Community Structure in Large-Scale Complex Networks

Mojtaba Rezvani

A thesis submitted for the degree of
Doctor of Philosophy at
The Australian National University

May 2019

Typeset in Palatino by TEX and LATEX 2ε.

Some of the results in Chapter 4 are joint work with Qing Wang and Weifa Liang. These results have also appeard in [Rezvani et al. 2018]. Chapter 5 is joint work with Weifa Liang, Wenzheng Xu, Chengfei Liu and Jeffrey Xu Yu. The content of this chapter has also appeared in [Rezvani et al. 2015; Xu et al. 2017]. Chapter 6 is joint work with Weifa Liang, Chengfei Liu and Jeffrey Xu Yu. The outcome of this chapter has appeared in [Rezvani et al. 2018]. Chapter 7 is joint work with Qing Wang and Weifa Liang.

Except where otherwise indicated, this thesis is my own original work.

Mojtaba Rezvani
23 May 2019

Dedicated to my dear parents and my dear wife.

Acknowledgements

I have been fortunate enough to have a significant number of accompanies who supported me throughout my journey towards achieving these results. It is my pleasure to express my sincere gratitude to all of them.

I owe my supervisors, Qing Wang, Brendan McKay and Alistair Rendell, a debt of appreciation for their continuous guidance and encouragement over the past few years. Qing consistently made her knowledge and support available to me during the times of confusion and gave me constructive feedback to improve the quality of my works at different stages. Brendan always shared his wisdom and experience with me and showed me new ways of thinking about a problem. Alistair also provided me with an enormous amount of support, both as a supervisor and as the director of the school.

I am indebted to many of my colleagues at the research school of computer science who provided me with extensive support, during my time in Canberra. I would like to thank professor Weifa Liang for his excellent advice on my papers and providing me with financial support during the first three years of my studies. The staff of the administration and student offices, Janette, Paul, Natalie, Harriet and Christie were extremely helpful in getting me through the shortcuts of the bureaucratic processes. Several colleagues at the Computer Science Students Association, the Student Leadership Team of the graduate and university house, and the Postgraduate and Research Students Association were beneficial in providing me with various kinds of support.

I would like to show my sincere gratitude to my family for their immense amount of support and their remarkable companionship. I am grateful for having great parents, Hossein and Khadijeh, who planted the seeds of greatness in me and cultivated my values. I cherish the understanding and extensive support that I received from my parents and my siblings, Mohammad, Mohsen, Mehdi, Morteza, Mostafa and Narges. Most of all, my exceptional gratitude goes out to my loving wife, Fazeleh, who trusted me and dedicated her endless love to me on such a long and eventful journey. She selflessly accompanied me during the ups and downs of academic research and unabatingly encouraged me throughout the obstacles and challenges of my studies. It would have been impossible to find the inspiration and motivation needed to deliver this work without the help of such an incredible lady.

Publications

Primary Publications

Contributions from work presented in this thesis have been published across multiple peer-reviewed journals and conferences. A list of publications in reverse chronological order is given below:

[1] M. Rezvani, W. Liang, C. Liu, J. X. Yu, "Efficient Detection of Overlapping Communities Using Asymmetric Triangle Cuts", *IEEE Transactions on Knowledge and Data Engineering*, 2018.

The concepts, development of the new fitness metric and algorithm, complexity analysis, implementation and evaluations of this paper are presented in Chapter 6.

[2] M. Rezvani, Q. Wang, W. Liang, "Hierarchical Community Detection in Large-Scale Social Networks", *ACM International