$ that now has the lowest β value and will be removed from C in the next iteration. The removal of v_1 must have reduced $\beta(v_2)$ by one such that $\beta(C) = \beta(v_2) = \rho - 1$.

If the set C does not become an (α, β)-community as we recursively remove vertices in this way, then $\beta(C)$ must be reduced by one during each iteration. Further, if a vertex v_i is removed from C in the ith iteration, $\beta(v_i)$ should be equal to $\rho - (i - 1)$ upon its removal, which means $\beta(v_i) = \rho$ initially and v_i is connected to all the vertices $v_1, v_2, \cdots, v_{i-1}$ removed from C. Thus, if no (α, β)-community is ever found until the last vertex v_n, $n = |V|$, has been removed from C, the graph G is simply a clique. □

Given an integer k and a graph G with self-loops, define k-COMMUNITY as the problem of finding an (α, β)-community of size k in G. Given an integer k and a graph G, define k-CLIQUE as the problem of determining whether there exists a clique of size k in G.

Theorem 2.2.3. *The k-COMMUNITY problem is* NP*-hard.*

Proof. We will show that if k-COMMUNITY is polynomial-time solvable, so is k-CLIQUE, which is a well-known NP-hard problem.

Let $\{k, G = (V, E)\}$ be an input to the k-CLIQUE problem, where the goal is to decide whether G contains a clique of size k. Without loss of generality, assume that G is not a clique and $k \geqslant 3$. Let $n = |V|$ and for each ℓ such that $k \leqslant \ell \leqslant n-1$, construct a graph $H_\ell = (V_\ell, E_\ell)$ as follows:

$$\begin{aligned} V_\ell &= V_{\ell,1} \cup V_{\ell,2}, V_{\ell,1} = \{x_i | 1 \leqslant i \leqslant n+\ell+1\}, V_{\ell,2} = \{y_j | 1 \leqslant j \leqslant \ell+1\}; \\ E_\ell &= \{(x_{i_1}, x_{i_2}) \,|1 \leqslant i_1 < i_2 \leqslant n+\ell+1\} \cup \{(y_{j_1}, y_{j_2}) \,|1 \leqslant j_1 < j_2 \leqslant \ell+1\} \cup \\ &\quad \{(y_j, x_i) \,|1 \leqslant j \leqslant \ell+1, 1 \leqslant i \leqslant \ell-1\}. \end{aligned}$$

H_ℓ contains two cliques of size $n+\ell+1$ and $\ell+1$, where each vertex of the second clique is connected to a fixed subset of $\ell-1$ vertices of the first clique. Let $G_\ell = G^* \cup H_\ell^*$, where G^* and H_ℓ^* are obtained by adding self-loops to all the vertices. Note that G^* and H_ℓ^* are disjoint.

The graph G has a clique of size k if and only if it has a maximal clique of size ℓ, $k \leqslant \ell \leqslant n-1$. Then, we proceed to prove that G has a maximal clique of size ℓ if and only if G_m contains an (α, β)-community of size $n+2\ell+1$. Assume that G contains a maximal clique on the subset $V' \subseteq V$ with $|V'| = \ell$. Let $S = V' \cup V_{\ell,1}$, and clearly, $\beta(S) = \ell$. By the maximality of V', each vertex in $V - V'$ is adjacent to at most $\ell-1$ vertices in V'. Further, by the construction of H_ℓ, each vertex in $V_{\ell,2}$ is adjacent to $\ell-1$ vertices in $V_{\ell,1}$. Thus, S is an (α, β)-community of size $n+2\ell+1$ since $\alpha(S) = \ell - 1 < \beta(S)$.

Now, assume that G_ℓ has an (α, β)-community S of size $n+2\ell+1$. Since the subset S contains at least $(n+2\ell+1) - (n+\ell+1) = \ell$ vertices in $V_{\ell,1}$,

there exists at least one vertex $v \in S \cap V_{\ell,1}$ that is not connected to any vertex in $V_{\ell,2}$. Suppose that S contains k vertices in $V_{\ell,1}$, $\ell \leqslant k \leqslant n+\ell+1$, and thus $\beta(S) \leqslant \beta(v) = k$. If $k < |V_{\ell,1}|$, there exists at least one vertex outside of S adjacent to k vertices in S, leading to $\alpha(S) \geqslant k \geqslant \beta(S)$ which contradicts the definition of (α,β)-community. Hence, $V_{\ell,1} \subseteq S$.

Suppose that there exists some vertex $y_j \in S \cap V_{\ell,2}$, i.e. $|S \cap V_{\ell,2}| \geqslant 1$. Since $|S| - |V_{\ell,1}| = \ell < |V_{\ell,2}|$, at least one vertex in $V_{\ell,2}$ is outside of S. Note that $V_{\ell,2}$ is a clique and each vertex in $V_{\ell,2}$ is connected to $\ell-1$ vertices in $V_{\ell,1}$. Thus, $\beta(S) \leqslant (\ell-1)+|S\cap V_{\ell,2}|$ and $\alpha(S) \geqslant (\ell-1)+|S\cap V_{\ell,2}|$, which contradict the assumption that S is an (α,β)-community. Then, $|S \cap V_{\ell,2}| = 0$ and the remaining ℓ vertices of S are all from V. Recall that $\alpha(S) \geqslant \ell-1$ and there are no edges between V and $V_{\ell,1}$. If $S - V_{\ell,1}$ is not a clique, then $\beta(S) \leqslant \ell-1 \leqslant \alpha(S)$, again leading to a contradiction. Hence, $S - V_{\ell,1}$ is a clique and $\beta(S) = \ell$. $S - V_{\ell,1}$ is also a maximal clique of size ℓ, since $\alpha(S) < \beta(S) = \ell$. Therefore, we have completed the proof by constructing a correspondence between the k-COMMUNITY problem and the k-CLIQUE problem. □

2.2.2 Algorithm

In this section, we give a heuristic algorithm for finding an (α,β)-community of size at least k in a graph $G = (V,E)$. Starting with a random subset $S \subseteq V$ of k vertices, the algorithm proceeds as follows. If $\alpha(S) > \beta(S)$, swap a vertex in S with the lowest β-value and a vertex outside of S with the highest α-value. Each such swap increases the value of $\sum_{v\in S}\beta(v)$ by $-(2\beta-1)+(2\alpha+1) = 2(\alpha-\beta)+2$ if the two vertices are not connected, or by $-(2\beta-1)+(2\alpha-1) = 2(\alpha-\beta)$ if the

two vertices are connected by an edge. Note that $\sum_{v \notin S} \alpha(v)$ may also increase upon each such swap. Since $\sum_{v \in S} \beta(v)$ cannot increase infinitely, the algorithm either returns an (α, β)-community S or reaches a state in which $\alpha(S) = \beta(S)$.

Let $A = \{v \in V - S | \alpha(v) = \alpha(S)\}$ and $B = \{w \in S | \beta(w) = \beta(S)\}$ denote the two subsets of vertices with the highest α-value and the lowest β-value. If $\alpha(S) = \beta(S)$, the algorithm finds a pair of vertices $a \in A$ and $b \in B$ that are not connected, if such a pair exists, and swaps a and b. Since self-loops are counted, the sum $\sum_{v \in S} \beta(v)$ is increased by two, as illustrated in Fig. 2.9.

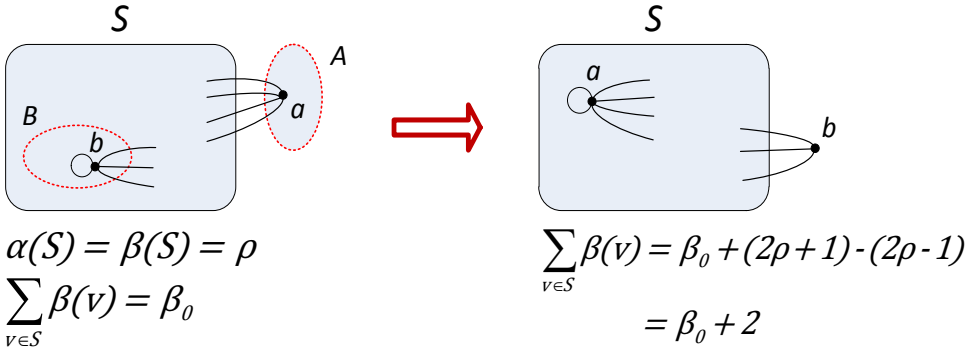


Figure 2.9 The (α, β)-COMMUNITY algorithm.

Then, the condition $\alpha(S) = \beta(S)$ may no longer hold such that the algorithm continues swapping a vertex in S with the lowest β-value and a vertex outside of S with the highest α-value. Again, since $\sum_{v \in S} \beta(v)$ cannot increase infinitely, the algorithm will find either an (α, β)-community S or the case when $\alpha(S) = \beta(S)$ and the sets A and B form a bi-clique. In the latter situation, if a vertex $v \in A$ is not connected to any other vertex in A, adding v to S will increase $\beta(S)$ by one but not increase $\alpha(S)$, thus obtaining an (α, β)-community. Similarly, removing a vertex $w \in B$ that is not connected to any other vertex in B will also produce an (α, β)-community.

Thus, upon termination, the algorithm returns either an (α, β)-community or a subset $S \subseteq V$ where $\alpha(S) = \beta(S)$ and the sets A and B form a bi-clique. Further, neither A nor B has an isolated vertex in the corresponding subgraphs induced by the two sets. Then, we simply add all the vertices in A to S and start the algorithm over. Though we cannot guarantee to find an (α, β)-community due to this latter case, in practice when k is not too small (e.g., $\leqslant 20$), we never run into the bi-clique situation and thus always find an (α, β)-community.

A mathematical description of this (α, β)-COMMUNITY algorithm, along with a subroutine called SWAPPING, is given below. Three corollaries are also given to demonstrate the correctness and proper termination of the SWAPPING algorithm. Their proofs are straightforward and thus omitted from this section.

```
1  S ← a random subset of k vertices
2  while β(S) ⩽ α(S) do
3  |  S ← SWAPPING(G, S)
4  |  A ← {v ∉ S | α(v) = α(S)}
5  |  B ← {v ∈ S | β(v) = β(S)}
6  |  if {(a_i, b_j) ∉ E | a_i ∈ A, b_j ∈ B} ≠ ∅ then
7  |  |  pick such a pair of vertices (a_i, b_j)
8  |  |  S ← (S − {b_j}) ∪ {a_i}
9  |  else if {a_i ∈ A | (a_i, a_k) ∉ E, ∀a_k ∈ A, k ≠ i} ≠ ∅ then
10 |  |  pick such a vertex a_i
11 |  |  S ← S ∪ {a_i}
12 |  else if {b_j ∈ B | (b_j, b_k) ∉ E, ∀b_k ∈ B, k ≠ j} ≠ ∅ then
13 |  |  pick such a vertex b_j
14 |  |  S ← S − {b_j}
15 |  else
16 |  |  S ← S ∪ A
17 return S
```

Algorithm 2.1: (α, β)-COMMUNITY$(G = (V, E), k)$

Corollary 1. *Each iteration of* SWAPPING *strictly increases* $\sum_{v \in S} \beta(v)$.

Corollary 2. SWAPPING *always terminates. When it terminates, swapping any pair of vertices in* A *and* B *will not increase* $\sum_{v \in S} \beta(v)$.

```
while β(S) < α(S) do
    A ← {v ∉ S | α(v) = α(S)}
    B ← {v ∈ S | β(v) = β(S)}
    pick a vertex a ∈ A and a vertex b ∈ B
    S ← (S − {b}) ∪ {a}
return S
```

Algorithm 2.2: SWAPPING$(G = (V, E), S)$

Corollary 3. SWAPPING *returns a subset of vertices* S *with* $\beta(S) \geqslant \alpha(S)$.

2.2.3 Experimental Results

In this section, we conduct experiments on a number of social and random graphs to demonstrate, explore, and analyze the core structure.

Twitter Graph

The Twitter dataset [31] was crawled in 2009 from the online social networking and microblogging service `Twitter.com` that contains friendship links among a group of Twitter users. Each vertex represents a Twitter user account, and each edge represents a following relation. For simplicity, we consider this graph as undirected, ignoring the direction of the edges and combining multiple edges with the same endpoints. Further, we remove the isolated and degree-one vertices from the graph to discard the insignificant outliers. This results in a smaller graph of 112,957 vertices and 481,591 edges with average degree 8.52.

Starting with random subsets of size k, the (α, β)-COMMUNITY algorithm is applied to the Twitter graph for finding (α, β)-communities. Theoretically, this algorithm is not guaranteed to terminate within a reasonable amount of

running time, thus we specify an upper bound (e.g., 1,000) on the number of iterations. However, in practice, we rarely observe the case of not finding any (α, β)-community within 1,000 iterations of the algorithm.

In most cases, 500 runs of the algorithm return 500 (α, β)-communities. However, more than 45,000 runs of the algorithm return only 6,912 distinct (α, β)-communities for $k = 200$, which gives an estimate of the number of (α, β)-communities in the Twitter graph. Surprisingly, these (α, β)-communities are all clustered into a small number of disjoint groups. Two (α, β)-communities in the same group share a resemblance higher than 0.9 and differ by only a few vertices, while two (α, β)-communities in different groups share a resemblance lower than 0.06. Here, the pairwise resemblance (a.k.a. Jaccard index) $r(A, B)$ between two sets A and B is defined as:

$$r(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Thus, the (α, β)-communities form a "core" overlapping structure rather than a "chain" overlapping structure, as shown in Fig. 2.10. Further, the intersection of the (α, β)-communities in each group has an over 75% resemblance with every single (α, β)-community in that group. At $k = 200$, all the 6,912 (α, β)-communities found in the Twitter graph cluster into 4 "cores". The "cores" are disjoint from each other and correspond to dense regions of the graph. In contrast to what we would have expected, there are no isolated (α, β)-communities scattered between these densely-clustered "cores".

For a group of massively overlapping (α, β)-communities, we define the *core* to be the intersection of those (α, β)-communities. The number of cores can be determined by computing the resemblance matrix of all the (α, β)-communities. Then, the (α, β)-communities can be categorized in a way that any two (α, β)-

(a) Core (b) Chain

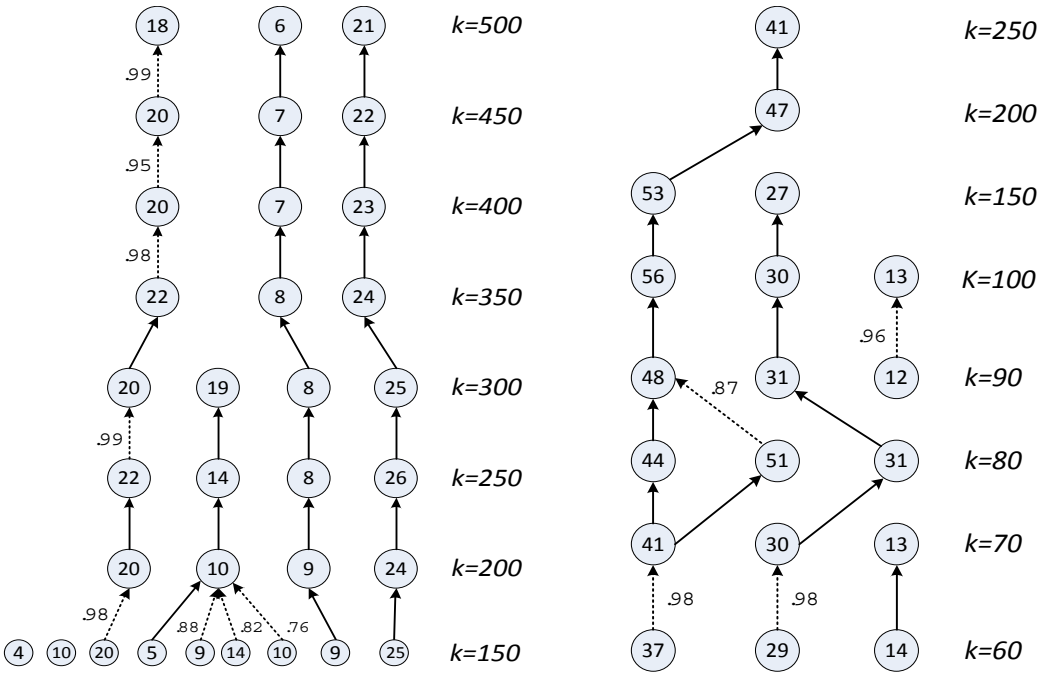
Figure 2.10 The overlapping structure.

communities in the same category are similar to each other. A pairwise resemblance is considered sufficiently large if it is greater than 0.6, while in practice we frequently observe resemblance greater than 0.9. Thus, the cores can be obtained by taking the intersection of all the (α, β)-communities in each category. The number of cores is simply the number of blocks along the diagonal of the resemblance matrix. The number and average size of cores in the Twitter graph with respect to the community size k are given in Table 2.3.

Table 2.3 Cores in the Twitter graph.

k	25	50	100	150	200	250	300	350	400	450	500
number of cores	221	94	19	9	4	4	4	3	3	3	3
average core size	23	45	73	112	151	216	276	332	364	402	440

Observation. The number of cores decreases as the size k increases. This number becomes relatively small when k is large, and will eventually decrease to one as k further increases. Thus, (α, β)-communities are well clustered into a small number of cores before gradually merging into one large core. For example, the (α, β)-communities are clustered into 9 cores for $k = 150$ and 4 cores for $k = 200$, where the cores are disjoint in both cases. As k increases, the cores obtained for a small k either disappear or merge into the cores obtained for a larger k. A layered tree diagram is given to illustrate this phenomenon in Fig. 2.11(a).



(a) Twitter (b) Slashdot

Figure 2.11 The tree diagram for Twitter and Slashdot. (Each circle represents a core obtained for a given k, in which the integer denotes its β value. Each dotted arrow represents a partial merge with the fraction of overlap labeled, and each solid arrow represents a full merge.)

Each level in the tree diagram contains the cores obtained for the corresponding size k. For a pair of cores in adjacent levels, a directed arrow is added from lower to upper level if they have significant overlap, that is, a substantial fraction (e.g., 60%) of vertices in the lower-level core is contained in the upper-level core. If this fraction of overlap is smaller than one, a dotted arrow labeled with the fraction is added to represent a partial merge. Otherwise, a solid arrow is added to represent a full merge. As shown in Fig. 2.11(a), the fraction of overlap is close to one as we move up the tree. Thus, a lower-level core is (almost) entirely merged into an upper-level core.

The definition of (α, β)-community allows a community to have more edges connecting it to the rest of the graph than those connecting within itself. Empir-

ically, there are many more vertices outside of an (α, β)-community, and thus the cut edges are almost always more than the internal edges. This definition provides an intuitive criterion as to whether to classify a subgraph as a community. The edges connecting each vertex in the community to vertices of the community should be strictly more than those connecting any vertex outside the community to vertices of the community. Further, by taking the intersection of a number of massively overlapping (α, β)-communities, the set of (α, β)-communities which differ by only a few vertices is reduced to an underlying core. Thus, each (α, β)-community contains one of the few cores and some peripheral vertices, and these peripheral vertices are what gives rise to such a large number of (α, β)-communities.

Analysis. One question is what causes the Twitter graph to display this core structure, and further, why the graph shows only a small number of disjoint cores for a large size k. As shown later, this is due to the fact that an underlying social structure, as opposed to randomness, exists in the Twitter network. To take a closer look into this, we simplify the Twitter graph by removing low-degree vertices, i.e. vertices of degree lower than 19, and then obtain a smaller graph with 4,144 vertices and 99,345 edges. The minimum β value for most (α, β)-communities is 19, thus this will discard the less important low-degree vertices without destroying the fundamental structure. The (α, β)-COMMUNITY algorithm is applied to this simplified graph for $k = 200, 250, 300, 350, 400$, and we obtain exactly two disjoint cores in each case. For any two adjacent levels in the corresponding tree diagram, the two lower-level cores are completely contained in the upper-level cores. One reason for such a small number of cores could be that the vertices in the two cores are more "powerful" in pulling other vertices toward them. If we remove the two cores from the graph and repeat

the experiment for $k = 200$, then (α, β)-communities are no longer clustered and form a large number of scattered communities.

Another question is why there are exactly two cores in the simplified graph. Define S_1 and S_2 as the two cores obtained for $k = 200$. Then, S_1 corresponds to a fairly dense subgraph with 156 vertices and 3,029 edges, in which the minimum degree is 23 and the average degree is 38.8. S_2 has 159 vertices and 2,577 edges, in which the minimum degree is 19 and the average degree is 32.4. Surprisingly, there are only 105 cross edges between S_1 and S_2, while 110 (70%) vertices of S_1 and 100 (63%) vertices of S_2 are not associated with any cross edge. Thus, S_1 and S_2 correspond to two subsets of vertices that are densely connected internally but sparsely connected with each other. As a result, they are returned as the cores of two groups of massively overlapping (α, β)-communities.

Disappear and Merge. We have observed that, in the Twitter graph, a core obtained for some k disappears from the tree diagram as k increases, and two cores obtained for some k merge into a larger core as k increases. By examining these interesting phenomena, we discover that the disappearance of a core is possibly due to its small effective neighborhood, and the merging of two cores is possibly due to their high closeness. Now, we give the following definitions:

Definition 2.2.4. *The* neighborhood *of a core S is defined as a subset of vertices that are more closely connected to S than any other core.*

The neighborhood of a core can be determined by an iterative process. Any vertex with more connections to one core than any other must belong to the neighborhood of that core. Thus, these vertices can be associated with some core in the first iteration, and we call them tier-1 neighbors. Then, any vertex with more connections to one core and its tier-1 neighbors should also belong to

the neighborhood of that core. Thus, these vertices can be associated with some core in the second iteration, and we call them tier-2 neighbors. This process can be recursively performed until no more vertices can be categorized into any neighborhood.

Definition 2.2.5. *The* closeness *between two cores S_1 and S_2 is defined as their cross-edge density, i.e.*

$$c(S_1, S_2) = \frac{|\{(v, w) \in E \mid v \in S_1, w \in S_2\}|}{|S_1| \cdot |S_2|},$$

where $|S_1|$ and $|S_2|$ denote the number of vertices in S_1 and S_2. This is also an alternate definition of the conductance of a cut.

If a core has a small neighborhood, then there are many low-degree vertices in the neighborhood that do not contribute to the SWAPPING algorithm. Thus, vertices in an adjacent neighborhood are likely to be swapped in, since they may also have a large number of connections to the core and its neighborhood. As the adjacent neighborhood becomes dominant in the algorithm, the vertices in the starting subset are gradually replaced by the vertices in that adjacent neighborhood. Then, the algorithm converges to the corresponding core, causing the initial core to disappear. We notice that the adjacent neighborhood to which the algorithm converges is usually much larger than the small neighborhood of the initial core.

Further, we observe that two cores with comparative size of neighborhood combine to form a larger core as k increases. In such cases, these two cores are very close to each other, and the strong interconnection between them becomes dominant such that they merge rather than disappear, even if they both have small neighborhood. Thus, two cores with high closeness value will merge to form a larger core as k increases.

An example is given in Fig. 2.12 to illustrate the disappearance and merging of cores in the Twitter graph. We obtain 8 cores for $k = 150$. Two cores have fairly small neighborhood, and thus disappear as k increases to 250. Three cores have comparative size of neighborhood and significantly high pairwise closeness, and thus merge to form a larger core as k increases to 250. Hence, we obtain 4 cores for $k = 250$. Further, as k increases from 250 to 350, two cores merge and we obtain three cores. One core has a relatively small neighborhood compared with the others. As k continues to increase to 500, this core eventually disappears.

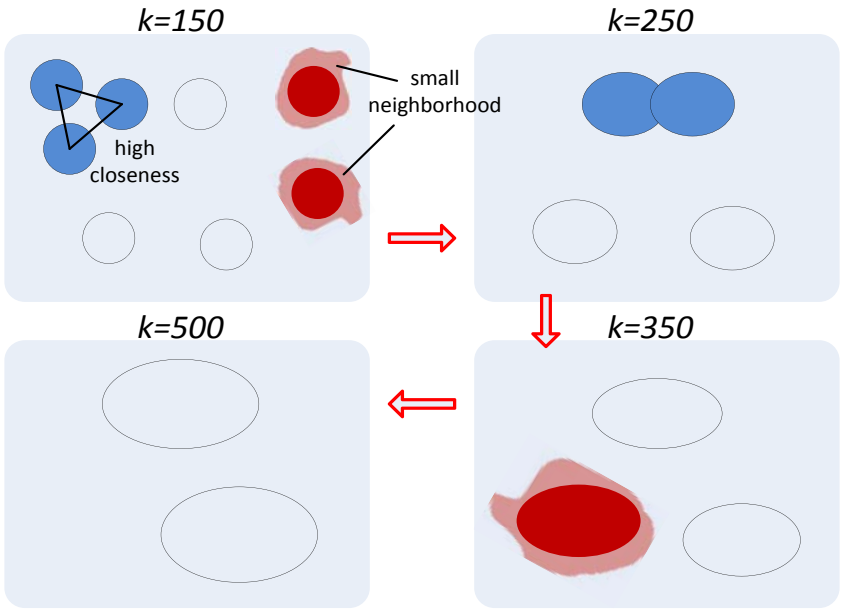


Figure 2.12 The disappearing and merging of cores in the Twitter graph. The disappearing cores are colored red and the merging cores are colored blue.

Bridge. A *bridge* between two cores S_1 and S_m is a sequence of intermediate (α, β)-communities $S_2, \cdots, S_{m-1}$, where the pairwise resemblance is large between adjacent subsets but small between the first and last subsets (e.g., $r(S_1, S_m) < 0.3$ and $r(S_i, S_{i+1}) > 0.6$ for all $i \in \{1, 2, \cdots, m-1\}$). The *length* of the bridge is thus given by $m - 1$. Recall that for $k = 200$, (α, β)-communities are all clustered into 4 disjoint cores, and no bridge is detected between any two

cores. However, the possible bias of our algorithm might prevent a bridge from being found in the Twitter graph. Then, the following experiments are designed to determine whether there exists a bridge.

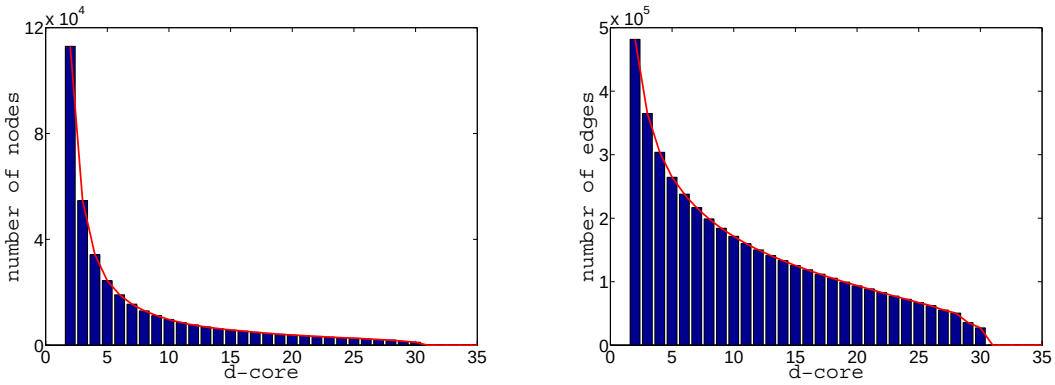
Select any two cores obtained for $k = 200$ and perform the following steps repeatedly. Randomly pick r vertices from one core and $200 - r$ vertices from the other to form an initial subset of size 200, and apply the (α, β)-COMMUNITY algorithm to this subset. If every run returns an (α, β)-community substantially overlapping with one core but disjoint from the other, then it suggests that there does not exist any bridge between the two cores. With 100 runs of the algorithm, 99 return such an (α, β)-community, and only one returns an (α, β)-community C that contains 95.54% of one core A and 26.22% of the other core B. However, no other intermediate (α, β)-communities can be found between B and C using the same approach, which demonstrates the absence of bridge.

Another approach to finding a bridge is to search for (α, β)-communities that fall between cores. Generate random subsets of size 200 and run the (α, β)-COMMUNITY algorithm repeatedly. After 4 disjoint cores have been obtained with 500 runs of the algorithm, (α, β)-communities returned by another 45,361 runs are compared with the 4 cores to check whether there is any intermediate (α, β)-community. This approach is also useful for estimating the total number of (α, β)-communities of a given size. No intermediate (α, β)-communities are found, however, only 6,912 distinct (α, β)-communities are obtained, which indicates a relatively small number of (α, β)-communities of size 200 and / or a possible bias of our algorithm that favors some communities over others.

Overall, these experiments have suggested that there is no bridge between cores, that is, there is no sequence of intermediate (α, β)-communities that con-

nect two cores with substantial overlap between adjacent pairs. The absence of bridge demonstrates the underlying social structure of the Twitter network with (α, β)-communities neatly clustered into a few disjoint cores.

Degree Core. We conduct experiments on the same Twitter dataset using the degree core method. When $d = 9$, it returns one connected subgraph of 11,133 nodes and 184,146 edges. When $d = 20$, it returns one connected subgraph of 3,835 nodes and 93,533 edges. When $d = 30$, it returns one connected subgraph of 1,127 nodes and 27,344 edges. The degree core method always identifies one connected subgraph of high-degree vertices as community, as shown in Fig. 2.13.



(a) # nodes vs. d-core (b) # edges vs. d-core

Figure 2.13 The degree core method.

This means, while degree cores tend to identify subsets of high-degree vertices as communities, the concept of (α, β)-community highlights more the contrast of inter- and intra-connectivity. Our analysis is robust to the specific values of α and β, in particular, we believe the positive difference $\beta - \alpha$ gives a strong intuitive indication of community, rather than their absolute values. The greater $\beta - \alpha$ is, the better. This concept gives a natural type of community that we

are interested in. I don't have to be a star to belong to some community, but I should belong to this community if I have (many) more connections inside this community than anybody outside this community does.

Slashdot Graph

Slashdot is a technology-related news website known for its professional user community. The website features contemporary technology-oriented news submitted by users and evaluated by editors. Slashdot introduced the Slashdot Zoo feature in 2002, allowing users to tag others as friends or foes. The social network based on common interest shared by Slashdot users was obtained and released by Leskovec et al. [27] in February 2009.

The Slashdot graph has 82,168 vertices and 504,230 edges, with an average degree of 12.3. The (α, β)-COMMUNITY algorithm is applied to this graph and the statistics are given in Table 2.4. Our heuristic algorithm discovers a core structure similar to that of Twitter. As in the Twitter graph, the number of cores decreases as the community size k increases and becomes relatively small for large k. The cores found in the Slashdot graph are almost disjoint from each other, with few edges connecting in between, and they correspond to dense regions of the graph. Thus, this suggests that (α, β)-communities are well clustered into a small number of disjoint cores for large k. For example, (α, β)-communities are clustered into three nearly disjoint cores for $k = 100$, where only 171 edges connect the two cores of size 93 and 100 with 2,142 and 1,105 internal edges, respectively. As k increases, the cores obtained for a small k either disappear (due to their small neighborhood), or merge into the cores obtained for a larger k (due to their high closeness). A layered tree diagram is given to

illustrate this phenomenon in the Slashdot graph, as shown in Fig. 2.11(b).

Table 2.4 Cores in the Slashdot graph.

k	30	40	50	60	70	80	90	100	150	200	250
number of cores	29	10	3	3	3	3	3	3	2	1	1
average core size	25	33	41	53	62	72	85	97	148	197	244

Coauthor Graph

The Coauthor dataset was crawled from the e-print arXiv that contains scientific coauthorship between authors of the papers submitted to the hep-ph archive [32]. If author i coauthors a paper with author j, there is an undirected edge between vertex i and vertex j in the corresponding graph. If a paper has k authors, then there is a clique of size k in the graph. The dataset contains papers published between January 1993 and April 2003 (124 months), starting within a few months of the inception of arXiv, and thus it represents essentially the complete history of the hep-ph archive.

The arXiv hep-ph Coauthor graph contains 12,006 vertices and 118,489 edges, with an average degree of 19.7. Since there exists a clique of size 239 in this graph, the (α, β)-COMMUNITY algorithm returns this clique or a substantial part of it as a core for $k \geqslant 200$. After removing this clique, we obtain a similar core structure to that of Twitter and Slashdot. We apply the algorithm and obtain the statistics as shown in Table 2.5.

Table 2.5 Cores in the Coauthor graph.

k	30	40	50	60	70	80	90	100	150	200	250
number of cores	64	49	41	45	32	31	25	30	32	20	18
average core size	36	45	52	63	73	81	88	101	146	182	223

Citation Graph

The Citation dataset was crawled from the e-print arXiv that contains 421,578 citation links among a collection of 34,546 papers in the hep-ph archive [33, 34]. If paper i cites paper j or vice versa, then there is an undirected edge between vertex i and vertex j in the corresponding graph. This dataset was originally released in the KDD Cup 2003 [33], and represents essentially the complete history of the hep-ph archive.

The Citation graph has 34,546 vertices and 420,877 edges, with an average degree of 24.4. Again, the (α, β)-COMMUNITY algorithm is applied to this graph and the statistics are shown in Table 2.6. In this graph, we again discover a core structure similar to that of Twitter, Slashdot, and Coauthor. The Citation graph contains more cores than other social graphs for the same value of k. There are 4 disjoint cores for $k = 900$, and as k continues to increase, the number of cores eventually decreases to one as in the other social graphs.

Table 2.6 Cores in the Citation graph.

k	30	40	50	60	70	80	90	100	150	200	250
number of cores	168	123	90	76	64	57	47	43	33	35	28
average core size	28	38	47	55	65	75	84	93	139	182	223

Random Graphs

A similar set of experiments can be performed on random graphs to demonstrate the existence of core structure in various social networks. The comparison of the results confirms that the structure we have found in many social graphs is more than just a random artifact.

First, we generate a random graph according to the $G(n,p)$ model with $n = 112,957$ and $p = 8.52$ (those of the Twitter graph). This graph contains 597,674 edges (self-loop counted), which are also similar to that of the Twitter graph. However, conducting the same experiment on this graph reveals a completely different structure from what we have seen in social graphs. The (α,β)-COMMUNITY algorithm is employed to find 500 (α,β)-communities of size 30 to 300. For each size, the 500 obtained (α,β)-communities have little overlap (less than 5% in most cases), and are scattered all over the graph where no massively overlapping clusters can be found. We observe that $\alpha = 1$ and $\beta = 2$ for each (α,β)-community in this random graph, as opposed to those as large as 20 in the Twitter graph. Thus, random subsets are extracted from $G(n,p)$ which are not even connected, implying the absence of an underlying social structure.

An interesting question is whether high-degree vertices lead to the massively overlapping clusters found in the Twitter graph. To answer this question, we generate random d-regular graphs with 4,144 vertices (that of the Twitter graph with low-degree vertices removed) for a wide range of values of d. Recall that the lowest β value is 19 for most (α,β)-communities in the Twitter graph, thus removing vertices of degree lower than 19 does not destroy the fundamental structure of the graph. For each value of d, the algorithm still returns scattered (α,β)-communities with little overlap among them. Thus, high-degree vertices are not the primary reason for such few number of cores in the Twitter graph.

Another question is whether a particular degree distribution of the Twitter graph leads to the massively overlapping clusters. To answer this question, we conduct similar experiments on randomly generated graphs with 4,144 vertices and a given degree distribution (e.g., power-law). There are several ways to

generate random graphs with a given degree distribution, two of which give the same distribution as that of the Twitter graph while the third gives a power-law distribution.

Uniform model. Given the degree distribution, we place edges by selecting vertices uniformly at random. As a result, high-degree vertices are not as densely connected as in the Twitter graph. This uniform model displays the same behavior as the $G(n,p)$ model for small (α,β)-communities. As the size k increases, (α,β)-communities gradually overlap with each other. Cores can be extracted from the graph, but they also have significant overlap among them.

Further, most high-degree vertices are contained in the cores as expected. For example, consider the two cores obtained for $k = 450$. One core is of size 172, containing 93% of the vertices of degree higher than 200 and 63% of those of degree higher than 150. The other core is of size 351, containing 100% of the vertices of degree higher than 200 and 84% of those of degree higher than 150.

Proportional model. Given the degree distribution, we place edges by selecting vertices with probability proportional to their degree. As a result, high-degree vertices are densely connected, and for $k \geqslant 150$, there is only one core returned by the algorithm with 200 (α,β)-communities. Further, almost all high-degree vertices are contained in that core. For example, the core is of size 125 for $k = 200$, containing 94% of the vertices of degree higher than 200 and 73% of those of degree higher than 150. The core corresponds to the dense region of the graph due to the way the edges are placed, in which high-degree vertices are more likely to be selected.

Preferential attachment model. We first create a clique of small size (e.g., 5),

then recursively add a new vertex and randomly pick some of the existing vertices to be its neighbors with probability proportional to their degree. Thus, the resulting graph displays a power-law degree distribution, different from that of the Twitter graph. For each size from 50 to 300, the (α, β)-COMMUNITY algorithm returns a small number of cores with substantial overlap among them. In contrast to what we have observed in the Twitter graph, the number of cores steadily increases with the size k. For example, we obtain 7 cores for $k = 90$ and 11 cores for $k = 250$.

According to these experiments, random graph models do not produce well-defined clusters as social graphs do. The cores found in random graphs usually have significant overlap among them, and correspond to dense regions due to the way the graph was generated. This demonstrates that the core structure displayed by various large social networks is indeed due to the existence of underlying social structure of those networks.

2.2.4 Conclusion

In many social networks, (α, β)-communities of a given size k are well clustered into a small number of disjoint cores, each of which is the intersection of a group of massively overlapping (α, β)-communities. Two (α, β)-communities in the same group share a significant overlap and differ by only a few vertices, while the pairwise resemblance of two (α, β)-communities in different groups is extremely small. The number of cores decreases as k increases and becomes relatively small for large k. The cores obtained for a small k either disappear or merge into the cores obtained for a larger k. Further, the cores correspond to

dense regions of the graph, and there are no isolated (α, β)-communities scattered between the cores. There are no bridges of (α, β)-communities connecting one core to another. We have explored various large social networks, all of which display the core structure rather than the chain structure.

By constructing random graphs with a power-law degree distribution or the same degree distribution as that of the social graphs, it is demonstrated that neither high-degree vertices nor a particular degree distribution can lead to the core structure displayed in many social networks. The cores found in random graphs usually have significant overlap and are increasingly scattered across the graph as the size k increases, which implies the absence of well-defined clusters in random graphs and verifies the existence of core structure in various social networks.

Our work opens several questions about the structure of large social networks. It demonstrates the successful use of the (α, β)-COMMUNITY algorithm on real-world networks to discover their social structure. Further, our work inspires an effective way of finding overlapping communities and extracting the underlying core structure. We conjecture that, in many social graphs, the vertices inside an (α, β)-community but outside of the corresponding core are actually located in the overlapping regions of multiple communities. Other interesting questions include whether different types of social networks display fundamentally different social structure, how the core structure evolves over time, whether the cores represent the stable backbones of the network, and whether the vertices that belong to multiple communities constitute the unstable regions of the network.

CHAPTER 3

HOW TO IDENTIFY DIFFERENT LEVELS OF SOCIAL INFLUENCE?

3.1 Introduction

The Pareto principle (a.k.a. 80-20 rule) [35] exists almost everywhere. For example, 80% of a country's land is owned by 20% of the population, and 80% of a company's sales revenue comes from 20% of its clients. This is also the case for many social networks. In these networks, there exist two types of users that exhibit different influence and different behavior. For instance, it has been shown that less than 1% of the Twitter users (e.g., writers, entertainers, politicians) produce roughly 50% of the content on the micro-blogging site [1], while the other 99% (e.g., fans, followers, readers) have much less influence and completely different social behavior. Then, an interesting question is: "how do these influential users interact with each other?" Further, influential users are typically followed more than others. For example, Oprah Winfrey has more than 5 million followers, and Barack Obama has more than 7 million. Hence, another interesting question is: "what is the underlying structure between influential users and their followers?"

The problem of community detection has been extensively studied and many algorithms have been proposed, such as cut- and conductance-based methods [11–14], spectral clustering [4,15,16], (α, β)-clustering [10,17], and topic modeling methods [18]. The cut- and conductance-based and spectral clustering methods are usually based on a fundamental assumption that communities have dense internal connections and sparse external connections. (α, β)-clustering methods relax this assumption by allowing communities to have

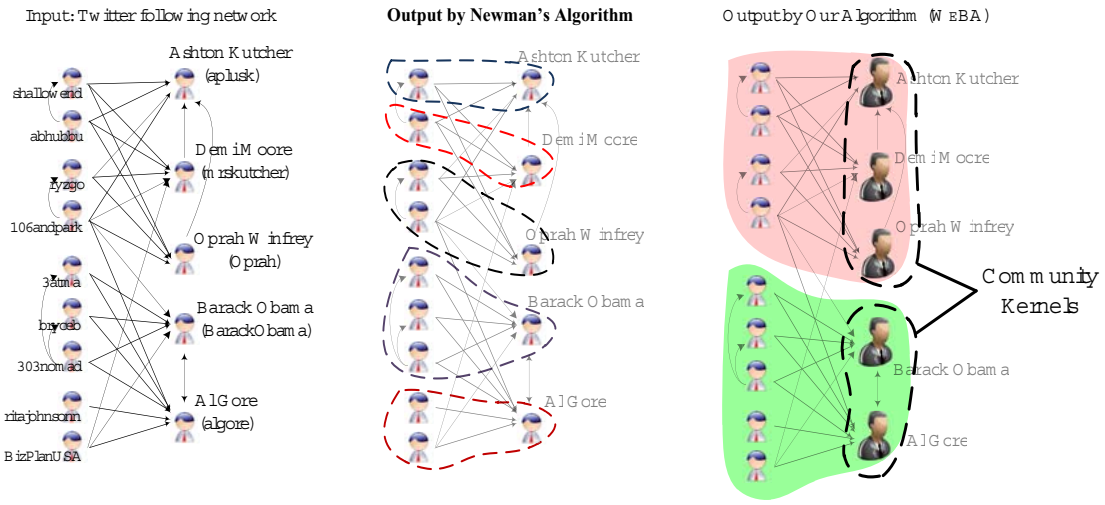
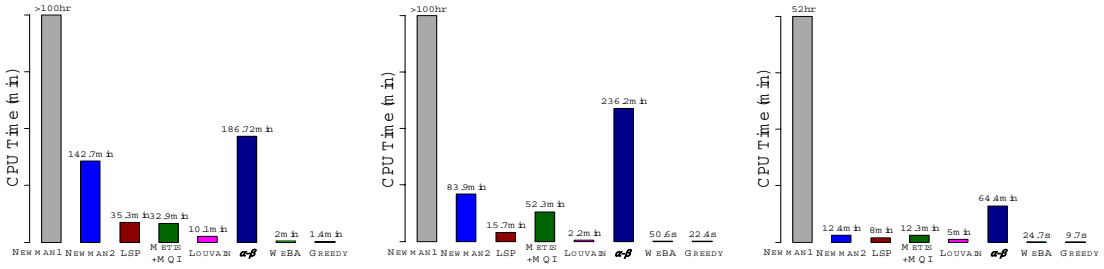


Figure 3.1 An illustration of community kernel detection on the Twitter network. The left figure shows the original Twitter network (three entertainers and two politicians with their followers), the middle figure shows the five communities detected by Newman's algorithm [4], and the right figure shows two community kernels and their corresponding auxiliary communities detected by our algorithm WEBA.

dense external connections. Topic modeling methods are based on statistical analysis of the content information associated with each vertex. However, these methods ignore an important fact that the community structure of influential users is quite different from that of others. Our preliminary statistical analysis shows that the average degree of influential users is almost 10 times more than that of the others in the Twitter network.

To clearly demonstrate this, we present an example from the Twitter network as shown in Fig. 3.1. The left figure is an input of the Twitter following network with three entertainers (Oprah Winfrey, Ashton Kutcher, and Demi Moore) and two politicians (Barack Obama and Al Gore) as well as some of their followers. This input represents a typical network structure with a few influential users connected with the rest of the network via a large number of links. To detect the community structure of this network, we consider Newman's algorithm [4], a state-of-the-art method based on modularity. The middle figure shows the



(a) Coauthor (b) Wikipedia (c) Twitter

Figure 3.2 Efficiency comparison of WEBA and GREEDY with the comparative algorithms (no parallelization).

community structure obtained by Newman's algorithm. We observe that, since there are a large number of connections between each influential user and its followers, Newman's algorithm tends to partition the influential users into different communities and to group them with their respective followers. The lack of ability to distinguish influential users from their numerous followers is a key problem with this method. The right figure shows the community structure obtained by our algorithm WEBA later introduced in Section 3.3. By contrast, this is exactly what one would expect a community detection algorithm to discover: two community kernels (one of entertainers and one of politicians) consist of influential users and two auxiliary communities associated with the kernels. Thus, in this chapter, we refer to this problem as community kernel detection, which includes two parts: (1) how to distinguish influential users (kernel members) from others, and (2) how to detect the community structure (community kernels) among influential users and their respective auxiliary communities.

The problem of community kernel detection has many practical applications, including representative user finding, friend recommendation, network visualization, and marketing. However, this problem is non-trivial and poses a set of challenges. First, it is difficult to identify the truly influential users. One may

consider to use the number of followers as an indicator. Unfortunately, the follower count gives no information about who follows them. Second, it is unclear how influential users interact with each other. Would a politician tend to follow another politician or an actress? Finally, real-world social networks are growing fast with thousands or millions of vertices. It is important to develop an algorithm with high scalability.

Contributions. In this chapter, we formulate the problem of *community kernel detection* in large social networks as two subtasks: identifying influential (kernel) members and detecting the structure of community kernels. We propose two algorithms to complete these two subtasks in a unified approach. The first algorithm is a greedy algorithm based on maximum cardinality search. It can efficiently obtain an approximate solution, but does not have a bounded error. In the second algorithm WEBA, we define and optimize an objective function which explicitly quantifies the detected community kernels. It can efficiently obtain an approximate solution with a small error bound. We validate the effectiveness and efficiency of our algorithms on three large social networks: Coauthor, Wikipedia, and Twitter. Experimental results show that WEBA and GREEDY outperforms eight other state-of-the-art methods for detecting community kernels. In addition, WEBA can efficiently detect community kernels. Fig. 3.2 shows an efficiency comparison of eight algorithms on the three networks. Clearly, WEBA is on average 6–2,000 times faster than the other comparative algorithms.

The rest of this chapter is organized as follows. In Section 3.2, we formally define the problem of community kernel detection. In Section 3.3, we propose two efficient algorithms and provide theoretical analysis. In Section 3.4, we con-

duct data analysis on the Coauthor and Twitter graph to explore their statistical properties. In Section 3.5, we present experimental results to validate the effectiveness and computational efficiency of our algorithms. Finally, we conclude in Section 3.6 with comments on the problems considered and future work.

3.2 Problem Definition

In this section, we first introduce the concept of community kernel and auxiliary community, and then give a formal definition of the problem. A social network can be modeled as a graph $G = (V, E)$, where V is the set of $|V| = n$ entities and $E \subseteq V \times V$ is the set of $|E| = m$ directed / undirected links between entities. Then, we have the following definition:

Definition 3.2.1 (Community Kernel and Auxiliary Community)**.** *Given a graph $G = (V, E)$, ℓ disjoint subsets $\{\mathcal{K}_1, \cdots, \mathcal{K}_\ell\}$ of vertices are called* **community kernels** *if*

$$(1)\forall i, \forall u \in \mathcal{K}_i, \forall v \notin \mathcal{K}_i, |E(u, \mathcal{K}_i)| \geqslant |E(v, \mathcal{K}_i)| \text{ and } |E(\mathcal{K}_i, u)| \geqslant |E(\mathcal{K}_i, v)|.$$

where $E(A, B) = \{(u, v) \in E | u \in A, v \in B\}$ for $A, B \subseteq V$. Further, ℓ associated subsets $\{\mathcal{A}_{\mathcal{K}_1}, \cdots, \mathcal{A}_{\mathcal{K}_\ell}\}$ of vertices are called auxiliary communities *if*

$$(2)\forall i \in \{1, \cdots, \ell\},\ \mathcal{A}_{\mathcal{K}_i} \cap \mathcal{K}_i = \emptyset;$$

$$(3)\forall i, \forall j \neq i, \forall u \in \mathcal{A}_{\mathcal{K}_i},\ |E(u, \mathcal{K}_i)| \geqslant |E(u, \mathcal{K}_j)|;$$

$$(4)\forall i, \forall u \in \mathcal{K}_i,\ |E(\mathcal{A}_{\mathcal{K}_i}, \mathcal{K}_i)| \geqslant |E(\mathcal{K}_i, \mathcal{K}_i)|.$$

For any $i \in \{1, \cdots, \ell\}$, each vertex in $\mathcal{K}_i$ is a kernel member *and each vertex in $\mathcal{A}_{\mathcal{K}_i}$ is an* **auxiliary member**.

A community kernel is disjoint from its auxiliary community. Each member of a community kernel has more connections to / from the kernel than a vertex outside the kernel does. Each member of an auxiliary community has more connections to the associated kernel than to any other kernel. Further, each member of a community kernel is followed by more vertices in its auxiliary community than in the kernel. See Fig. 3.1 for an example.

Consider a set of community kernels $\mathbf{K} = \{\mathcal{K}_1, \cdots, \mathcal{K}_\ell\}$. Each community kernel is closely associated with an auxiliary community, and the corresponding set of auxiliary communities is given by $\mathbf{A} = \{\mathcal{A}_{\mathcal{K}_1}, \cdots, \mathcal{A}_{\mathcal{K}_\ell}\}$. Note that auxiliary communities can overlap with each other.

Community kernels and their auxiliary communities can be interpreted in different ways for different networks. For example, in a coauthorship network, a community kernel can be a group of senior professors in a certain research area, while its auxiliary community consists of students or junior researchers in the same area. In a Twitter network, a community kernel can be a group of well-known entertainers, while the associated auxiliary community consists of followers of these celebrities. Based on the above concept, we define the following problem of detecting community kernels:

Problem (Community Kernel Detection)**.** *Given a graph $G = (V, E)$, how to identify kernel members and auxiliary members, i.e. $\cup\mathcal{K}_i$ and $\cup\mathcal{A}_{\mathcal{K}_i}$, and how to determine the structure of community kernels, i.e. $\mathbf{K} = \{\mathcal{K}_1, \cdots, \mathcal{K}_\ell\}$?*

Our problem formulation is very different from previous work on community detection. Many algorithms have been proposed for detecting communities in social networks [4, 11, 14, 26, 27], however they ignore the difference among vertices and links. Thus, these algorithms fail to distinguish community ker-

nels from their auxiliary communities. In addition, Ahn et al. [36] categorized links instead of vertices to discover hierarchical community structure. Mishra et al. [10] proposed the concept of (α, β)-community to allow communities to overlap. However, these algorithms do not consider the existence and structure of community kernels.

Observation. Interestingly, community kernels and their auxiliary communities form an unbalanced weakly-bipartite structure. Such a structure can be observed in many real-world social networks, as shown in Table 3.1[1].

Table 3.1 Selected UWB networks.

Networks	d_{21}	d_{11}	d_{22}	d_{12}
Coauthor	14.19	5.34	4.42	0.37
Wikipedia	1689.31	104.22	4.69	0.60
Twitter	110.78	26.78	2.94	0.29
Slashdot	180.90	84.56	10.75	0.64
Citation	76.69	35.81	23.80	0.26
Web	15.21	28.81	1.04	1.69
Amazon	12.98	3.74	4.33	1.44

An ***unbalanced weakly-bipartite*** *(UWB)* structure consists of two disjoint subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that

$$d_{21} > d_{11} > d_{22} \gg d_{12}. \tag{3.1}$$

d_{11} and d_{22} are the average degree of G_1 and G_2, respectively. d_{21} is the average number of edges from G_2 to G_1 per vertex $u \in V_1$, and d_{12} is the average number of edges from G_1 to G_2 per vertex $u \in V_2$. G_1 is considered as a community kernel and G_2 is considered as the auxiliary community associated with G_1,

[1]Coauthor: 822,415 authors and 2,928,360 co-author links; Wikipedia: 310,990 editors and 10,780,996 co-editing links; Twitter: 465,023 users and 833,590 following links; Slashdot: 82,168 users and 504,230 friendship links; Citation: 34,546 publications and 420,877 citation links; Web: 325,729 websites and 1,117,563 hyperlinks; Road: 262,111 intersections and 899,792 streets.

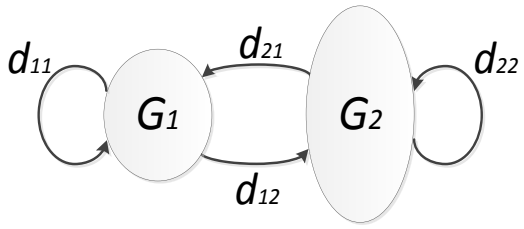


Figure 3.3 A UWB structure.

as shown in Fig. 3.3. Specifically, $d_{ij} = |E(V_i, V_j)|/|V_j|$, $i, j \in \{1, 2\}$, where $E(V_i, V_j) = \{(u, v) \in E \mid u \in V_i, v \in V_j\}$, and (u, v) is an ordered pair of vertices.

3.3 Algorithms

In this section, we propose two algorithms for detecting community kernels in large social networks. We first give a simple greedy algorithm based on maximum cardinality search, and then an efficient weight-balanced algorithm based on local search. Given an undirected graph $G = (V, E)$, let n and m be the number of vertices and edges. Note that the algorithms for undirected graphs can be easily extended to those for directed graphs.

3.3.1 Basic Principles

Existing cut- and conductance-based algorithms (e.g., [4, 11–15, 26, 27, 37]) cannot distinguish kernel members from auxiliary ones. In these methods, edges between different types of vertices are treated the same way. Thus, the large number of links from auxiliary members to kernel ones may dominate the results of community detection.

An intuitive method to distinguish kernel members from others is to first perform a link analysis algorithm (e.g., degree ranking, PageRank [38], HITS [39]) on the network to find the "influential" vertices, and then apply a cut- or conductance-based community detection algorithm to those vertices only. In this way, we obtain communities solely based on the link information between influential vertices. However, this approach ignores an important piece of information in the network, that is, the link information between auxiliary and kernel members. For example, in the Twitter network, fans may follow several members of the same kernel (e.g., politicians). This collective following behavior indicates that the target members that are being followed should be grouped in the same community kernel. Thus, the lack of this information prevents the method from finding community kernels.

With these considerations, we propose two algorithms for efficiently finding community kernels in large social networks. Different from existing cut- and conductance-based algorithms in which the goal is to find communities with dense internal connections and sparse external connections, we aim to find communities with dense internal connections but allow them to have dense external connections. Our first algorithm GREEDY is based on maximum cardinality search, which is efficient but does not have a bounded error. Further, we propose a second algorithm WEBA in which we heuristically solve an optimization problem. WEBA satisfies all the requirements for detecting community kernels. We prove its theoretical validity and analyze its error bound. GREEDY and WEBA apply to both undirected and directed graphs. For simplicity, we only provide the pseudocode for the undirected case.

3.3.2 Greedy Algorithm

Consider an undirected graph $G = (V, E)$ with n vertices and m edges. Given a kernel size k, initialize a subset $S \subseteq V$ to be a random vertex $v \in V$. Then, iteratively enlarge S by adding the vertex with the maximum number of connections to S. If there are multiple vertices with the maximum number of connections to S, pick the one with the highest degree. If there are multiple vertices with the highest degree, randomly pick one of them. This subroutine can be executed recursively to find multiple community kernels in the graph. Recall that $E(A, B) = \{(u, v) \in E | u \in A, v \in B\}$ for $A, B \subseteq V$.

Input: $G = (V, E)$ and kernel size k
Output: community kernels $\mathbf{K} = \{\mathcal{K}_1, \mathcal{K}_2, \cdots, \mathcal{K}_\ell\}$
$\mathbf{K} \leftarrow \emptyset$
repeat
  $S \leftarrow$ random $v \in V$
  while $|S| < k$ **do**
    $R^* = \{u \notin S || E(u, S)| = \max\{|E(v, S)|, \forall v \notin S\}\}$
    if $|R^*| = 1$ **then** $S \leftarrow S \cup R^*$
    else $U^* = \{u \in R^* \mid d(u) = \max\{d(v), \forall v \in R^*\}\}$
    if $|U^*| = 1$ **then** $S \leftarrow S \cup U^*$
    else $S \leftarrow S \cup$ random $u \in U^*$
  if $S \notin \mathbf{K}$ **then** $\mathbf{K} \leftarrow \{\mathbf{K}, S\}$
until *a sufficiently large number of times*;
return $\mathbf{K}$

Algorithm 3.1: GREEDY

As discussed later in detail, GREEDY provides a simple way to approximately solve the optimization problem given in Section 3.3.3, allowing integer weights only and no relaxation. The space complexity and running time required to find one kernel are both $O(n + m)$. However, GREEDY does not have a guaranteed error bound, and it ignores the link information between auxiliary and kernel members. As shown in Section 3.5, its performance is not as good as WEBA.

3.3.3 Weight-Balanced Algorithm (WEBA)

Consider an undirected graph $G = (V, E)$ with n vertices and m edges. Intuitively, vertices in community kernels are more influential than those in auxiliary communities. Then, we associate a weight vector $\vec{w}(v) = \{w_1(v), \cdots, w_\ell(v)\}$ with each vertex $v \in V$ to represent its relative importance for each community kernel. In this way, we can determine community kernels by classifying vertex weights. Given a positive integer k (i.e. kernel size), we define the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \mathcal{L}(\vec{w}) = \sum_{(u,v)\in E} \vec{w}(u) \cdot \vec{w}(v) \\ \text{subject to} \quad & \sum_{v\in V} w_i(v) = k, \ \forall i \in \{1, \cdots, \ell\}; \\ & \sum_{1\leqslant i\leqslant \ell} w_i(v) \leqslant 1, \ \forall v \in V; \\ & w_i(v) \geqslant 0, \ \forall v \in V, \ \forall i \in \{1, \cdots, \ell\}. \end{aligned} \tag{3.2}$$

Solving this optimization problem is intractable [40]. Thus, we approximate the solution by iteratively solving its one-dimensional version $\mathcal{L}(w)$. For each detected kernel, we give the following theorem:

Theorem 3.3.1. *A global maximum of the objective function $\mathcal{L}(w)$ corresponds to a community kernel.*

Proof. Assume that a global maximum $\mathcal{L}^*(w)$ is obtained for vertex weights $\{w(u), u \in V\}$. Let $w(u)$ be the probability that the vertex u belongs to a community kernel, and let $nw(u) = \sum_{(u,v)\in E} w(v)$ be the neighboring weight of u. We prove by contradiction that $nw(u) < nw(v)$ if $w(u) < w(v)$ for any pair of vertices u, v.

Assume that there exists a pair of vertices u, v such that $w(u) < w(v)$ and $nw(u) > nw(v)$. Then, define

$$\delta = \min\left\{1 - w(u), w(v), \frac{nw(u) - nw(v)}{2}\right\} > 0.$$

We increase $w(u)$ by δ and decrease $w(v)$ by δ. Then, $\mathcal{L}^*(w)$ is increased by

$$\begin{cases} \delta\left(nw(u) - nw(v)\right) > 0, & \text{for } (u,v) \in E; \\ \delta\left(nw(u) - nw(v)\right) - \delta^2 > 0, & \text{for } (u,v) \notin E. \end{cases}$$

which contradicts the fact that $\mathcal{L}^*(w)$ is a global maximum. Thus, we have $nw(u) < nw(v)$ if $w(u) < w(v)$, which indicates property (1) of Definition 3.2.1. In addition, property (4) is indicated by the unbalanced weakly-bipartite structure. Then, a global maximum of $\mathcal{L}(w)$ corresponds to a community kernel of the graph. □

The problem of maximizing $\mathcal{L}(w)$ is still NP-hard [40], but an approximate solution can be obtained based on pairwise relaxation. Given a kernel size k and an initial subset S obtained by the greedy algorithm, assign weight 1 to each vertex in S and weight 0 to others. Let $N(v)$ be the set of neighboring vertices of v, i.e. $N(v) = \{u \in V | (u,v) \in E\}$, and let $d(v)$ be the degree of v, i.e. $d(v) = |N(v)|$. Then, in each iteration, search for a pair of vertices $u, v \in V$ satisfying the following *relaxation conditions*:

- $w(u) < 1, w(v) > 0$
- $nw(u) > nw(v)$

where $nw(u) = \sum_{v \in N(u)} w(v)$ is the neighboring weight of u. The weights of u and v are modified to locally maximize the objective function $\mathcal{L}(w)$, as shown

in Algorithm 3.2. Repeat this process until no pair of vertices can be found to satisfy the relaxation conditions. Then, all vertices with weight 1 form a community kernel. Further, we can execute this subroutine recursively to obtain multiple community kernels.

Input: $G = (V, E)$ and kernel size k
Output: community kernels $\mathbf{K} = \{\mathcal{K}_1, \mathcal{K}_2, \cdots, \mathcal{K}_\ell\}$

$\mathbf{K} \leftarrow \emptyset$
repeat
  $S \leftarrow \textsc{Greedy}(G, k, 1)$
  $\forall v \in S$, $w(v) \leftarrow 1$; $\forall v \notin S$, $w(v) \leftarrow 0$
  while $\exists\, u, v \in V$ *satisfying the relaxation conditions* **do**
    if $(u, v) \notin E$ **then** $\delta \leftarrow \min\{1 - w(u), w(v)\}$
    else $\delta \leftarrow \min\left\{1 - w(u), w(v), \dfrac{nw(u) - nw(v)}{2}\right\}$
    pick one pair $\{u, v\}$ with the maximum δ value
    $w(u) \leftarrow w(u) + \delta,\ w(v) \leftarrow w(v) - \delta$
  $C \leftarrow \{v \in V \mid w(v) = 1\}$
  if $C \notin \mathbf{K}$ **then** $\mathbf{K} \leftarrow \{\mathbf{K}, C\}$
until *a sufficiently large number of times*;
return $\mathbf{K}$

Algorithm 3.2: WEBA

Theoretical Analysis. Clearly, each vertex should be associated with a valid weight, i.e. $w(v) \in [0, 1], \forall v \in V$, and the sum of all vertex weights should be exactly the kernel size k at the end of each iteration. Moreover, the objective function should increase during each iteration. Now, we prove the correctness of WEBA by induction.

Theorem 3.3.2. *The weight-balanced algorithm is valid and guaranteed to converge.*

Proof. By initialization, $\sum_{v \in V} w(v) = k$ and $0 \leqslant w(v) \leqslant 1$ for each $v \in V$. According to Algorithm 3.2, let $u, v \in V$ be a pair of vertices whose assigned weights $w(u)$ and $w(v)$ are modified to $w'(u)$ and $w'(v)$ in some iteration.

(1) If $(u, v) \notin E$, then $\delta = \min\{1 - w(u), w(v)\} > 0$. Thus,

$$0 \leqslant w(u) < w'(u) = w(u) + \delta \leqslant w(u) + (1 - w(u)) = 1,$$
$$0 = w(v) - w(v) \leqslant w(v) - \delta = w'(v) < w(v) \leqslant 1.$$

The objective function $\mathcal{L}(w)$ is increased by

$$(w'(u) - w(u)) \sum_{p \in N(u)} w(p) + (w'(v) - w(v)) \sum_{p \in N(v)} w(p)$$
$$= \delta \left(nw(u) - nw(v)\right) > 0.$$

(2) If $(u, v) \in E$, then

$$\delta = \min\left\{1 - w(u), w(v), \frac{nw(u) - nw(v)}{2}\right\} > 0.$$

Thus,

$$0 \leqslant w(u) < w'(u) = w(u) + \delta \leqslant w(u) + (1 - w(u)) = 1,$$
$$0 = w(v) - w(v) \leqslant w(v) - \delta = w'(v) < w(v) \leqslant 1.$$

The objective function $\mathcal{L}(w)$ is increased by

$$w'(u)w'(v) - w(u)w(v) + \sum_{\substack{p \in N(u) \\ p \neq v}} \delta w(p) - \sum_{\substack{p \in N(v) \\ p \neq u}} \delta w(p)$$
$$= \delta w(v) - \delta w(u) - \delta^2 + \delta \left(nw(u) - w(v)\right) - \delta \left(nw(v) - w(u)\right)$$
$$= \delta \cdot nw(u) - \delta \cdot nw(v) - \delta^2 = \left(nw(u) - nw(v) - \delta\right) \geqslant \delta^2 > 0.$$

Hence, the validity and correctness of the weight-balanced algorithm is proved.

□

According to the correctness proof, after an infinite number of iterations, each vertex $v \in V$ has an ultimate weight $w^*(v)$, and we have the following theorem:

Theorem 3.3.3. *For any assigned weights* $\{w(v), \forall v \in V\}$ *and any* $\varepsilon > 0$*, after*

$$\max\left\{\frac{4k^3D^5}{\varepsilon^2}, \frac{2mkD^3}{\varepsilon}\right\}$$

iterations, we have $\mathcal{L}(w^*(v)) - \mathcal{L}(w(v)) \leqslant \varepsilon$*, where* k *is the given kernel size and* D *is the highest degree of vertices in the graph* $G = (V, E)$*.*

Proof. In each iteration, among all pairs of vertices that satisfy the relaxation conditions, we choose the one with the maximum δ value and modify their weights. Let

$$\varepsilon' = \min\left\{\frac{\varepsilon}{2kD^2}, \sqrt{\frac{\varepsilon}{2mD^2}}\right\}.$$

Without loss of generality, assume that $\varepsilon' \leqslant nw(u) - nw(v)$ and $\varepsilon' \leqslant \delta$. By the proof of Theorem 3.3.2, the objective function is increased by at least $(\varepsilon')^2$ in each iteration. Since $\mathcal{L}(w) \geqslant 0$ initially and $\mathcal{L}(w^*) \leqslant kD$, the total number of iterations is

$$\leqslant \frac{kD}{(\varepsilon')^2} = \max\left\{\frac{4k^3D^5}{\varepsilon^2}, \frac{2mkD^3}{\varepsilon}\right\}.$$

Assume that the algorithm terminates when $\delta < \varepsilon'$. Since $\varepsilon' D \geqslant |w^*(v) - w(v)|$ for each $v \in V$ upon termination,

$$\begin{aligned}
\mathcal{L}(w^*) - \mathcal{L}(w) &= \sum_{(u,v)\in E} (w^*(u)w^*(v) - w(u)w(v)) \\
&\leqslant \sum_{(u,v)\in E} \left((w(u) + w(v))\,\varepsilon' D + (\varepsilon' D)^2\right) \\
&\leqslant m\,(\varepsilon' D)^2 + \sum_{v\in V} w(v)d(v)\varepsilon' D \\
&\leqslant m\,(\varepsilon' D)^2 + k\varepsilon' D^2 \leqslant \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.
\end{aligned}$$

Thus, after a finite number of iterations, WEBA can obtain a near-optimal solution with a very small error bound. □

3.3.4 Auxiliary Community

After obtaining the community kernels, we use the following approach to find their respective auxiliary communities such that property (2)–(4) of Definition 3.2.1 are satisfied. Initially, label each vertex not in any kernel as unassociated. For each unassociated vertex, rank the kernels according to the number of edges from the vertex to each kernel and the vertices that have already been associated with that kernel. Then, associate the vertex with the top-ranked kernel. If there are ties, leave the vertex unassociated. Repeat this process until no more vertices can be associated with any kernel. Finally, associate each unassociated vertex with every kernel. Then, the auxiliary community of a kernel consists of all the vertices that are associated with that kernel, as shown in Algorithm 3.3.

Input: community kernels $\mathbf{K} = \{\mathcal{K}_1, \mathcal{K}_2, \cdots, \mathcal{K}_\ell\}$
Output: auxiliary communities $\mathbf{A} = \{\mathcal{A}_{\mathcal{K}_1}, \mathcal{A}_{\mathcal{K}_2}, \cdots, \mathcal{A}_{\mathcal{K}_\ell}\}$
$\forall i \in \{1, \cdots, \ell\}$, $\mathcal{A}_{\mathcal{K}_i} \leftarrow \emptyset$
repeat
  $\forall i \in \{1, \cdots, \ell\}$, $R_i = \mathcal{K}_i \cup \mathcal{A}_{\mathcal{K}_i}$
  for $i \leftarrow 1$ **to** ℓ **do**
    $S \leftarrow \{v \notin \cup R_i || E(v, R_i)| > |E(v, R_j)|, \forall j \neq i\}$
    $\mathcal{A}_{\mathcal{K}_i} \leftarrow \mathcal{A}_{\mathcal{K}_i} \cup S$
  end
until *no more vertices can be added*;
$\forall i \in \{1, \cdots, \ell\}$, $\mathcal{A}_{\mathcal{K}_i} \leftarrow \mathcal{A}_{\mathcal{K}_i} \cup \{V - \cup R_i\}$
return A

Algorithm 3.3: Auxiliary Community

3.3.5 Parallelization

To scale up the algorithm to large networks, we develop a parallel implementation. The idea is to distribute the iterative pairwise relaxation (i.e. the outer

loop in Algorithm 3.2) across multiple processors, while keeping the initialization and cleanup phase centralized. Fig. 3.4 shows the speedup of WEBA on the Coauthor, Wikipedia, and Twitter networks for different number of computer nodes (1-6 cores). The speedup curve is close to optimal when the number of cores is relatively small, and it increases steadily with a lower rate than that of the optimal line. It can achieve about 4 times speedup for 6 cores.

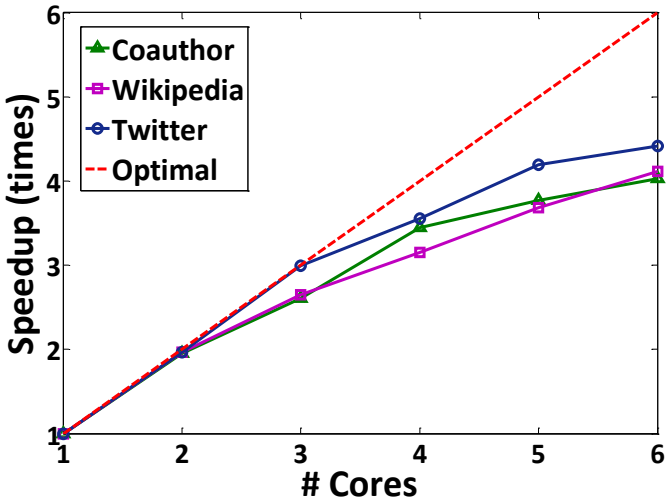


Figure 3.4 Parallelization performance of WEBA.

3.3.6 Other Algorithms

An interesting approach to community detection in social networks was proposed in [11]. Newman proposed the concept of "edge betweenness" and partitioned the graph by removing the edges that are most "between" communities. The betweenness of an edge is defined as the number of shortest paths between pairs of vertices that traverse the edge. If there are more than one shortest path between a pair of vertices, each path is equally weighted such that the total weight of all shortest paths is one. Specifically, the algorithm is as follows:

(1) Compute the betweenness of each edge in the graph.

(2) Remove the edge with the highest betweenness.

(3) Recompute the betweenness of all edges affected by the removal.

(4) Repeat from Step 2 until no edges remain.

In this way, the graph can be partitioned into a number of connected components as the edges are removed, which reveals a hierarchical community structure of the graph. Given a graph $G = (V, E)$ with n vertices and m edges, the time required to compute the betweenness of all edges is $O(nm)$, and the total running time is $O(nm^2)$ in the worst case. However, after each removal, we only need to recompute the betweenness of those edges that are affected by this removal. Thus, the running time is usually better than the worst case. For a graph with about 600k vertices and 4000k edges, the algorithm takes about 18 hours to complete.

Newman also proposed another popular method for community detection based on the concept of "modularity" [4]. Modularity is a quality function Q defined as follows: Let e_{ij} be the fraction of edges that connect between vertices of community i and j, and let $a_i = \sum_j e_{ij}$. Then,

$$Q = \sum_i \left(e_{ii} - a_i^2 \right). \tag{3.3}$$

Starting with an initial state where each vertex is a single community, a greedy optimization algorithm was proposed to find a hierarchical community structure by combining two communities in each iteration that leads to the largest increase (or, smallest decrease) of the modularity Q. After each iteration, Q is changed by $\Delta Q = 2\left(e_{ij} - a_i a_j\right)$, which can be computed in constant time. Then,

each iteration requires $O(n+m)$ time in the worst case and there are at most $n-1$ merge operations. Thus, the overall running time is given by $O((n+m)n)$.

Mishra et al. [10] defined (α, β)-community as a new approach to community detection in social networks that do not require each vertex to belong to exactly one cluster. The objective is to identify (α, β)-communities that are internally dense, i.e. each vertex in the community is adjacent to at least a β-fraction of the cluster, and externally sparse, i.e. any vertex outside of the cluster is adjacent to at most an α-fraction of the vertices in the cluster ($\alpha < \beta$). Further, He et al. [17] proposed a heuristic algorithm based on the concept of (α, β)-community. Given a community size k, it is observed that a large number of (α, β)-communities of size k are neatly categorized into a few massively overlapping clusters. Thus, the algorithm finds a core structure by taking the intersection of each cluster of overlapping (α, β)-communities.

3.4 Data Analysis

In this section, a benchmark Coauthor graph and a Twitter friendship graph are studied. We explore their statistical properties by applying two widely used data analysis techniques: connectivity and degree distribution analysis. Both the Coauthor and Twitter graph exhibit a power-law degree distribution. The Twitter graph also displays a bow-tie structure with a fairly small strongly connected component (SCC).

3.4.1 Coauthor Graph

The Coauthor graph contains the coauthorship of more than 8 thousand academic papers (by more than 50 thousand authors) published at 27 computer science conferences from 2008 to 2010. The subject areas and abbreviations of these conferences are given in Table 3.2.

Table 3.2 Major computer science conferences.

AI	DB	DP	GV	NC
IJCAI	VLDB	PPoPP	SIGGRAPH	SIGCOMM
AAAI	SIGMOD	PACT	CVPR	PERFORMANCE
ICML	PODS	IPDPS	ICCV	SIGMETRICS
UAI	ICDE	ICPP	I3DG	INFOCOM
UM	ICDT	Euro-Par		MOBICOM
NIPS	EDBT			
AAMAS				

There are a total of 52,146 vertices and 134,539 edges in the Coauthor graph. The degree distribution is shown in Fig. 3.5, which exhibits a power law with a consistent exponent about 2.4.

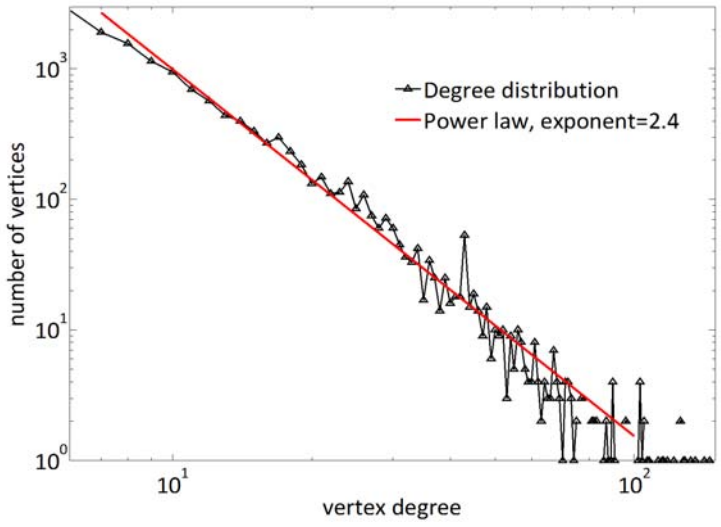


Figure 3.5 Power-law distribution of the Coauthor graph with exponent 2.4.

Computer scientists are usually associated with only one primary subject area, and conferences associated with different research areas usually have different program committee members who are academically active in their respective fields. Thus, the program committee members of the conferences in each research area form a naturalized co-authorship kernel, which represents a common research interest of these computer scientists.

3.4.2 Twitter Graph

The Twitter graph contains the one-way following relationship of more than 40 thousand users registered at `Twitter.com`. There are a total of 465,023 vertices in the Twitter graph, and 38,913 vertices (8%) in its strongly connected component (SCC). It is well-known that many real-world social networks have a bow-tie structure [41]. There is a directed path from each vertex of the set IN to (all the vertices of) SCC. Similarly, there is a directed path from (all the vertices of) SCC to each vertex of the set OUT. Fig. 3.6 shows the bow-tie structure of the Twitter graph, and a detailed description of this structure is given in Table 3.3.

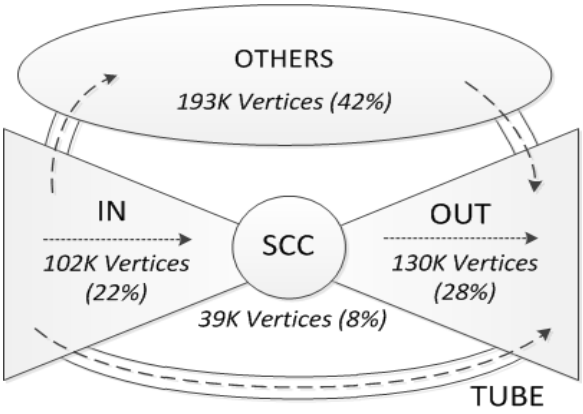
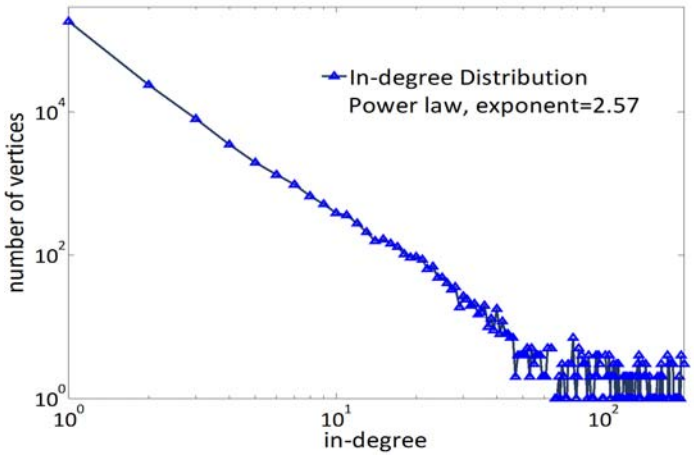


Figure 3.6 Connectivity of the Twitter Graph. There exist paths from any vertex of IN through SCC to any vertex of OUT. There also exist paths from some vertices of IN through a TUBE to some vertices of OUT.

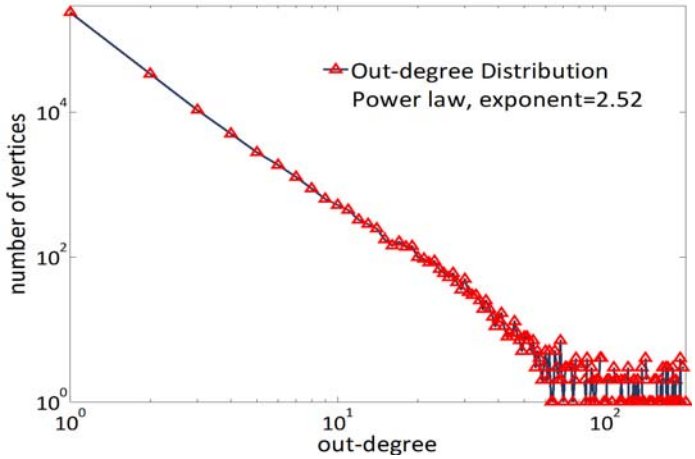
REGION	SCC	IN	OUT	OTHERS	TOTAL
SIZE	38,913	102,379	130,444	193,287	465,023

Table 3.3 The bow-tie structure of the Twitter graph.

Fig. 3.7 shows the in- and out-degree distributions of the Twitter graph. It is interesting to note that both distributions exhibit a power law. Specifically, the exponent of the power law is consistently around 2.57 in the in-degree case and 2.52 in the out-degree case.



(a) In-degree distribution with power-law exponent 2.57.



(b) Out-degree distribution with power-law exponent 2.52.

Figure 3.7 In-degree and out-degree distributions of the Twitter graph.

3.5 Experimental Results

We conduct various experiments in this section to evaluate and analyze the effectiveness and efficiency of our algorithms WEBA and GREEDY. All datasets and codes are publicly available[2].

3.5.1 Experimental Setup

Date Sets. Our experiments are conducted on three different real-world social networks:

- ◇ **Coauthor** (a co-authorship network with 822,415 nodes and 2,928,360 undirected edges). Each vertex represents an author and each edge represents a co-author relation.
- ◇ **Wikipedia** (a co-editorship network with 310,990 nodes and 10,780,996 undirected edges crawled from `wikipedia.org`). Each vertex represents a Wikipedia editor and each edge represents a co-editing relation.
- ◇ **Twitter** (a following network with 465,023 nodes and 833,590 directed edges crawled from `twitter.com`). Each vertex represents a Twitter user account and each edge represents a following relation. It is well-known that the web displays a bow-tie structure [41], where 30% of the vertices are strongly connected. We conduct a bow-tie analysis on Twitter, and discover that only 8% (38,913) of the vertices are strongly connected.

[2] `http://www.cs.cornell.edu/~lwang`

The Twitter dataset is crawled from `twitter.com` starting from Carel Pedre (with username carelpedre), one of Haiti's most popular radio DJs, who used Twitter to inform the world about the earthquake that ravaged Haiti. We extract all followers (>11,704) of "carelpedre" and the users he is following, and continue the process for each extracted Twitter user.

To quantitatively evaluate our algorithms, we construct a benchmark coauthor network and two benchmark wikipedia networks. The benchmark coauthor network contains the co-authorship of more than 8,000 papers published at 27 major computer science conferences from 2008 to 2010. These conferences cover five research areas: Artificial Intelligence (AI), Databases (DB), Distributed and Parallel Computing (DP), Graphics, Vision and HCI (GV), and Networks, Communications and Performance (NC). Computer scientists are usually associated with one primary subject area, and conferences associated with different areas usually have different program committee (PC) members who are academically active in their respective fields. Then, the PC members of the conferences in each research area form a co-authorship kernel, which represents a common research interest of these computer scientists. Our goal is to uncover the five community kernels and their kernel members.

Similarly, the two benchmark wikipedia networks contain the co-editorship of more than 500,000 namespace talk pages and user personal pages modified by both administrators and regular editors. The administrators appointed by Wikipedia are usually knowledgeable in their respective fields, and they are actively maintaining pages with access to restricted technical features. Thus, the administrators form a co-editorship kernel, and our goal is to identify these administrators from others.

Evaluation Measures. To evaluate the performance of WEBA and GREEDY, consider the following aspects:

- ◇ **Quantitative performance.** We use Precision, Recall, and F1-score to evaluate and compare WEBA and GREEDY with other methods. These measures focus on the number of correct pairs of vertices clustered into the same community kernel. For example, for any two PC members in the same field that have coauthored papers together, if they are grouped into the same community kernel, then consider it as a correct pair. We also use pairwise resemblance to measure how similar the ground truth A and a community kernel B detected by an algorithm are. It is defined as $|A \cap B|/|A \cup B|$.
- ◇ **Application case study.** We conduct case study on the Twitter network as the anecdotal evidence to further demonstrate the effectiveness of WEBA.
- ◇ **Efficiency.** We evaluate and compare the efficiency (i.e. elapsed time required for detecting community kernels) of WEBA and GREEDY with alternative algorithms, and analyze the scalability of WEBA.

Comparative Methods. Compare WEBA and GREEDY with the following algorithms for community kernel detection:

- ◇ Local Spectral Partitioning (**LSP**) [15]: community detection algorithm based on conductance. This algorithm is in general a spectral-based graph partitioning method.
- ◇ **d-LSP**: apply LSP to high-degree (top 20%) nodes to find communities. Degree is considered as the relative influence of each vertex.

- ◇ **p-LSP**: apply LSP to high-PageRank (top 20%) nodes [38] to find communities. PageRank is considered as the relative influence of each vertex.
- ◇ **METIS+MQI** [12, 13]: community detection algorithm based on conductance. This algorithm is a flow-based partitioning method for finding low-conductance cuts. This algorithm first partitions the graph into two equal-sized subgraphs, and then finds the cut with the lowest conductance whose smaller side is contained in one of the two subgraphs.
- ◇ **LOUVAIN** [37]: community detection algorithm based on modularity. This algorithm is in general a greedy optimization method.
- ◇ **NEWMAN1** [11]: community detection algorithm based on betweenness. This algorithm is in general an agglomerative hierarchical clustering method.
- ◇ **NEWMAN2** [4]: community detection algorithm based on modularity. This algorithm interprets community detection as a spectral problem in linear algebra.

- $\boldsymbol{\alpha}$-$\boldsymbol{\beta}$ [17]: community detection algorithm based on (α, β)-community.

The first seven algorithms are based on the assumption that communities are densely connected internally and sparsely connected externally, while the last algorithm α-β, similar to WEBA, allows communities to have dense external connections. In addition, d-LSP and p-LSP consider the relative influence of vertices, while the other five do not. All algorithms are implemented using C++ and all experiments are performed on a PC running Windows 7 with an Intel(R) Core(TM) 2 CPU 6600 (2.4GHz and 2.39GHz) and 4GB memory.

3.5.2 Quantitative Performance

We conduct experiments on the benchmark coauthor and wikipedia networks to evaluate and compare GREEDY and WEBA with eight other algorithms. The performance comparison of these algorithms for each metric is given in Table 3.4. Then, we have the following observations:

Performance comparison. WEBA and GREEDY perform much better than the other comparative algorithms for detecting community kernels. On average, WEBA achieves a 14–50% and a 15–42% performance improvement over comparative algorithms in terms of F1-score for the wikipedia and coauthor networks. GREEDY also achieves a better performance than comparative algorithms, but on average works 10% and 7% less well than WEBA.

Fundamental assumption. Similar to WEBA, α-β allows communities to have dense external connections. Thus, it can achieve a better performance than the rest seven algorithms when dealing with unbalanced weakly-bipartite networks. However, it tends to include more vertices in the community kernels, since the relative influence of vertices is not considered here. Thus, α-β has higher recall but lower precision.

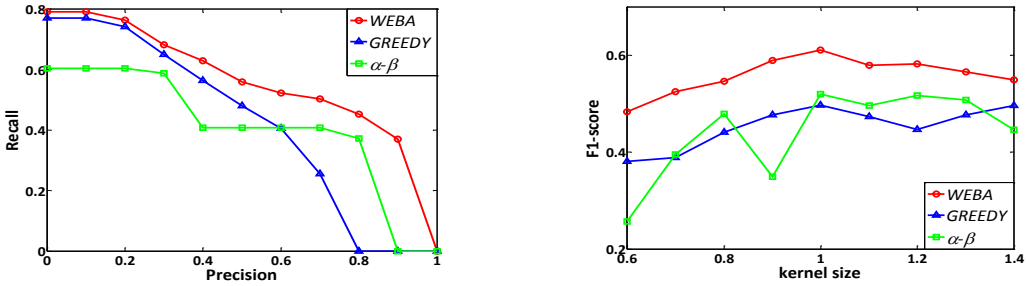
Link information. The relative influence of vertices is not considered in the Local Spectral Partitioning (LSP) algorithm. In d-LSP and p-LSP, we first select the "influential" vertices with respect to degree and PageRank, and then apply LSP for finding community kernels. However, such an algorithm ignores the important link information between auxiliary and kernel vertices. Thus, though both d-LSP and p-LSP achieve some improvement with respect to F1-score and pairwise resemblance, their performance is still not good. Since WEBA consid-

Table 3.4 Algorithm performance comparison on the benchmark coauthor and wikipedia networks. The maximum values for each metric are marked bold.

METRIC	METHOD	WIKIPEDIA			COAUTHOR					
		Talk	**User**	**Average**	**AI**	**DB**	**DP**	**GV**	**NC**	**Average**
Precision	LSP	0.061	0.085	0.073	0.502	0.341	**1.000**	0.682	0.342	0.573
	d-LSP	0.051	0.091	0.071	0.528	0.355	**1.000**	0.697	0.504	0.617
	p-LSP	0.046	0.082	0.064	0.678	0.434	**1.000**	0.692	0.403	0.641
	METIS+MQI	0.049	0.012	0.030	0.847	0.071	0.774	0.692	0.055	0.488
	LOUVAIN	0.063	0.122	0.092	0.216	0.122	**1.000**	0.577	0.272	0.437
	NEWMAN1	0.033	0.203	0.118	0.400	0.027	0.834	0.636	0.259	0.431
	NEWMAN2	0.039	0.085	0.062	0.298	0.320	0.914	0.170	0.613	0.463
	α-β	0.324	0.336	0.330	0.443	**0.868**	0.807	0.267	0.747	0.626
	WEBA	**0.456**	**0.460**	**0.458**	**0.852**	**0.868**	**1.000**	**1.000**	**0.837**	**0.911**
	GREEDY	0.334	0.403	0.368	0.830	0.485	0.844	0.856	0.746	0.752
Recall	LSP	0.171	0.315	0.243	0.458	0.268	0.899	0.783	0.398	0.561
	d-LSP	0.427	0.273	0.350	0.519	0.381	0.899	0.783	0.463	0.609
	p-LSP	0.442	0.237	0.340	0.337	0.428	0.899	0.713	0.491	0.574
	METIS+MQI	0.062	0.361	0.212	0.089	0.047	0.899	0.783	0.077	0.379
	LOUVAIN	0.388	0.348	0.368	0.184	0.148	0.410	0.783	0.190	0.343
	NEWMAN1	0.009	0.077	0.043	0.306	0.075	0.764	0.234	0.174	0.311
	NEWMAN2	0.029	0.075	0.052	0.364	0.386	0.211	0.247	0.467	0.335
	α-β	0.422	0.427	0.424	**0.602**	0.371	**0.908**	**0.822**	0.568	0.654
	WEBA	**0.589**	**0.570**	**0.580**	0.577	0.479	0.899	0.783	**0.582**	**0.664**
	GREEDY	0.432	0.499	0.466	0.545	**0.508**	0.899	0.783	0.560	0.659
F1-score	LSP	0.090	0.134	0.112	0.479	0.300	0.947	0.729	0.368	0.565
	d-LSP	0.091	0.137	0.114	0.524	0.368	0.947	0.737	0.483	0.612
	p-LSP	0.083	0.121	0.102	0.450	0.431	0.947	0.702	0.443	0.595
	METIS+MQI	0.055	0.023	0.039	0.162	0.056	0.832	0.735	0.064	0.370
	LOUVAIN	0.108	0.181	0.144	0.199	0.134	0.582	0.664	0.224	0.361
	NEWMAN1	0.014	0.111	0.062	0.346	0.040	0.797	0.342	0.208	0.347
	NEWMAN2	0.033	0.080	0.056	0.327	0.350	0.343	0.202	0.530	0.350
	α-β	0.367	0.376	0.372	0.510	0.520	0.854	0.403	0.646	0.587
	WEBA	**0.514**	**0.509**	**0.512**	**0.688**	**0.618**	**0.947**	**0.878**	**0.686**	**0.763**
	GREEDY	0.377	0.446	0.412	0.658	0.496	0.870	0.818	0.640	0.696
Resemblance	LSP	0.177	0.175	0.176	0.143	0.143	0.223	0.198	0.138	0.169
	d-LSP	0.175	0.149	0.162	0.164	0.184	0.223	0.189	0.204	0.193
	p-LSP	0.177	0.153	0.165	0.130	0.218	0.223	0.189	0.208	0.194
	METIS+MQI	0.130	0.090	0.110	0.022	0.028	0.104	0.068	0.018	0.048
	LOUVAIN	0.212	0.245	0.228	0.101	0.109	0.117	0.159	0.102	0.118
	NEWMAN1	0.127	0.208	0.168	0.139	0.040	0.193	0.110	0.119	0.120
	NEWMAN2	0.131	0.148	0.140	0.137	0.154	0.088	0.071	0.198	0.130
	α-β	0.436	0.444	0.440	0.178	0.219	0.213	0.180	0.227	0.203
	WEBA	**0.561**	**0.557**	**0.559**	**0.234**	**0.274**	**0.236**	**0.229**	**0.259**	**0.246**
	GREEDY	0.445	0.503	0.474	0.216	0.237	0.216	0.207	0.234	0.222

ers the link information between auxiliary and kernel members, it achieves a much better performance.

Sensitivity analysis. Fig. 3.8(a) shows the Recall of WEBA, GREEDY, and α-β as a function of Precision. Fig. 3.8(b) shows the F1-score of WEBA, GREEDY, and α-β as a function of the kernel size. WEBA has the highest Recall for the same Precision and the highest F1-score for the same kernel size. α-β is more sensitive to the kernel size change, though in some cases, it achieves a better F1-score than GREEDY for the same kernel size.



(a) Precision vs. Recall (b) F1-score vs. kernel size

Figure 3.8 Sensitivity analysis on the benchmark coauthor network.

3.5.3 Application Case Study

A typical application of our problem is to identify influential users. We present an example on the Twitter network, as shown in Fig. 3.9. A clear difference can be observed in the results obtained by WEBA, METIS+MQI, and NEWMAN2. The left figure shows four community kernels obtained by WEBA. The yellow nodes represent the auxiliary members surrounding the four kernels. Some kernel members are enlarged to highlight the details of the community kernels. Interestingly, the blue kernel consists of a group of well-known entertainers and the red kernel consists of a group of active politicians, which verifies the defini-

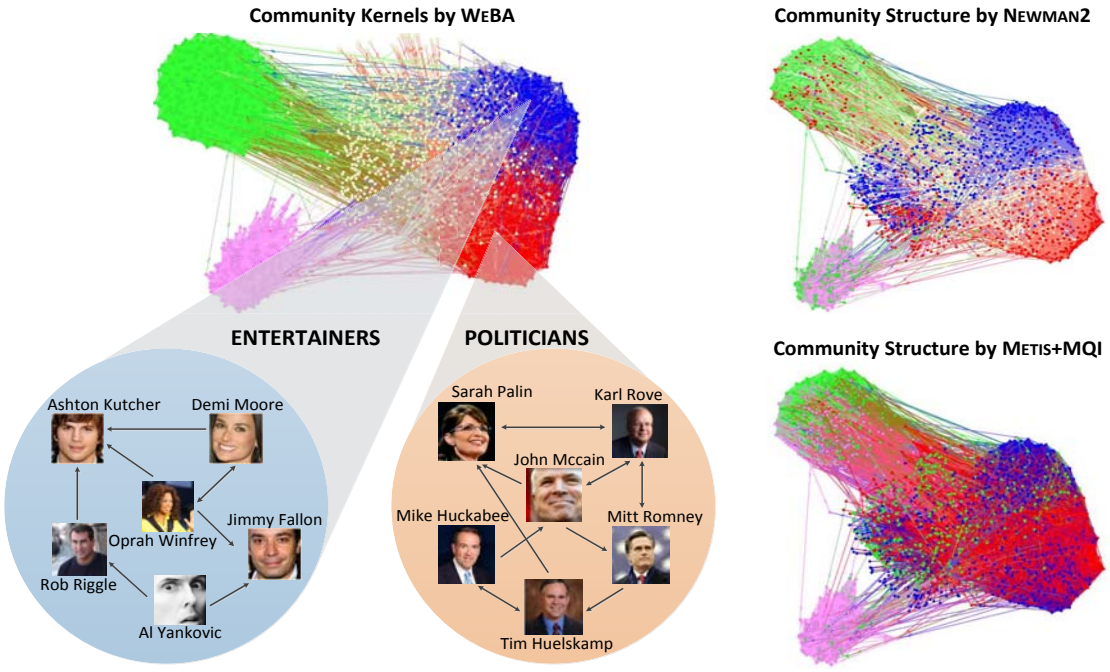


Figure 3.9 Case study on the Twitter network. WEBA discovers four meaningful community kernels from their numerous followers (colored yellow). The blue kernel consists of entertainers and the red kernel consists of politicians.

tion of community kernel. The upper and lower right figures show four communities obtained by METIS+MQI and NEWMAN2. By contrast, most of the yellow nodes are grouped into one of the four communities here, and the communities are blended with each other. The case study results further demonstrate the better performance of WEBA for finding meaningful communities.

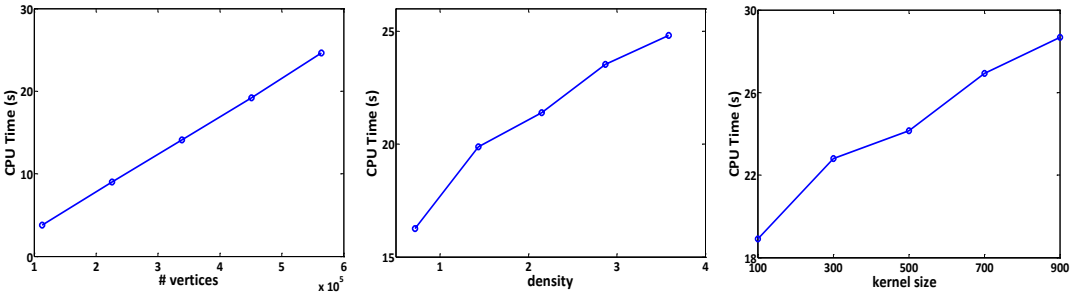
3.5.4 Efficiency and Scalability

We now evaluate the efficiency performance of GREEDY and WEBA by comparing their computational time required to detect community kernels with that of other algorithms on the Coauthor, Wikipedia, and Twitter networks. We also evaluate the scalability performance of WEBA with respect to three main parameters: the number of vertices, the density, and the kernel size.

Table 3.5 Efficiency comparison on Twitter, Coauthor, and Wikipedia.

Datasets	NEWMAN1	NEWMAN2	LSP	METIS+MQI	LOUVAIN	WEBA	GREEDY	α-β
Twitter	52 hr	12.4 min	8.0 min	12.3 min	5.0 min	24.7 s	9.7 s	64.4 min
Coauthor	$>$ 100 hr	142.7 min	35.3 min	32.9 min	10.1 min	2.0 min	1.4 min	186.7 min
Wikipedia	$>$ 100 hr	83.9 min	15.7 min	52.3 min	2.2 min	50.6 s	22.4 s	236.2 min

The CPU time required by each algorithm for detecting community kernels in the Coauthor, Wikipedia, and Twitter networks is given in Fig. 3.2(a)-3.2(c) and Table 3.5. Clearly, both WEBA and GREEDY significantly reduce the required CPU time compared with the other algorithms. Further, we analyze the scalability of WEBA to understand how it can be affected by the network structure and the input parameter (i.e. kernel size). We generate a synthetic dataset on which a series of experiments are conducted by varying the number of vertices, the density $|E|/|V|$, and the kernel size k. The analysis results are shown in Fig. 3.10(a)-3.10(c). Clearly, the CPU time required by WEBA increases (almost) linearly with respect to the number of vertices, the density, and the kernel size, which demonstrates the high scalability of WEBA.



(a) CPU time vs. # vertices (b) CPU time vs. density (c) CPU time vs. kernel size

Figure 3.10 Scalability performance of WEBA with respect to the number of vertices, density, and kernel size (no parallelization).

3.6 Conclusion

A structure of community kernels and their auxiliary communities can be found in many real-world social networks that are unbalanced weakly-bipartite. Community kernels are particularly useful to distinguish different groups of social entities and to capture the common property shared by each group. We formally define the problem of detecting community kernels in large social networks. We propose a greedy algorithm and an efficient weight-balanced algorithm WEBA with guaranteed error bound for finding community kernels. The experimental results on the benchmark coauthor and wikipedia networks show that WEBA significantly improves the performance over traditional cut-based and conductance-based algorithms, since the relative influence of vertices and the link information between auxiliary and kernel members are both considered. The qualitative case study on the Twitter network further demonstrates the ability of WEBA to find meaningful community kernels, which reveal the common profession, interest, or popularity of groups of influential individuals.

For future work, we would like to explore the dynamic behavior of community kernels and their auxiliary communities. We are interested in how community kernels take shape and evolve over time. In addition, we would like to combine link and content information in our problem definition and algorithm design for broader practical applications, such as query-dependent community detection.

CHAPTER 4

HOW TO INFER NETWORKS OF DIFFUSION?

4.1 Introduction

Cascading processes, such as the spread of a computer virus or an infectious disease, are a pervasive phenomenon in many networks. Diffusion and propagation processes have been studied in a broad range of disciplines, such as information diffusion [2, 19, 20, 42], social networks [43, 44], viral marketing [45, 46], epidemiology [47], and ecology [48]. In previous work, researchers have mostly focused on a number of optimization problems derived from cascading processes, where the goal is to devise intervention strategies to either maximize (e.g., viral marketing) or minimize (e.g., network interdiction, vaccination programs) the propagation. However, these studies often assume that the underlying network is known to the observer, which in practice is not true in many situations.

In this chapter, we revisit the problem of inferring latent network structure given observations of a diffusion process. For example, by observing a disease epidemic, we want to infer the underlying social contact network, or by observing the spread of trending topics, we want to estimate the connectivity of the social media. Fig. 4.1 illustrates a case of information diffusion in the popular Twitter network. The nodes represent a subset of the Twitter users that have posted about a common trending topic, and the directed edges represent the "following" relation between the users. There is a clear pattern in the figure. The bigger and darker nodes, followed by the smaller and lighter nodes, form the hubs of the diffusion process. By looking at the time-stamps of the

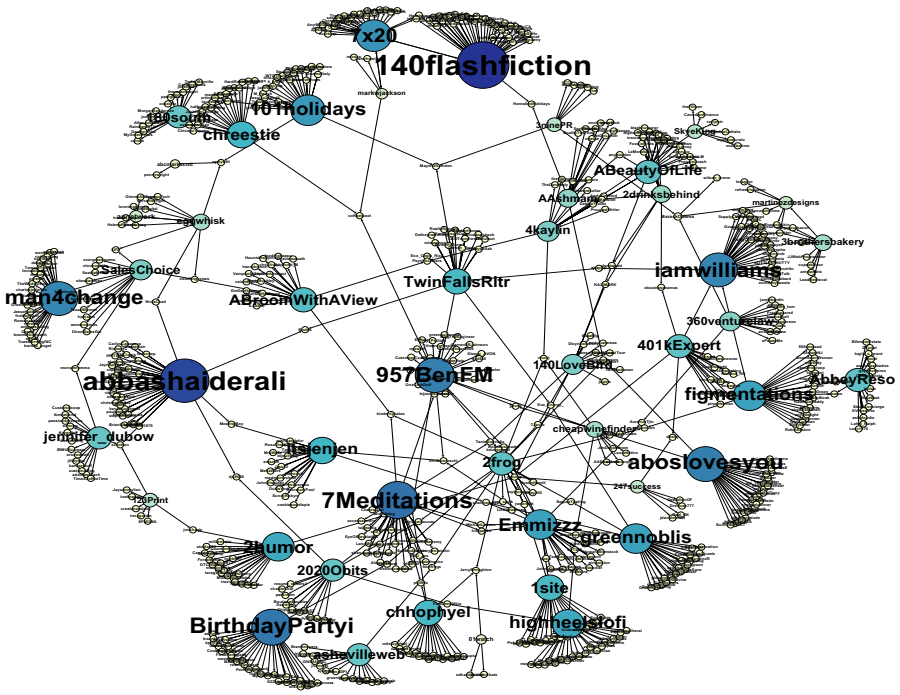


Figure 4.1 Information diffusion in the Twitter network.

messages and at the underlying network structure, we observe that most information flows initiate at a hub node and spread across the network to reach other hub nodes and their followers. However, it is non-trivial to come up with such a picture simply by looking at the time-stamps of the messages, since without knowing the underlying network structure, we cannot decide from whom a node copied the information from. Intuitively, messages carry implicit information about the social relations among users. For instance, users who repeatedly post messages about the same topic within a short period of time, are more likely to be connected. Thus, a motivating application of this chapter is to what extent we can estimate the relations in social networks by analyzing the messages published by users over time.

This type of latent network inference problem based on the time-stamps of infection (or, information-reproduction) events has received increasing interest

over the past few years [2, 19, 20]. Previous work was largely based on two major assumptions: (1) the diffusion process is causal (i.e., not affected by events in the future), and (2) infection events closer in time are more likely to be causally related (e.g., according to an exponential, Rayleigh, or power-law distribution). While the causality assumption is indeed crucial and always satisfied in practice, we realize that there are many other factors that can be highly informative as far as the causality relations are concerned. For example, the time-stamps at which two users publish their tweets are important to decide whether they are related, but other factors such as the language or the content of the messages can be as important. Even if the two messages are close in time, they are unlikely to be related if the messages are written in different languages. Further, previous models in the literature are mostly focused on monotonic processes, while real-world processes are often recurrent. For instance, it is very common for one user to post about the same topic multiple times on Twitter, or purchase the same item regularly on Amazon.

Contributions. Motivated by these challenges, we define a family of novel probabilistic models that generalize previous models based solely on time. We propose a primary approach MONET that can handle recurrent diffusion processes. Further, we consider a richer set of additional features for infection events, defining novel feature-enhanced models that can better explain the observed data. With distributed optimization and convex objective functions, we can efficiently solve the problem of inferring the most probable latent network structure. Using additional features such as the languages of the messages and Jaccard indexes between the messages, we can accurately recover the links of the Twitter network by analyzing the topics of a large number of messages posted by a subset of the Twitter users over a 10-month period. Experimental results

show that our models significantly improve the accuracy of the estimates over previous models by as much as 78.7%.

The rest of this chapter is organized as follows. In Section 4.2, we propose a family of feature-enhanced probabilistic models and formally define the network inference problem. In Section 4.3, we propose a set of solutions based on the primary approach MONET for the network inference problem. In Section 4.4, we present experimental results to validate the effectiveness and computational efficiency of our methods. Finally, we conclude in Section 4.5 with comments on the problems considered and future work.

4.2 Problem Definition

We consider a diffusion process across a network represented by a directed, weighted graph $G = (V, E)$. Let $\mathbf{A} = \{\alpha_{jk} | j, k \in V, j \neq k\}$ be the adjacency matrix of weights. A directed edge (j, k) has weight $\alpha_{jk} \geqslant 0$ that denotes the pairwise transmission rate from node j to node k. For example, in the case of an infectious disease spreading through a population, V represents a group of individuals and E represents the strength of the social contacts among them. In the case of an invasive species colonizing a new territory, V represents patches of land and E represents the connectivity between them. We assume that the diffusion process is stochastic but *causal*, that is, it depends on the past history but not on the future. Specifically, we consider a diffusion process that starts with one or more nodes, and spreads across the network subject to an independent local probabilistic model of "infection", where a node infects its neighbors independently of the status of other nodes in the network [43].

When studying such diffusion processes, the underlying network is often unknown (latent). However, we assume that one can observe a set of *cascades* of "infection" (or, information-reproduction) events. A ***cascade*** is a sequence of infection events

$$\pi = \{(v_0, t_0), \cdots, (v_N, t_N)\}$$

during a given time interval T, where $v_i \in V$ is a node that becomes infected at time t_i. T is the *horizon* of cascade π. Note that different cascades may have different horizons. For example, in the Twitter network, each cascade corresponds to a trending topic, and we have an entry (v_i, t_i) for each tweet posted by user v_i at time t_i. Given a probabilistic model $P(\pi|G)$ that gives the probability of observing a certain cascade π when the underlying network is G, the problem of inferring the latent network structure from observed cascades has received considerable attention [2, 19, 20]. It is usually assumed that V is known but E is unknown, and the goal is to find a (weighted) graph

$$G^* = \arg\max_G \prod_{c=1}^{M} P(\pi^c|G)$$

that maximizes the probability of observing cascades $\pi^1, \cdots, \pi^M$, which are assumed to be i.i.d. (independent, identically distributed) realizations of the underlying diffusion process.

The building block to define $P(\cdot)$ is the likelihood function $f(t_k|t_j; \alpha_{jk})$ that gives the probability density that node v_j infected at time t_j infects node v_k at time t_k (see below for more details). Such models are centered on the time differences between the infection events of a cascade, and exploit the causal nature of the diffusion process by setting the likelihood to zero whenever $t_k - t_j < 0$. Further, they assume that events closer in time are more likely to be causally related. For instance, if node v_k became infected shortly after node v_j was in-

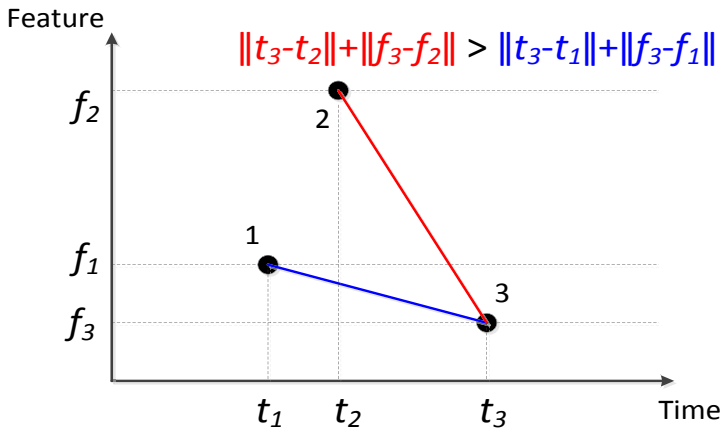


Figure 4.2 Feature-enhanced probabilistic model.

fected, this is considered as an indication that the two events are causally related (i.e., v_k was infected by v_j). These time-based models are the foundation of our work.

While time is indeed a crucial element of the network inference problem, in practical applications, observations of a diffusion process often carry additional key information. For example, the diagnosis of an infection often comes with additional information about the specific strain. When a topic or a rumor spreads through a social network, one can also observe the context in which it appears. This motivates our definition of a ***generalized cascade***

$$\pi_g = \{(v_0, t_0, \mathbf{f}_0), \cdots, (v_N, t_N, \mathbf{f}_N)\} \tag{4.1}$$

where $v_i \in V$ is a node infected at time t_i, and $\mathbf{f}_i \in \mathcal{F}$ is a feature vector describing the additional information available for the i-th infection event. Using the additional information contained in a generalized cascade, we can define a generalized feature-enhanced probabilistic model where the probability of a transmission event depends not only on the time differences, but also on the additional features. Specifically, we use a probability density function $f(t_k, \mathbf{f}_k | t_j, \mathbf{f}_j; \alpha_{jk})$ as a building block, which depends causally on the relative

time difference $t_k - t_j$ as well as on the additional features $\mathbf{f}_k$ and $\mathbf{f}_j$. Fig. 4.2 gives an example of the difference between previous models that are based solely on time and a case of our feature-enhanced models where $f(t_k, \mathbf{f}_k | t_j, \mathbf{f}_j; \alpha_{jk})$ depends on $||t_k - t_j|| + ||\mathbf{f}_k - \mathbf{f}_j||$. Node 3 is considered to be more related to node 1 than node 2 by our feature-enhanced models, while it is determined to be more related to node 2 by models based only on time.

Furthermore, previous models are focused on monotonic diffusion processes, while most real-world processes are recurrent. For example, it is common for one user to post about the same topic multiple times on Twitter, or purchase the same item multiple times on Amazon. Repeated posts of the same topic show a higher level of interest in that topic, and exchanged posts of the same topic between a group of nodes also show a higher level of connectivity in that group. We take these factors into account in our feature-enhanced models and assign respective reward / penalty to each scenario.

There are two different ways of modeling a diffusion process where nodes can be infected multiple times in one cascade. The first model considers an infection event as the result of all previous events, and thus we call it *non-splitting*. By contrast, the second model considers an infection event of a node as the result of all previous events *up to its last infection*. This model is memoryless and thus we call it *splitting*. We mainly focus on the non-splitting model in this section, but the results can be extended to the splitting case. We will later present experimental results in Section 4.4 for both non-splitting and splitting models.

4.2.1 Generalized Cascade Model

We first recall some standard notation from previous literature, and then define our feature-enhanced models based on generalized cascades.

Recap. Recall the standard notation from [2] and [49]. Given that node j was infected at time t_j, the *survival function* of edge (j,k) is the probability that, by time t_k, node k was not infected by node j. That is,

$$S\left(t_k|t_j;\alpha_{jk}\right) = 1 - F\left(t_k|t_j;\alpha_{jk}\right), \tag{4.2}$$

where α_{jk} denotes the transmission rate from node j to node k, and $F\left(t_k|t_j;\alpha_{jk}\right)$ is the cumulative distribution function. Further, the *hazard function* (or, instantaneous infection rate) of edge (j,k) is given by

$$H\left(t_k|t_j;\alpha_{jk}\right) = \frac{f\left(t_k|t_j;\alpha_{jk}\right)}{S\left(t_k|t_j;\alpha_{jk}\right)}, \tag{4.3}$$

where

$$f\left(t_k|t_j;\alpha_{jk}\right) = \left.\frac{d}{dt}F\left(t|t_j;\alpha_{jk}\right)\right|_{t_k}$$

is the likelihood function. Table 4.1 shows the survival and hazard functions based on the exponential, Rayleigh, and power-law distribution.

Table 4.1 Parametric Models [2].

Model	Likelihood Function $f\left(t_k\|t_j;\alpha_{jk}\right)$	Survival Function $S\left(t_k\|t_j;\alpha_{jk}\right)$	Hazard Function $H\left(t_k\|t_j;\alpha_{jk}\right)$
Exponential	$\begin{cases} \alpha_{jk} e^{-\alpha_{jk}(t_k - t_j)} \\ 0 \end{cases}$	if $t_j < t_k$ $e^{-\alpha_{jk}(t_k - t_j)}$ otherwise	α_{jk}
Rayleigh	$\begin{cases} \alpha_{jk}\left(t_k - t_j\right) e^{-\alpha_{jk}\frac{\left(t_k - t_j\right)^2}{2}} \\ 0 \end{cases}$	if $t_j < t_k$ $e^{-\alpha_{jk}\frac{\left(t_k - t_j\right)^2}{2}}$ otherwise	$\alpha_{jk}\left(t_k - t_j\right)$
Power Law	$\begin{cases} \frac{\alpha_{jk}}{\delta}\left(\frac{t_k - t_j}{\delta}\right)^{-1-\alpha_{jk}} \\ 0 \end{cases}$	if $t_j < t_k$ $\left(\frac{\delta t_k - t_j}{\delta}\right)^{-\alpha_{jk}}$ otherwise	$\frac{\alpha_{jk}}{t_k - t_j}$

Multiple Occurrences. Real-world diffusion processes are often recurrent, that is, we often observe multiple occurrences of the same node in one cascade. With

multiple occurrences of node k and node j, the survival function for the non-splitting case is

$$\begin{aligned} S\left(t_k|t_j;\alpha_{jk}\right) &= \prod_{k:t_k^{(1)}\leqslant T^c}\prod_{1\leqslant i\leqslant N_k^c}\prod_{j\neq k:t_j^{(1)}<t_k^{(i)}} S\left(t_k^{(i)}|t_j;\alpha_{jk}\right) \\ &= \prod_{k:t_k^{(1)}\leqslant T^c}\prod_{1\leqslant i\leqslant N_k^c}\prod_{j\neq k:t_j^{(1)}<t_k^{(i)}}\prod_{1\leqslant \ell\leqslant N_j^c(t_k^{(i)})} S\left(t_k^{(i)}|t_j^{(\ell)};\alpha_{jk}\right), \end{aligned}$$

where T^c is the horizon of cascade π^c, and $t_k^{(i)}, i \in \{0,\cdots,N_k^c, N_k^c+1\}$ denote the time-stamps of node k infections in cascade π^c. We assign two special time-stamps for every node: $t_k^{(0)} = 0$ and $t_k^{\left(N_k^c+1\right)} = T^c$. N_k^c denotes the number of node k infections in cascade π^c. $N_j^c(t_k^{(i)})$ denotes the number of node j infections before the i-th infection of node k. Similarly, the hazard function is given by

$$H\left(t_k^{(i)}|t_j^{(\ell)};\alpha_{j,k}\right) = \frac{f\left(t_k^{(i)}|t_j^{(\ell)};\alpha_{j,k}\right)}{S\left(t_k^{(i)}|t_j^{(\ell)};\alpha_{j,k}\right)}.$$

Additional Features. Consider two feature vectors $\mathbf{f}_k, \mathbf{f}_j \in \mathcal{F}$ associated with node $k, j \in V$ in a cascade. Let $d\left(\mathbf{f}_k, \mathbf{f}_j\right)$ denote the distance between the two feature vectors. We include an extra term $e^{-d(\mathbf{f}_k,\mathbf{f}_j)}$ in the likelihood function to reflect this distance factor. For example, given an exponential distribution, we have

$$f\left(t_k, \mathbf{f}_k|t_j, \mathbf{f}_j;\alpha_{jk}\right) = \begin{cases} \gamma\alpha_{jk}e^{-d(\mathbf{f}_k,\mathbf{f}_j)}e^{-\alpha_{jk}(t_k-t_j)}, & \text{if } t_j < t_k; \\ 0, & \text{otherwise.} \end{cases}$$

where γ is a normalization constant. Thus, the survival function is given by

$$\begin{aligned} S\left(t_k|t_j,\mathbf{f}_j;\alpha_{jk}\right) &= 1 - F\left(t_k|t_j,\mathbf{f}_j;\alpha_{jk}\right) = 1 - \int_{\mathcal{F}}\int_{t_j}^{t_k} f\left(t,\mathbf{f}|t_j,\mathbf{f}_j;\alpha_{jk}\right)\,\mathrm{d}t\,\mathrm{d}\mathbf{f} \\ &= e^{-\alpha_{jk}(t_k-t_j)}\int_{\mathcal{F}}\gamma e^{-d(\mathbf{f},\mathbf{f}_j)}\mathrm{d}\mathbf{f} = e^{-\alpha_{jk}(t_k-t_j)}. \end{aligned} \tag{4.4}$$

Then, the hazard function is

$$H\left(t_k, \mathbf{f}_k | t_j, \mathbf{f}_j; \alpha_{jk}\right) = \frac{f\left(t_k, \mathbf{f}_k | t_j, \mathbf{f}_j; \alpha_{jk}\right)}{S\left(t_k | t_j, \mathbf{f}_j; \alpha_{jk}\right)} = \begin{cases} \gamma \alpha_{jk} e^{-d(\mathbf{f}_k, \mathbf{f}_j)}, & \text{if } t_j < t_k; \\ 0, & \text{otherwise.} \end{cases} \tag{4.5}$$

4.2.2 MAP Inference

Given independent cascades, the likelihood of a set of cascades $\{\pi_g^1, \cdots, \pi_g^M\}$ is the product of the likelihood of each cascade:

$$\prod_{1 \leqslant c \leqslant M} f\left(\pi_g^c; \mathbf{A}\right), \tag{4.6}$$

where $\mathbf{A} = \{\alpha_{jk} | j, k \in V, j \neq k\}$ is a weighted adjacency matrix of transmission rates. Given a cascade π_g^c, the probability that node k was not infected by time T^c is the product of the survival functions of the infected nodes. The formulation can be extended according to the generalized cascade model discussed in Section 4.2.1. For example, if a node was infected multiple times during the observation window, this repeated lack of ability to infect node k should also be considered. Specifically, the probability that node k was not infected by time T^c is

$$\prod_{j: t_j^{(1)} \leqslant T^c} \prod_{1 \leqslant i \leqslant N_j^c} S\left(T^c | t_j^{(i)}, \mathbf{f}_j^{(i)}; \alpha_{jk}\right).$$

Given the parents of the infected nodes, infections are assumed to be conditionally independent. Thus, the likelihood of the observed cascade π_g^c is

$$f\left(\pi_g^c; \mathbf{A}\right) = \prod_{j: t_j^{(1)} \leqslant T^c} \prod_{1 \leqslant i \leqslant N_j^c} f\left(t_j^{(i)}, \mathbf{f}_j^{(i)} | \pi_g^c \backslash \left(j, t_j^{(i)}, \mathbf{f}_j^{(i)}\right); \mathbf{A}\right).$$

Given the i-th infection of node k, the likelihood of node j being its first parent is

$$f\left(t_k^{(i)}, \mathbf{f}_k^{(i)} | t_j, \mathbf{f}_j; \mathbf{A}\right) = \prod_{s \neq j : t_s^{(1)} < t_k^{(i)}} \prod_{1 \leqslant p \leqslant N_s^c(t_k^{(i)})} S\left(t_k^{(i)} | t_s^{(p)}, \mathbf{f}_s^{(p)}; \alpha_{s,k}\right) \times \sum_{1 \leqslant \ell \leqslant N_j^c(t_k^{(i)})} f\left(t_k^{(i)}, \mathbf{f}_k^{(i)} | t_j^{(\ell)}, \mathbf{f}_j^{(\ell)}; \alpha_{j,k}\right) \prod_{q \neq \ell} S\left(t_k^{(i)} | t_j^{(q)}, \mathbf{f}_j^{(q)}; \alpha_{j,k}\right).$$

Thus, the likelihood of the observed cascade π_g^c is

$$f\left(\pi_g^c; \mathbf{A}\right) = \prod_{k : t_k^{(1)} \leqslant T^c} \prod_{1 \leqslant i \leqslant N_k^c} \left(\sum_{j : t_j^{(1)} < t_k^{(i)}} f\left(t_k^{(i)}, \mathbf{f}_k^{(i)} | t_j, \mathbf{f}_j; \mathbf{A}\right) \right).$$

Combine the two equations above and include the condition $s = j$, we have

$$\begin{aligned} f\left(\pi_g^c; \mathbf{A}\right) &= \prod_{k : t_k^{(1)} \leqslant T^c} \prod_{1 \leqslant i \leqslant N_k^c} \left(\prod_{s : t_s^{(1)} < t_k^{(i)}} S\left(t_k^{(i)} | t_s, \mathbf{f}_s; \alpha_{s,k}\right) \times \right. \\ &\qquad \left. \sum_{j : t_j^{(1)} < t_k^{(i)}} \sum_{1 \leqslant \ell \leqslant N_j^c(t_k^{(i)})} \frac{f\left(t_k^{(i)}, \mathbf{f}_k^{(i)} | t_j^{(\ell)}, \mathbf{f}_j^{(\ell)}; \alpha_{j,k}\right)}{S\left(t_k^{(i)} | t_j^{(\ell)}, \mathbf{f}_j^{(\ell)}; \alpha_{j,k}\right)} \right) \\ &= \prod_{k : t_k^{(1)} \leqslant T^c} \prod_{1 \leqslant i \leqslant N_k^c} \left(\prod_{s : t_s^{(1)} < t_k^{(i)}} S\left(t_k^{(i)} | t_s, \mathbf{f}_s; \alpha_{s,k}\right) \times \right. \\ &\qquad \left. \sum_{j : t_j^{(1)} < t_k^{(i)}} \sum_{1 \leqslant \ell \leqslant N_j^c(t_k^{(i)})} H\left(t_k^{(i)}, \mathbf{f}_k^{(i)} | t_j^{(\ell)}, \mathbf{f}_j^{(\ell)}; \alpha_{j,k}\right) \right). \end{aligned}$$

Add the information that some nodes were never infected during the horizon T^c, and then,

$$\begin{aligned} f\left(\pi_g^c; \mathbf{A}\right) &= \prod_{k : t_k^{(1)} \leqslant T^c} \prod_{1 \leqslant i \leqslant N_k^c} \prod_{t_m > T^c} S\left(T^c | t_k^{(i)}, \mathbf{f}_k^{(i)}; \alpha_{i,m}\right) \times \\ &\prod_{s : t_s^{(1)} < t_k^{(i)}} S\left(t_k^{(i)} | t_s, \mathbf{f}_s; \alpha_{s,k}\right) \times \sum_{j : t_j^{(1)} < t_k^{(i)}} \sum_{1 \leqslant \ell \leqslant N_j^c(t_k^{(i)})} H\left(t_k^{(i)}, \mathbf{f}_k^{(i)} | t_j^{(\ell)}, \mathbf{f}_j^{(\ell)}; \alpha_{j,k}\right). \end{aligned} \quad (4.7)$$

Eq. (4.7) gives the likelihood of cascade π_g^c for the non-splitting model. However, for the splitting case, this likelihood is given by

$$
\begin{aligned}
f\left(\pi_g^c; \mathbf{A}\right) = & \prod_{k:t_k^{(1)} \leqslant T^c} \prod_{1 \leqslant i \leqslant N_k^c} \prod_{t_m > T^c} S\left(T^c | t_k^{(i)}, \mathbf{f}_k^{(i)}; \alpha_{i,m}\right) \times \\
& \prod_{s:t_s^{(1)} < t_k^{(i)}} \prod_{N_s^c(t_k^{(i-1)}) < p \leqslant N_s^c(t_k^{(i)})} S\left(t_k^{(i)} | t_s^{(p)}, \mathbf{f}_s^{(p)}; \alpha_{s,k}\right) \times \\
& \sum_{j:t_j^{(1)} < t_k^{(i)}} \sum_{N_j^c(t_k^{(i-1)}) \leqslant \ell \leqslant N_j^c(t_k^{(i)})} H\left(t_k^{(i)}, \mathbf{f}_k^{(i)} | t_j^{(\ell)}, \mathbf{f}_j^{(\ell)}; \alpha_{j,k}\right). \qquad (4.8)
\end{aligned}
$$

Eq. (4.8) is similar to Eq. (4.7) except that we only consider the segment between the $(i-1)$-th and i-th occurrence of node k for the survival and hazard function.

Problem Definition. Our goal is to infer the connectivity and estimate the infection rate α_{jk} for each pair of nodes (j, k) such that the likelihood of observed cascades $\{\pi_g^1, \cdots, \pi_g^M\}$ is maximized. Specifically,

$$
\begin{aligned}
\text{minimize}_{\mathbf{A}} \quad & -\sum_{1 \leqslant c \leqslant M} \log f\left(\pi_g^c; \mathbf{A}\right) \\
\text{subject to} \quad & \alpha_{jk} \geqslant 0,\ j, k \in V, j \neq k.
\end{aligned} \qquad (4.9)
$$

where $\mathbf{A} = \{\alpha_{jk} | j, k \in V, j \neq k\}$ are the variables. The inferred edges of the network are those pairs of nodes with infection rate $\alpha_{jk} > 0$.

4.3 Proposed Approach: MONET

In this section, we discuss the properties of the optimization problem arising from the MAP inference task in our feature-enhanced probabilistic models defined in Section 4.2. By Eq. (4.6) and Eq. (4.7), the log-likelihood of cascades

$\{\pi_g^1, \cdots, \pi_g^M\}$ is

$$L\left(\{\pi_g^1, \cdots, \pi_g^M\}; \mathbf{A}\right) = \sum_{1 \leqslant c \leqslant M} \Phi_1(\pi_g^c; \mathbf{A}) + \Phi_2(\pi_g^c; \mathbf{A}) + \Phi_3(\pi_g^c; \mathbf{A}), \tag{4.10}$$

where for each cascade $\pi_g^c \in \{\pi_g^1, \cdots, \pi_g^M\}$,

$$\Phi_1(\pi_g^c; A) = \sum_{k:t_k^{(1)} \leqslant T^c} \sum_{1 \leqslant i \leqslant N_k^c} \sum_{s:t_s^{(1)} < t_k^{(i)}} \log S\left(t_k^{(i)} | t_s, \mathbf{f}_s; \alpha_{s,k}\right),$$

$$\Phi_2(\pi_g^c; A) = \sum_{k:t_k^{(1)} \leqslant T^c} \sum_{1 \leqslant i \leqslant N_k^c} \log \sum_{j:t_j^{(1)} < t_k^{(i)}} \sum_{1 \leqslant \ell \leqslant N_j^c(t_k^{(i)})} H\left(t_k^{(i)}, \mathbf{f}_k^{(i)} | t_j^{(\ell)}, \mathbf{f}_j^{(\ell)}; \alpha_{j,k}\right),$$

$$\Phi_3(\pi_g^c; A) = \sum_{k:t_k^{(1)} \leqslant T^c} \sum_{1 \leqslant i \leqslant N_k^c} \sum_{t_m^{(1)} > T^c} \log S\left(T^c | t_k^{(i)}, \mathbf{f}_k^{(i)}; \alpha_{k,m}\right).$$

The above equations are a strict generalization of the ones presented in [2], that is, we recover the same formulation where there are no multiple occurrences and we do not consider any additional features other than time. Further, we also generalize several results of the model in [2] to our feature-enriched setting, for both splitting and non-splitting cases. Formally,

Theorem 4.3.1. *The following results hold:*

(1) Given any distance functions, log-concave survival functions, and concave hazard functions, the problem defined by Eq. (4.9) *is convex in* $\mathbf{A}$.

(2) The optimization problem defined by Eq. (4.9) *is convex for the feature-enhanced models with exponential, Rayleigh, or power law distribution.*

(3) The solution to Eq. (4.9) *gives a consistent maximum likelihood estimator.*

The proof of Theorem 4.3.1 is similar to that of [2], and is omitted from this chapter.

We call our primary approach MONET, which provides non-splitting and splitting solutions for the network inference problem defined by Eq. (4.9) where nodes can be repeatedly infected.

Analyzing MONET. We discuss some properties of the solution to the optimization problem defined in Eq. (4.9) for the generalized feature-enhanced models with the exponential, Rayleigh, and power-law distribution. This is equivalent to maximizing the log-likelihood defined in Eq. (4.10). Clearly, the expression in Eq. (4.10) depends on the transmission rate α_{jk} and the relative time difference $t_k - t_j$ between each occurrence of node j and node k. Note that it does not depend on the absolute values of the time-stamps. In general, however, Eq. (4.10) depends on the absolute values of the feature vectors (i.e., it depends not only on the distance between observed feature vectors), due to the normalization constant γ.

As discussed in [2], Φ_1 and Φ_3 encourage sparse solutions by imposing negative weights on $\mathbf{A}$. Specifically, Φ_1 penalizes α_{jk} based on the relative time difference $t_k - t_j$ and Φ_3 penalizes α_{ki} for uninfected node i based on $T^c - t_k$ (i.e., until the horizon cut-off). Note that MONET only infers impossible edges based on 0 transmission rates. Due to finite observation window, the lack of ability to infect some node i within time T^c does not mean it is impossible to infect node i (i.e., there is no edge).

The term Φ_2 emphasizes the intuition that infected nodes must have at least one parent (appearing before them in a cascade) by which they were infected. If this is not ensured, $\Phi_2 = -\infty$ will be negatively unbounded. The additional features used in our models only affect the term Φ_2. Specifically, infected nodes tend to select those that are more similar to them as their parents. Further, in

the non-splitting case, only the first occurrence of each node in the cascade is affected by this hard constraint (further occurrences can still be explained by the parent of the first occurrence). However, simply using the first explanation can be too penalizing, and adding more parents might improve the likelihood.

Computational Aspects. As in previous work, we can parallelize the solution to the optimization problem defined in Eq. (4.9). Given a network with n nodes, this optimization problem has $O(n^2)$ variables, but the objective function can be separated into n independent sub-problems with $O(n)$ variables each. For each node $k = 1, \cdots, n$, we optimize the k-th column of the matrix $\mathbf{A}$ of transmission rates, solving for $(n-1)$ unknown transmission rates $\{\alpha_{jk}\}$ where $j \neq k$. To compute the k-th column, we only require the infection times of the nodes in those cascades where node k appears. Optimal columns are joined to form a globally optimal transmission rate matrix.

If node j never appears before node k in any cascade, we have no evidence to suggest the existence of a directed edge (j, k). That is, α_{jk} only contributes to the non-positive term Φ_2 in Eq. (4.10). Thus, in every iteration, we set α_{jk} to the optimal value 0 to simplify the objective function $L\left(\{\pi_g^1, \cdots, \pi_g^M\}; \mathbf{A}\right)$.

Any convex optimization package can be used to solve the optimization problem. However, regular packages such as CVXOPT [50] could not handle the scale of our Twitter dataset and ran out of memory. Thus, we use the limited-memory BFGS algorithm with box constraints (L-BFGS-B) [51] to solve Eq. (4.9) and Eq. (4.10) by implicitly approximating the inverse Hessian matrix. We use the box constraints to enforce the non-negativity of the transmission rates.

4.4 Experimental Results

We evaluate the performance of our models by analyzing the diffusion of information in the popular Twitter network. Using a dataset crawled from January to October 2010 that contains 9,409,063 tweets published by 66,679 Twitter users, we analyze the cascading behavior of some trending topics and try to infer the underlying network structure.

4.4.1 Experimental Setup

Dataset Description. We conduct experiments on a subset of the Twitter network, which contains 66,679 nodes and 240,637 directed links. Each node represents a Twitter user and each edge represents a following relation. Contrary to previous work [2, 19], the adjacency matrix (ground truth) of this subgraph has also been crawled and thus is entirely known. In order to identify trending topics, we group the messages posted by these users according to their Hashtags[1]. We assume that messages containing the same Hashtag form a (generalized) cascade of a particular topic. Note that certain cascades corresponding to popular Hashtags might not be explained by our generative models. For example, #iphone is a widespread Hashtag that users often proactively include in their tweets rather than passively copy from another user. This sometimes creates problems that are impossible to solve without any additional information. When a cascade is formed by many local sub-cascades and their respective time-stamps are completely mixed, there is no way to correctly extract each sub-cascade. Such global cascades provide extremely noisy input to our optimiza-

[1] Hashtags are words or phrases prefixed with the symbol # to label groups and topics.

tion framework, and significantly affect the performance of our models. Therefore, we select a subset of not-so-popular Hashtags (e.g., #Mokpo and #GagaSouthAmerica2011), which are more specific and "local". That is, we consider those "local" cascades such that if node k writes about a Hashtag at time t_k, then it must have followed (or, have copied from) some node that wrote about the same Hashtag before time t_k.

This assumption is particularly important to our experiments. Since we only have a subset of the whole Twitter network, infected nodes observed in a cascade might have copied the information from some node that does not belong to this subset of users. Since we observe that MONET performs better on the cascades where the "locality" intuition holds, we trace the propagation of 500 Hashtags (that consist of 103,148 tweets) across the Twitter network from January to October 2010[2]. The inference is focused on the top 200 users that belong to the largest number of cascades. The size of the dataset is such that this inference problem can be solved by both NETRATE and NETINF.

Feature Model. When collecting the Hashtags, we also record the entire message (or, tweet) containing the Hashtag. This represents the additional feature $\mathbf{f}_j$ for each node $j \in V$ in the generalized cascade model. In this chapter, we use two primary distance metrics associated with texts: language and Jaccard index.

- ◇ **Language.** We observe that messages belonging to the same cascade (i.e., with the same Hashtag) are often written in several languages. For example, a cascade starting with an English tweet can spread to multilingual users who post tweets in Italian or Chinese but keep the original Hash-

[2]We will release an anonymized dataset due to Twitter's data privacy policy.

tag. Intuitively, tweets in different languages, even if published closely in time, should not be considered as an implication of connectivity. Let $\ell(\cdot)$ be a function mapping a tweet to its language. We define a distance function with respect to language

$$d_L(\mathbf{f}_i, \mathbf{f}_j) = \begin{cases} 0, & \ell(\mathbf{f}_i) = \ell(\mathbf{f}_j); \\ 1, & \ell(\mathbf{f}_i) \neq \ell(\mathbf{f}_j). \end{cases}$$

The language information is computed using the n-gram model proposed in [52]. Note that this language identification algorithm provides noisy estimates.

◇ **Pairwise similarity.** We include pairwise similarity (a.k.a. Jaccard index) as another distance metric in our models. Given two tweets $\mathbf{f}_j$ and $\mathbf{f}_k$ posted by node j and node k, the distance function with respect to Jaccard index is defined as

$$d_J(\mathbf{f}_i, \mathbf{f}_j) = 1 - J_{jk} = 1 - \frac{|\mathbf{f}_j \cap \mathbf{f}_k|}{|\mathbf{f}_j \cup \mathbf{f}_k|},$$

where we consider the tweets as sets of words. Intuitively, besides the time factor, node k is more likely to have copied the information from node j if their tweets have higher similarity.

◇ **Combination.** We also consider both language and Jaccard similarity as a combined feature, defining another distance function

$$d_{L+J}(\mathbf{f}_i, \mathbf{f}_j) = w_J d_J(\mathbf{f}_i, \mathbf{f}_j) + w_L d_L(\mathbf{f}_i, \mathbf{f}_j).$$

where w_J and w_L are the weights associated with the language and the Jaccard similarity feature, respectively. Since Jaccard similarity is typically small, we use a weight w_J to ensure that its contribution is comparable to that of the language distance.

Note that the normalization constant γ in Eq. (4.5) is hard to compute, since it involves a summation over all possible messages of up to 140 characters (the maximum length allowed by Twitter). In our experiments, we consider γ as fixed and independent of $\mathbf{f}_j$. In the case of language, this is equivalent to assuming that there are roughly the same number of possible messages for any given language.

Our optimization framework contains a hierarchical set of models for the MAP inference problem: splitting / non-splitting with multiple occurrences (MONET), language (MONET+L), Jaccard index (MONET+J), and their combination (MONET+LJ).

Evaluation Measures. To evaluate the performance of our feature-enhanced models, we consider the following aspects:

- ◇ **Baseline.** We use NETRATE [2] and NETINF [19] as two baselines to compare with our models. Since repeated occurrences are not allowed in NETRATE, we keep exactly one copy of each node and remove all other duplicates from each cascade. We use the true number of edges as an input parameter for NETINF. Due to license issues with the optimization software, we do not compare with CONNIE [20] in this chapter, but its performance is comparable with that of NETRATE and NETINF according to previous literature.
- ◇ **Quantitative performance.** We use precision, recall, and F1-score to evaluate the performance of our models against the baselines. These measures focus on the number of correct pairs of nodes inferred. For example, given a pair of nodes (k, j) such that k is following j, if our method suggests that $\alpha_{jk} > 0$ (i.e., information flows from j to k), then consider it as a true

positive (TP). False positives (FP) and false negatives (FN) are defined in a similar way.

◇ **Efficiency.** We evaluate the efficiency (i.e., elapsed time required for obtaining the optimum) of our feature-enhanced models.

All algorithms are implemented using Python with the Fortran implementation of L-BFGS-B available in Scipy [53], and all experiments are performed on a machine running CentOS Linux with a 6-core Intel x5690 3.46GHZ CPU and 48GB memory.

4.4.2 Quantitative Performance

We trace the propagation of a set of 500 Hashtags that consist of 103,148 tweets across a subset of the Twitter network that contains 66,679 nodes and 240,637 directed links. We want to infer the connectivity of the top 200 users that appear in the largest number of these 500 cascades. We evaluate our models against NETRATE and NETINF by comparing the inferred network and the ground truth via three metrics: precision, recall, and F1-score. Precision is the ratio of correctly inferred edges (i.e., true positives (TP)) to all the inferred edges (i.e., true positives (TP) and false positives (FP)). Recall is the ratio of correctly inferred edges (i.e., true positives (TP)) to all the ground truth edges (i.e., true positives (TP) and false negatives (FN)). F1-score, the harmonic mean of precision and recall, measures the accuracy of the estimates. Specifically,

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP+FN}}, \quad \text{F} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The primary model MONET handles the basic scenario where nodes can have multiple occurrences in one cascade. As discussed in Section 4.2.1, MONET

can be extended to consider a set of additional features, such as language (MONET+L), Jaccard similarity (MONET+J), and both (MONET+LJ).

Upper Bound of Recall. Similar to previous models, MONET requires node j to appear at least once before node k for $\alpha_{jk} > 0$ to be possibly inferred (i.e., information flows from j to k). For our dataset, no more than 86.4% of the edges in the ground truth can be recovered given the cascades. This represents an upper bound on recall for any probabilistic model based on the causality of the diffusion process.

Exponential Model. Table 4.2 and Table 4.3 compare the precision, recall, and F1-score of our non-splitting and splitting models introduced in Section 4.2.2 with NETRATE and NETINF according to the exponential distribution (see Table 4.1). NETRATE tends to be highly conservative when estimating the connectivity of the Twitter network, and thus has good precision but very low recall. NETINF knows how many edges there are in the true network, and slightly improves over NETRATE. Without knowing the ground truth, MONET balances the precision-recall trade-off and improves the accuracy over NETRATE by 65.5% for the non-splitting case and 77.6% for the splitting case. As expected, MONET+L, MONET+J, and MONET+LJ further improve the F1-score on top of MONET with the help of additional features. In particular, MONET+LJ improves the accuracy by as much as 78.7% over NETRATE and 37.7% over NETINF for the splitting case.

Rayleigh Model. Table 4.4 and Table 4.5 compare the precision, recall, and F1-score of our non-splitting and splitting models with NETRATE and NETINF according to the Rayleigh distribution (see Table 4.1). Similarly, without knowing the ground truth, MONET balances the precision-recall trade-off and improves

Table 4.2 Performance comparison on Twitter (non-splitting / exponential).

METRIC	METHOD					
	NETINF	NETRATE	MONET	MONET+L	MONET+J	MONET+LJ
PRECISION	0.362	0.592	0.434	0.464	0.524	0.533
RECALL	0.362	0.069	0.307	0.374	0.450	0.483
F1-SCORE	**0.362**	**0.124**	**0.359**	**0.414**	**0.484**	**0.507**
TP	518	99	439	535	644	692
FP	914	62	573	618	586	606
FN	914	1333	993	897	788	740

Table 4.3 Performance comparison on Twitter (splitting / exponential).

METRIC	METHOD					
	NETINF	NETRATE	MONET	MONET+L	MONET+J	MONET+LJ
PRECISION	0.362	0.592	0.514	0.516	0.531	0.534
RECALL	0.362	0.069	0.599	0.605	0.618	0.635
F1-SCORE	**0.362**	**0.124**	**0.554**	**0.557**	**0.571**	**0.581**
TP	518	99	858	867	885	910
FP	914	62	810	812	781	793
FN	914	1333	574	565	547	522

the accuracy over NETRATE by 55.7% for the non-splitting case and 75.5% for the splitting case. MONET+L, MONET+J, and MONET+LJ further improve the F1-score on top of MONET with the help of additional features. In particular, MONET+LJ improves the accuracy by as much as 76.2% over NETRATE and 33.4% over NETINF for the splitting case.

Remarks. We have similar observations for the performance comparison according to the power-law distribution, but the tables are omitted here due to the space limitation. Our results suggest that the splitting model performs better than the non-splitting one, with much more true positives and far fewer false negatives. This suggests that the information diffusion in the Twitter network is

Table 4.4 Performance comparison on Twitter (non-splitting / Rayleigh).

METRIC	METHOD					
	NETINF	NETRATE	MONET	MONET+L	MONET+J	MONET+LJ
PRECISION	0.354	0.560	0.420	0.454	0.479	0.484
RECALL	0.354	0.072	0.218	0.262	0.286	0.294
F1-SCORE	**0.354**	**0.127**	**0.287**	**0.332**	**0.358**	**0.366**
TP	507	103	312	375	409	421
FP	925	81	430	451	445	449
FN	925	1329	1120	1057	1023	1011

Table 4.5 Performance comparison on Twitter (splitting / Rayleigh).

METRIC	METHOD					
	NETINF	NETRATE	MONET	MONET+L	MONET+J	MONET+LJ
PRECISION	0.354	0.560	0.480	0.493	0.495	0.499
RECALL	0.354	0.072	0.562	0.566	0.570	0.572
F1-SCORE	**0.354**	**0.127**	**0.518**	**0.527**	**0.530**	**0.533**
TP	507	103	805	811	816	819
FP	925	81	872	835	834	821
FN	925	1329	627	621	616	613

better approximated by a memoryless process. Further, the exponential model provides slightly more accurate estimates over the Rayleigh one. The performance improvement achieved using the language information is smaller compared to that achieved using Jaccard similarity, but MONET+L improves over MONET and MONET+LJ improves over MONET+J. This suggests that the language feature does provide some useful information, although its effectiveness is likely to be limited by the noisy estimates provided by the language detection algorithm we use in our experiments.

4.4.3 Efficiency

Solving each of the sub-problems defined in the basic model MONET (i.e., optimizing one column of the transmission rate matrix $\mathbf{A}$) takes about 2 minutes on average using L-BFGS-B with the history parameter $m = 10$. The running time, however, depends on the specific column being optimized, and ranges from a few seconds to several minutes. Introducing additional features requires an additional preprocessing time (in the order of minutes) to precompute the languages of the messages and the Jaccard indexes between the messages, but it does not significantly affect the running time of the optimization procedure.

4.5 Conclusion

In this chapter, we propose a family of feature-enhanced probabilistic models to infer the latent network structure from observations of a diffusion process. We develop a primary model called MONET with non-splitting and splitting solutions that can explain recurrent processes where nodes can be repeatedly infected (i.e., multiple occurrences in one cascade). Further, our models take into account not only the time differences between infection events, but also a richer set of features. The MAP inference problem, which still involves the optimization of a convex objective function, can be decomposed into smaller sub-problems that we can efficiently solve in parallel. Our experiments on the Twitter network show that our models successfully recover the underlying network structure, and significantly improve the performance over previous models based solely on time.

For future work, we would like to extend our models to explore the cascading processes in different types of real-world networks. Different networks introduce different features, and different features lead to different cascading behaviors. We are interested in analyzing which features are more important than others, and how they impact the performance of the associated probabilistic model.

CHAPTER 5

RELATED WORK

Traditional research in computer science has been focused primarily on problems with small input size. For instance, in the past, stores tracked the items purchased by each individual customer and gave that customer discounts for future purchases of those items. However, with the help of modern data storage facilities and novel algorithms, service providers such as Netflix are now able to, not only make predictions based on a customers past preferences, but amalgamate preferences from millions of customers to make accurate and intelligent suggestions and effectively increase sales revenue. In this chapter, we discuss several related research projects previously explored in the literature that involve the analysis and interpretation of large datasets. They present challenges and promising directions for rediscovering fundamental properties of large-scale networks that will reshape our understanding of the world.

5.1 Community Discovery

We introduce the concept of (α, β)-community in Section 2.2.1, which is a useful tool for community detection. A closely related concept to (α, β)-community is that of degree core [54–57]. Given degree d, a degree core of a graph G is a maximal connected subgraph of G in which all vertices have degree at least d. Equivalently, it is one of the connected components of the subgraph of G formed by repeatedly deleting all vertices of degree less than d. Every d-core is a $(d-1, d+1)$-community, but there are many (α, β)-communities that are not degree cores. The concept of (α, β)-community can capture some structural properties of large social networks that other methods (such as degree core)

method cannot discover. Degree cores tend to identify subsets of high-degree vertices as communities, while the concept of (α, β)-community highlights more the contrast of intra- and inter-connectivity. This is a natural type of community that we are interested in. I don't have to be a star to belong to some community, but I should belong to this community if I have (many) more connections inside this community than anybody outside this community does.

Community discovery in large social networks has attracted much interest in recent years, most of which is based on the premise that it is a matter of common experience that communities exist in these networks [27]. Community was often considered to be a subset of vertices that are densely connected internally but sparsely connected to the rest of the network [4, 11, 15, 23, 27]. For example, Newman constructed the measure of betweenness and modularity to partition a social network into disjoint communities [4, 11]. Andersen et al. [15] proposed a local graph partitioning algorithm based on personalized PageRank vectors. An information-theoretic framework was also established to obtain an optimal partition and to find communities at multiple levels [14, 58]. However, communities can overlap and may also have dense external connections. Mishra et al. [10] proposed the concept of (α, β)-community and algorithms to efficiently find such communities. Ahn et al. [36] provided a novel perspective for finding hierarchical community structure by categorizing links instead of vertices.

A range of community detection methods have been empirically evaluated and compared in [59]. Further, community detection problem has been extended to handle query-dependent cases [60]. Many studies combined link and content information for finding meaningful communities [61, 62]. The dynamic behavior of communities was also extensively explored in previous

work [63–66]. Other models have been proposed to improve the accuracy of community detection in different scenarios [67–70]. New measures have also been proposed to better evaluate the quality of community [71, 72]. For example, Zhang et al. [67] proposed a novel community detection algorithm that employs a dynamic process by contradicting network topology and topology-based propinquity. Maiya and Berger-Wolf [69] utilized a novel method based on expander graphs to sample communities in networks. Yang et al. [73] explored a dynamic stochastic block model for finding communities and their evolution in dynamic social networks.

However, most existing work on community detection has not considered the existence of core structure in many social networks. It has also been ignored that many communities actually have a large number of external connections. In Section 2.2, we demonstrate and explore the core structure, and propose a heuristic algorithm to extract cores from large networks.

5.2 Social Influence

Identifying different levels of social influence is closely related to community detection in social networks. Much of the previous work has been focused on evaluating close-knit communities and comparing their social influence in large social networks. A community was often considered to be a subset of vertices that are densely connected internally but sparsely connected to the rest of the network [4, 11, 27]. For example, the measure of betweenness and modularity were constructed to partition a social network into barely-connected disjoint communities [4, 11]. An information-theoretic framework was also established

to obtain an optimal partition and to find communities at multiple levels [14, 58]. However, communities can overlap with each other and may also have dense external connections. Mishra et al. [10] proposed the concept of (α, β)-community and algorithms to efficiently find such communities. Ahn et al. [36] provided a novel perspective for finding hierarchical community structure by categorizing links instead of vertices.

Community detection problem has been extended to handle query-dependent cases [60]. Many studies combined link and content information for finding meaningful communities [61, 62]. The dynamic behavior of communities was also extensively explored in previous work [63–66]. Other models have been proposed to improve the accuracy of community detection in different scenarios [67–70]. New measures have also been proposed to better evaluate the quality of a community [71, 72].

Various techniques have been proposed for identifying and modeling social influence in large real-world networks. For example, Crandall et al. [74] studied the interactions between social influence and selection, Tang et al. [75] analyzed topic-level social influence in large-scale networks, and Gomez-Rodriguez et al. [19] developed a method to trace paths of influence and diffusion through networks.

However, most existing work on community detection has not considered the difference between kernel and auxiliary members. The important link information from auxiliary to kernel members has also been ignored. Existing work on social influence has not considered the community structure of networks. In Chapter 3, we introduce a new problem of community kernel detection to address these issues, and propose two algorithms for solving this problem.

5.3 Network Inference and Reconstruction

A substantial amount of work has been devoted to the task of studying cascading processes in large-scale networks. Largely motivated by marketing applications, the predominant focus over the past decade has been on optimization problems, where the goal is to maximize the spread of a certain cascade through a given network, either by selecting a good subset of nodes to initiate the cascade [43] or by applying a broader set of intervention strategies such as node and edge additions [45, 48]. As networks and networked systems are playing an increasingly important role in a number of disciplines, ranging from the interconnections between financial systems to epidemiology and ecology, researchers have recently begun to consider the problem of inferring the unknown (latent) underlying network given some observed cascading behavior [2, 19, 20]. Specifically, several generative probabilistic models have been developed to explain cascading behaviors, where the task of inferring the underlying network is tractable, involving the optimization of submodular [19] or convex objective functions [2, 20]. These models have been shown to perform well on a number of synthetic datasets, but there has been very limited experimentation on real-world scenarios. Moreover, the MemeTracker dataset [19] commonly used in previous work has no ground truth.

There are several obstacles when trying to apply these models to real-world problems, such as inferring the latent structure of a social network based on the diffusion of trending topics. Specifically, cascades are often formed by a mixed set of sub-cascades and it is difficult to obtain i.i.d. samples. However, real-world cascades also present a range of new opportunities to define richer probabilistic models. Previous work combined latent features with explicit ones

to solve structural link prediction problems [76]. In Chapter 4, we propose a feature-enhanced framework to address the scenario where nodes can be repeatedly infected. We develop a family of novel probabilistic models based not only on the time intervals between infection events, but also on a set of additional features, such as the content and the language of the messages exchanged in social media.

5.4 Tracking Flow of Ideas in Scientific Literature

Remarkable development in data storage has facilitated the creation of gigantic digital document collections available for searching and downloading. When searching information in a digital document collection, the ability to identify topics with their time of appearance and to predict their evolution over time is of significant help. For example, before starting research in a specific area, scientists may quickly survey the area, determine how the topics in the area have evolved, locate important ideas and the papers that introduced those ideas. With a specific topic, scientists may find out whether it has been discussed in previous literature or it is a fairly new concept. As another example, a funding agency that manages a digital document collection may be interested in visualizing the landscape of topics in the collection to show the emergence and evolution of topics, the bursts of topics, and the interactions between different topics that change over time.

Information-seeking activities often require the ability to identify topics with their time of appearance and to track their evolution. Recently, Jo et al. [77] have developed a unique approach to achieving this goal in a time-stamped

document collection with an underlying document network which represents a wide range of digital texts available over the internet. Examples are scientific paper collections, text collections associated with social networks such as blogs and Twitter, and more generally, web documents with hyperlinks. A document collection without an explicit network can be converted into this format by connecting textually similar documents to generate a document network.

This approach emphasizes discovering the topology of topic evolution inherent in a corpus. As shown in [77], the topology inherent in the corpus carries surprisingly rich information about the evolution of topics. Topics, along with the time they start to appear in the corpus, can be identified by visiting each document in the corpus chronologically and determining if it initiates a topic. A document is considered to initiate a topic if it has a textual content that is not explained by previously discovered topics and persists in a significant number of later documents. After topics are obtained by the chronological scan, an associated graph can be built whose vertices are topics and whose edges reflect cross-citation relations between topics. Globally, this generates a rich topological map showing the landscape of topics over time. Fig. 5.1 gives the main results of [77] applying this approach to the ACM corpus. Topics in the network research area emerged in the 1980s without significant ancestors, while the areas of compiler and graphics research exhibit steady evolution with an appreciable number of topics in the early years of the ACM corpus. We can also construct an individual topic evolution graph for a given seed topic, and such a graph may contain multiple threads indicating that the seed topic has been influenced by multiple fields. The relationship between these threads may change over time as well.

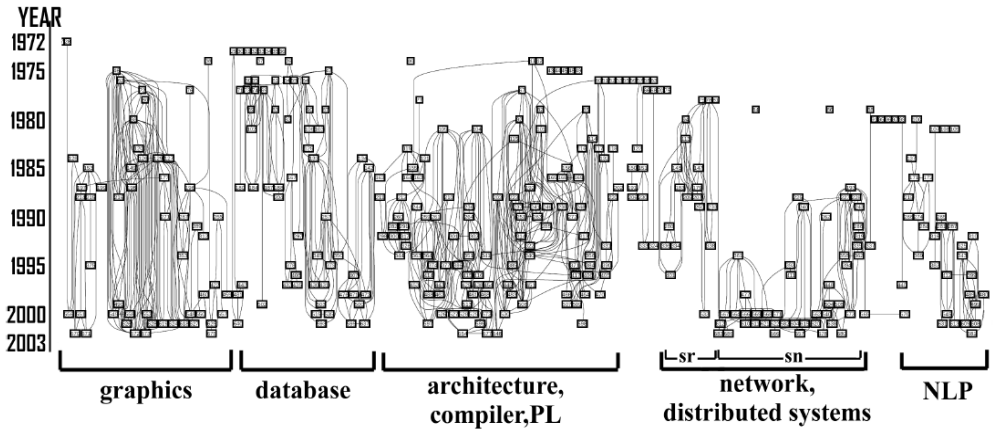


Figure 5.1 Topic evolution map of the ACM corpus.

As a related topic, content-based inferring and learning has been extensively studied recently. Various methods have been proposed to improve question-answer services in social networks [62, 78–80]. In addition, tracking popular events in social communities can be achieved using a statistical model [18].

5.5 Tracking Bird Migration in North America

Hidden Markov models (HMMs) assume a generative process for sequential data whereby a sequence of states (i.e. a sample path) is drawn from a Markov chain in a hidden experiment. Each state generates an output symbol from a given alphabet, and these output symbols constitute the sequential data (i.e., observations). The classic single path problem, solved by the Viterbi algorithm, is to find the most probable sample path given certain observations for a given Markov model [81].

Two generalizations of the single path problem for performing collective inference on Markov models are introduced in [81], motivated by an effort to

model bird migration patterns using a large database of static observations. The eBird database maintained by the Cornell Lab of Ornithology contains millions of bird observations from throughout North America reported by the general public using the eBird web application. Recorded observations include date, location, species and number of birds observed. The eBird dataset is very rich, and the human eye can easily discern migration patterns from animations showing the observations as they unfold over time on a map of North America. However, the eBird data entries are static and movement is not explicitly recorded, only the distributions at different points in time. Conclusions about migration patterns are made by the human observer, and the goal is to build a mathematical framework to infer dynamic migration models from the static eBird data. Quantitative migration models are of great scientific and practical importance. For example, this problem comes from an interdisciplinary project at Cornell University to model the possible spread of avian influenza in North America through wild bird migration.

The migratory behavior of a species of birds can be modeled by a single generative process that independently governs how individual birds fly between locations. This gives rise to the following inference problem: a hidden experiment draws many independent sample paths simultaneously from a Markov chain, and the observations reveal collective information about the sample paths at each time step, from which the observer attempts to reconstruct the paths.

Fig. 5.2 displays the pattern of ruby-throated hummingbird migration inferred by this model for the four weeks starting on the dates indicated. The top row shows the distributions and migrating paths inferred by the model: grid cells colored in lighter shades represent more birds; arrows indicate flight paths

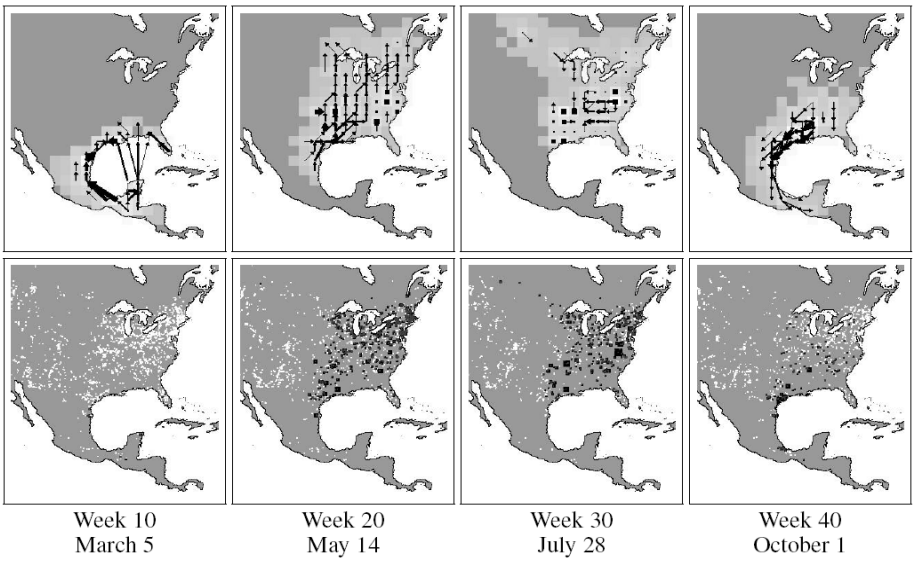


Figure 5.2 Ruby-throated hummingbird (Archilochus colubris) migration.

between the week shown and the following week, with line width proportional to bird flow. The bottom row shows the raw data for comparison: white dots indicate negative observations; black squares indicate positive observations, with size proportional to bird count; locations with both positive and negative observations appear a charcoal color. This leads to a somewhat surprising prediction that when migrating north, some hummingbirds will fly across the Gulf of Mexico while others follow the coastline, but when flying south, they generally stay above land. This prediction has been confirmed by work performed by ornithologists. For example, in the summary paragraph on migration from the Archilochus colubris species account [82], Robinson et al. write "Many fly across Gulf of Mexico, but many also follow coastal route. Routes may differ for north- and south-bound birds." The inferred distributions and paths are consistent with both seasonal ranges and written accounts of migration routes.

CHAPTER 6

CONCLUSIONS

To better understand the structure and dynamics of large social networks, we explore three major aspects of human society in this thesis from a computer science perspective that are often observed in everyday life: (1) the community structure of social networks, (2) the social difference between individuals, and (3) the underlying law that governs the propagation of information through social networks.

We propose an innovative definition of community for real-world social networks as opposed to what was generally assumed in previous studies, where communities were thought to be better connected internally than connected with the rest of the network. In fact, a community is more densely connected internally than expected from chance, but it is also connected to the rest of the network by a significant number of edges. Based on this intuition, we define two complementary terms: whiskers and the core. The type of community we would like to identify is exclusively contained in the core. Then, our community detection method consists of two steps: (1) identifying the core in which no whiskers exist, and (2) identifying communities within the core. We prove that extracting the exact core is NP-complete for both weighted and unweighted graphs, and design three heuristic algorithms to show the existence of large communities in real-world social networks.

In many social networks, (α, β)-communities of a given size k are clustered into a small number of disjoint cores, each of which is the intersection of a group of massively overlapping (α, β)-communities. Two (α, β)-communities in the same group share a significant overlap and differ by only a few vertices, while

the pairwise resemblance of two (α, β)-communities in different groups is extremely small. The number of cores decreases as k increases. The cores obtained for a small k either disappear or merge into the cores obtained for a larger k. The cores correspond to dense regions of the graph, and there are no isolated (α, β)-communities scattered between the cores. By constructing random graphs with a power-law degree distribution or the same degree distribution as that of the social graphs, we show that neither high-degree vertices nor a particular degree distribution can lead to the core structure found in many social networks. The cores found in random graphs usually have significant overlap and are increasingly scattered as the size k increases, which implies the absence of well-defined clusters in random graphs and demonstrates the existence of core structure in social networks.

A structure of community kernels and their auxiliary communities can be found in many real-world social networks that are unbalanced weakly-bipartite. Community kernels are particularly useful to distinguish different groups of social entities and to capture the common property shared by each group. We formally define the problem of detecting community kernels in large social networks. We propose a greedy algorithm and an efficient weight-balanced algorithm WEBA with guaranteed error bound for finding community kernels. The experimental results on the benchmark coauthor and wikipedia networks show that WEBA significantly improves the performance over traditional cut-based and conductance-based algorithms, since the relative influence of vertices and the link information between auxiliary and kernel members are both considered. The qualitative case study on the Twitter network further demonstrates the ability of WEBA to find meaningful community kernels, which reveal the common profession, interest, or popularity of groups of influential individuals.

We propose a family of feature-enhanced probabilistic models to infer the latent network structure from observations of a diffusion process. We develop a primary model called MONET with non-splitting and splitting solutions that can explain recurrent processes where nodes can be repeatedly infected (i.e., multiple occurrences in one cascade). Further, our models take into account not only the time differences between infection events, but also a richer set of features. The MAP inference problem, which still involves the optimization of a convex objective function, can be decomposed into smaller sub-problems that we can efficiently solve in parallel. Our experiments on the Twitter network show that our models successfully recover the underlying network structure, and significantly improve the performance over previous models based solely on time.

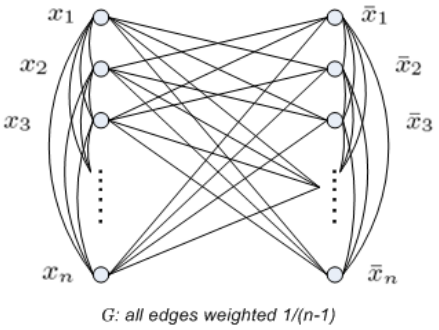
We believe that gathering, analyzing, visualizing, and interpreting large amounts of data are the most important tasks for future computer science research. As social media pervades every facet of our lives and large-scale computing becomes increasingly ubiquitous, scalable algorithms are becoming more and more necessary for efficiently handling the vast quantities of available information to better understand large social networks. In this thesis, we have discussed several research topics that address and explore important aspects of social networks. Our studies include massive data collection, mathematical definition, algorithm design, and large-scale experimentation. They also present new challenges and promising directions for future computer science research. Striking results discovered by these innovative studies imply a fundamental change in computer science that will reshape our knowledge of the world.

APPENDIX A

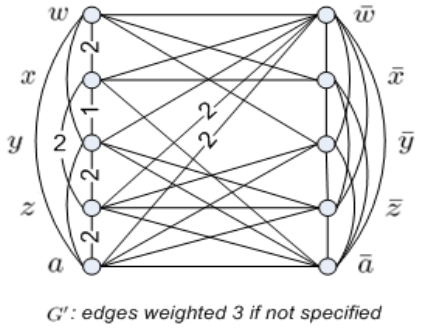
PROOF OF THEOREM 2.1.10

Proof. Given an instance of the WHISKER problem, we can guess a solution and verify in linear time whether it is indeed a whisker, thus WHISKER $\in$ NP.

Consider a weighted undirected graph G as depicted in Fig. A.1(a). A total of $2n$ vertices are arranged in two columns of n vertices each, which correspond to the literals $\{x_i, \bar{x}_i | 1 \leqslant i \leqslant n\}$. Each vertex is connected by an edge to every other vertex except its complement in the other column, and each edge is assigned weight $1/(n-1)$. Note in particular that the size of any cut in G has been generalized to the weighted sum of the cut edges. Clearly, this graph can be constructed in polynomial time.



(a) The weighted graph G. All edges are weighted by $1/(n-1)$.



(b) The weighted graph G' obtained by combining G_1, G_2, G_3 in Fig. A.2. The edges are weighted by 3 except specified. G' is complete but not all edges are displayed.

Figure A.1 Schematic illustrations of constructing G and G'.

Pick one vertex from each row and the resulting subgraph has n vertices and $n(n-1)$ cut edges. Since

$$n(n-1) \cdot \frac{1}{n-1} = n,$$

this subgraph is actually a whisker by Definition 2.1.4. Hence, there are 2^n such whiskers and we claim that no more whiskers can be found in the graph. Suppose that there is a whisker that has $2k + j$ vertices consisting of both vertices from k rows and one of the two vertices from j rows. We require that $2k + j \leqslant n$ since a whisker cannot contain more than half of the vertices. Then, the whisker gives a cut size of

$$\frac{1}{n-1}\left[2k(2n-2k-j)+j(2n-2k-j-1)\right].$$

According to the hypothesis,

$$\frac{1}{n-1}\left[(2k+j)(2n-2k-j)-j\right] \leqslant 2k+j,$$

and it follows that

$$(2k+j)(n-2k-j+1) \leqslant j. \tag{A.1}$$

For $k = 0$, the inequality holds only if $j = n$, and this simply corresponds to one of the 2^n whiskers we previously found. For $k \geqslant 1$ and $j \leqslant n-2k$, the inequality cannot hold since $2k + j \geqslant j + 2$ and $n - 2k - j + 1 \geqslant 1$. Thus, no other whiskers exist in the graph and these 2^n whiskers are also maximal whiskers. Note that the union of any two of these whiskers is no longer a whisker, since it contains more than n vertices.

Next, consider a given 3-CNF Boolean formula with c clauses and n variables. For each i ($1 \leqslant i \leqslant c$), we can construct a unit-weighted undirected graph G_i for the ith clause with $2n$ vertices arranged in two columns of n vertices each, which represent the n variables and their negations, respectively. Such a graph G_i is complete except that there are no edges between pairs of literals in the ith clause, as shown in Fig. A.2. Then, the graphs $G_1, G_2, \cdots, G_c$ can be combined into a weighted graph G' by accumulating the weight of each edge while preserving the label of each vertex, as shown in Fig. A.1(b).

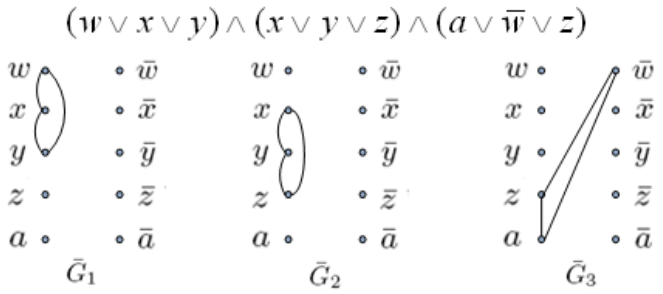


Figure A.2 The unit-weighted graphs $G_1, G_2, \cdots, G_c$ corresponding to the example clauses. For graphical simplicity, the complement of each graph is drawn instead of the original graph.

Reduce the edge weights of G by a small amount to $1/(n-\varepsilon)$, where $0 \ll \varepsilon < 1$. The 2^n whiskers have a slight excess of vertices but no new whiskers are formed, which allows more edges to be added. Scale the edge weights of G' by a small amount $\delta > 0$, where $cn^2\delta \ll 1$. Now, G and G' can be merged into a weighted graph G^*, replacing the vertex labels of G by those of G' and summing their corresponding edge weights. The whiskers in G^*, like those in G, still come from selecting one vertex from each row.

For any truth assignment, rearrange the graphs with the two columns corresponding to true literals and false literals, respectively. If there is a not-all-equal truth assignment for the Boolean formula, then each clause must have one true literal and one false literal, which indicates that the literals of each clause cannot lie within the same column of the rearranged graph. Clearly, for each i, there are $n^2 - 2$ edges connecting the two columns of G_i with no edges between the literals of the ith clause. Thus, the weighted sum of the edges connecting the two columns of G' is given by $cn^2 - 2c$.

Selecting the true literal from each row, we have a subset of n vertices that is linked to the rest of G' by $cn^2 - 2c$ edges. For this subset to be a whisker after

merging G and G', we require that

$$\frac{1}{n-\varepsilon}n(n-1)+\delta\left(cn^2-2c\right)\leqslant n. \tag{A.2a}$$

In contrast, if there is no such truth assignment for the Boolean formula, then at least one clause has its literals located within the same column of the rearranged graph. Clearly, there are n^2-2 edges connecting the two columns for at most $c-1$ of the c graphs, and n^2 edges connecting the two columns for at least one of them. Thus, the weighted sum of the edges connecting the two columns of G' is at least $(c-1)(n^2-2)+n^2=cn^2-2c+2$. Selecting one literal from each row, we have a subset of n vertices that is linked to the rest of G' by at least cn^2-2c+2 edges. For this subset not to be a whisker after merging G and G', we require that

$$\frac{1}{n-\varepsilon}n(n-1)+\delta\left(cn^2-2c+2\right)>n. \tag{A.2b}$$

Combining Equation (A.2a) and (A.2b), it follows that

$$\frac{n(1-\varepsilon)}{(n-\varepsilon)\left(cn^2-2c+2\right)}<\delta\leqslant\frac{n(1-\varepsilon)}{(n-\varepsilon)\left(cn^2-2c\right)}. \tag{A.3}$$

With ε and δ satisfying Equation (A.3) for the given c and n, the true literals of a not-all-equal assignment for the formula correspond to the vertices of a whisker in G^*, and the vertices of a whisker in G^* also correspond to the true literals of a not-all-equal assignment for the formula. Therefore, we have established a one-to-one correspondence between not-all-equal truth assignments and whiskers, that is, a weighted graph can be constructed for a given 3-CNF Boolean formula such that whiskers can be found in the graph if and only if the formula is not-all-equal satisfiable. Clearly, NAE-3-SAT reduces to WHISKER in polynomial time, thus, WHISKER is NP-complete. □

APPENDIX B

PROOF OF THEOREM 2.1.11

Proof. As shown above in the proof of Theorem 2.1.10, after we get G^*, we can enlarge the bipartite graph such that all edge weights become integers, while each vertex is replaced by a clique. The weights of all edges, except those between the literals and their negations, are at least $(k-1)(n-1)$, since the original weights are $1/(n-\varepsilon)$, where ε is a small positive number.

Then, it is guaranteed that every vertex of each clique is connected to at least one edge. In fact, almost every vertex is connected to two edges, and only a small portion of vertices is connected by only one edge. For simplicity, we assume that each clique is of size $k(n-1)$, where k is a large integer and n is the number of vertices of each column in the original graph.

(1) If a whisker contains only whole cliques, then it must be exactly one clique from each row.

Proof. Suppose a whisker contains two cliques from k rows and one clique from j rows. Then,

$$
\begin{aligned}
&(2k+j)(n-1) \leqslant n(n-1) \\
&\Rightarrow 2k+j \leqslant n \\
&2k(2n-2k-j)+j(2n-2k-j-1) \leqslant (2k+j)(n-1) \\
&\Rightarrow (2k+j)(n-2k-j+1) \leqslant j
\end{aligned}
$$

If $k \geqslant 1$, $n-2k-j+1 \geqslant 1$ and the inequalities cannot hold; If $k=0$, we have $j \leqslant n$ and $n-j+1 \geqslant 1$, the inequalities holds only when $j=n$. □

(2) If a whisker contains a partial clique, then including the rest of the clique will form another whisker.

Proof. Consider the original whisker containing a partial clique. We have $e_0 \leqslant v_0$, where e_0 is the number of cut edges and v_0 is the number of vertices inside the whisker. By including the rest of the whisker, we have a new subset of $(v_0 + n - 1 - j)$ vertices, where j is the number of vertices in the partial clique $(1 \leqslant j \leqslant n-2)$. Moreover, the number of cut edges has an upper bound of

$$e_0 - j(n-1-j) + 2(n-1-j) \leqslant v_0 - (j-2)(n-1-j) \leqslant v_0 + n - 1 - j,$$

which means the new subset is also a whisker. If we apply (2) recursively to the partial cliques, according to (1), it concludes that if a whisker contains partial cliques, then it must also be one from each row. □

(3) A whisker cannot contain partial cliques.

Proof. Proof by contradiction. If a whisker contains a certain number of partial cliques, then there must exist one whisker W_1 that can be formed by the iterative procedure described in (2), which contains one partial clique and $(n-1)$ whole cliques from different rows. Further, there exists one whisker as described in (1), which is formed by including the rest of the partial clique of W_1.

However, for such a whisker in (1), we have $e_0 = v_0$, where e_0 is the number of cut edges and v_0 is the number of vertices inside the whisker. If we move j vertices in any of its whole cliques out from the whisker, we have $(v_0 - j)$ vertices, $1 \leqslant j \leqslant n-2$. The number of cut edges has a lower bound

of

$$e_0 + j(n-1-j) - j = v_0 + j(n-1-j) - j > v_0 - j,$$

which is not a whisker, contradicting to the hypothesis that a whisker contains partial whiskers. Now, we complete the proof that a whisker cannot contain any partial cliques. □

Therefore, we also establish a one-to-one correspondence between whiskers and NAE truth assignments. Then, it is proved that finding whiskers in unweighted graphs is also NP-complete. □

BIBLIOGRAPHY

[1] S. Wu, J. M. Hofman, W. A. Mason, and D. J. Watts. Who says what to whom on twitter. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 705–714. ACM, 2011.

[2] M. Gomez-Rodriguez, D. Balduzzi, and B. Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *Proceedings of the 28th international conference on Machine learning*, ICML '11, pages 561–568. ACM, 2011.

[3] B. Shaparenko, R. Caruana, J. Gehrke, and T. Joachims. Identifying temporal patterns and key players in document collections. In *Proceedings of the IEEE ICDM Workshop on Temporal Data Mining: Algorithms, Theory and Applications*, TDM '05, pages 165–174. Citeseer, 2005.

[4] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6):066133, 2004.

[5] L. Wang, T. Lou, J. Tang, and J. E. Hopcroft. Detecting community kernels in large social networks. In *Proceedings of IEEE 11th International Conference on Data Mining*, ICDM '11, pages 784–793. IEEE, 2011.

[6] L. Wang, S. Ermon, and J. E. Hopcroft. Feature-enhanced probabilistic models for diffusion network inference. In *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, ECML-PKDD '12, 2012.

[7] J. He, J. E. Hopcroft, H. Liang, S. Suwajanakorn, and L. Wang. Detecting the structure of social networks using (α, β)-communities. *Algorithms and Models for the Web Graph*, pages 26–37, 2011.

[8] J. He H. Liang L. Wang, J. E. Hopcroft and S. Suwajanakorn. Extracting the core structure of social networks using (α, β)-community. *Internet Mathematics*, 2012.

[9] L. Wang and J. E. Hopcroft. Community structure in large complex networks. *Theory and Applications of Models of Computation*, pages 455–466, 2010.

[10] N. Mishra, R. Schreiber, I. Stanton, and R. E. Tarjan. Finding strongly-knit clusters in social networks. *Internet Mathematics*, 5(1–2), 2009.

[11] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

[12] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[13] K. Lang and S. Rao. A flow-based method for improving the expansion or conductance of graph cuts. *Integer Programming and Combinatorial Optimization*, pages 383–400, 2004.

[14] M. Rosvall and C. T. Bergstrom. An information-theoretic framework for resolving community structure in complex networks. *Proceedings of the National Academy of Sciences*, 104(18):7327, 2007.

[15] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. In *Proceedings of the 47th IEEE symposium on Foundations of computer science*, FOCS '06, pages 475–486. IEEE, 2006.

[16] S. White and P. Smyth. A spectral clustering approach to finding communities in graphs. In *Proceedings of the 5th SIAM international conference on Data mining*, SDM '05, pages 76–84, 2005.

[17] J. He, J. Hopcroft, H. Liang, S. Suwajanakorn, and L. Wang. Detecting the structure of social networks using (α, β)-communities. *Algorithms and Models for the Web Graph*, pages 26–37, 2011.

[18] C. X. Lin, B. Zhao, Q. Mei, and J. Han. Pet: a statistical model for popular events tracking in social communities. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 929–938. ACM, 2010.

[19] M. Gomez-Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(4):21, 2012.

[20] S. A. Myers and J. Leskovec. On the convexity of latent social network inference. In *Proceedings of the 24th annual conference on Neural information processing systems*, NIPS '10, Red Hook, NY, USA, 2010. Curran.

[21] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community

structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[22] M. Gaertler. Clustering. *Network analysis: Methodological foundations*, pages 178–215, 2005.

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

[24] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[25] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th annual ACM symposium on Theory of computing*, pages 216–226. ACM, 1978.

[26] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[27] J. Leskovec, K.J. Lang, A. Dasgupta, and M.W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceeding of the 17th international conference on World wide web*, WWW '08, pages 695–704. ACM, 2008.

[28] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

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

[30] M. E. J. Newman. Detecting community structure in networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):321–330, 2004.

[31] M. De Choudhury, Y. R. Lin, H. Sundaram, K. S. Candan, L. Xie, and A. Kelliher. How does the sampling strategy impact the discovery of information diffusion in social media? In *Proceedings of the 4th international AAAI conference on Weblogs and social media*, ICWSM '10, pages 34–41, 2010.

[32] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[33] J. Gehrke, P. Ginsparg, and J. Kleinberg. Overview of the 2003 kdd cup. *ACM SIGKDD Explorations Newsletter*, 5(2):149–151, 2003.

[34] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the 11th ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 177–187. ACM, 2005.

[35] V. Pareto. *Manual of political eocnomy*. Macmillan, London, 1927.

[36] Y. Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764, 2010.

[37] V. D. Blondel, J. L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of community hierarchies in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(P10008), 2008.

[38] L. Page, S. Brin, R. Motwani, and T. Winograd. *The PageRank citation ranking: Bringing order to the web*. Stanford InfoLab, 1999.

[39] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[40] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[41] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer networks*, 33(1):309–320, 2000.

[42] E. Adar and L. A. Adamic. Tracking information epidemics in blogspace. In *Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on*, pages 207–214. IEEE, 2005.

[43] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 137–146. ACM, 2003.

[44] T. Lappas, E. Terzi, D. Gunopulos, and H. Mannila. Finding effectors in social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1059–1068. ACM, 2010.

[45] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 61–70. ACM, 2002.

[46] D. J. Watts and P. S. Dodds. Influentials, networks, and public opinion formation. *Journal of Consumer Research*, 34(4):441–458, 2007.

[47] J. Wallinga and P. Teunis. Different epidemic curves for severe acute respiratory syndrome reveal similar impacts of control measures. *American Journal of Epidemiology*, 160(6):509–516, 2004.

[48] D. Sheldon, B. Dilkina, A. Elmachtoub, R. Finseth, A. Sabharwal, J. Conrad, C. Gomes, D. Shmoys, W. Allen, O. Amundsen, et al. Maximizing the spread of cascades using network design. In *Proceedings of the 26th conference on Uncertainty in artificial intelligence*, UAI '10, pages 517–526, 2010.

[49] J. F. Lawless. *Statistical models and methods for lifetime data*. Wiley New York, 1982.

[50] J. Dahl and L. Vandenberghe. CVXOPT: A Python package for convex optimization. In *Proc. Eur. Conf. Op. Res*, 2006.

[51] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

[52] W. B. Cavnar and J. M. Trenkle. N-gram-based text categorization. *Ann Arbor MI*, 48113(2):161–175, 1994.

[53] E. Jones, T. Oliphant, and P. Peterson. Scipy: Open source scientific tools for Python. `http://www.scipy.org/`, 2001.

[54] I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. k-core decomposition: A tool for the visualization of large scale networks. *Arxiv preprint cs/0504107*, 2005.

[55] I. Alvarez-Hamelin, L. Dall'Asta, A. Barrat, and A. Vespignani. k-core decomposition: A tool for the analysis of large scale internet graphs. *Arxiv preprint cs.NI/0511007*, 2005.

[56] V. Batagelj and M. Zaveršnik. Generalized cores. *Journal of the ACM (JACM)*, 5:1–8, 2002.

[57] J. Healy, J. Janssen, E. Milios, and W. Aiello. Characterization of graphs using degree cores. *Algorithms and Models for the Web Graph*, pages 137–148, 2008.

[58] S. Papadimitriou, J. Sun, C. Faloutsos, and P. Yu. Hierarchical, parameter-free community discovery. *Machine Learning and Knowledge Discovery in Databases*, pages 170–187, 2008.

[59] J. Leskovec, K. J. Lang, and M. W. Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 631–640. ACM, 2010.

[60] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 939–948. ACM, 2010.

[61] J. Gao, F. Liang, W. Fan, C. Wang, Y. Sun, and J. Han. On community outliers and their efficient detection in information networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 813–822. ACM, 2010.

[62] T. Yang, R. Jin, Y. Chi, and S. Zhu. Combining link and content for community detection: a discriminative approach. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 927–936. ACM, 2009.

[63] L. Tang, H. Liu, J. Zhang, and Z. Nazeri. Community evolution in dynamic multi-mode networks. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 677–685. ACM, 2008.

[64] C. Tantipathananandh and T. Y. Berger-Wolf. Constant-factor approximation algorithms for identifying dynamic communities. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 827–836. ACM, 2009.

[65] Y. R. Lin, J. Sun, P. Castro, R. Konuru, H. Sundaram, and A. Kelliher. Metafac: community discovery via relational hypergraph factorization. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 527–536. ACM, 2009.

[66] T. L. Fond and J. Neville. Randomization tests for distinguishing social influence and homophily effects. In *Proceedings of the 19th international conference on World wide web*, pages 601–610. ACM, 2010.

[67] Y. Zhang, J. Wang, Y. Wang, and L. Zhou. Parallel community detection on large networks with propinquity dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 997–1006. ACM, 2009.

[68] V. Satuluri and S. Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 737–746. ACM, 2009.

[69] A. S. Maiya and T. Y. Berger-Wolf. Sampling community structure. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 701–710. ACM, 2010.

[70] M. D. Choudhury, W. A. Mason, J. M. Hofman, and D. J. Watts. Inferring relevant social networks from interpersonal communication. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 301–310. ACM, 2010.

[71] J. Chen, O. R. Zaïane, and R. Goebel. Detecting communities in social networks using max-min modularity. In *Proceedings of the 9th SIAM international conference on Data mining*, SDM '09, 2009.

[72] W. Chen, Z. Liu, X. Sun, and Y. Wang. A game-theoretic framework to identify overlapping communities in social networks. *Data Mining and Knowledge Discovery*, 21(2):224–240, 2010.

[73] T. Yang, Y. Chi, S. Zhu, Y. Gong, and R. Jin. A bayesian approach toward finding communities and their evolutions in dynamic social networks. In *Proceedings of the 9th SIAM international conference on Data mining*, SDM '09, pages 990–1001, 2009.

[74] D. Crandall, D. Cosley, D. Huttenlocher, J. Kleinberg, and S. Suri. Feedback effects between similarity and social influence in online communities. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 160–168. ACM, 2008.

[75] J. Tang, J. Sun, C. Wang, and Z. Yang. Social influence analysis in large-scale

networks. In *Proceeding of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 807–816. ACM, 2009.

[76] A. Menon and C. Elkan. Link prediction via matrix factorization. *Machine Learning and Knowledge Discovery in Databases*, pages 437–452, 2011.

[77] Y. Jo, J. Hopcroft, and C. Lagoze. The web of topics: discovering the topology of topic evolution in a corpus. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 257–266. ACM, 2011.

[78] Y. Liu, J. Bian, and E. Agichtein. Predicting information seeker satisfaction in community question answering. In *Proceedings of the 31st international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 483–490. ACM, 2008.

[79] K. Wang, Z. Ming, and T. Chua. A syntactic tree matching approach to finding similar questions in community-based qa services. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 187–194. ACM, 2009.

[80] X. J. Wang, X. Tu, D. Feng, and L. Zhang. Ranking community answers by modeling question-answer relationships via analogical reasoning. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 179–186. ACM, 2009.

[81] D. Sheldon, M. A. Saleh Elmohamed, and D. Kozen. Collective inference on Markov models for modeling bird migration. In *Proceedings of the 21st annual conference on Neural information processing systems*, NIPS '07, Red Hook, NY, USA, 2007. Curran.

[82] T. R. Robinson, R. R. Sargent, and M. B. Sargent. Ruby-throated hummingbird (Archilochus colubris). In A. Poole and F. Gill, editors, *The Birds of North America*, number 204. The Academy of Natural Sciences, Philadelphia, and The American Ornithologists' Union, Washington, D.C., 1996.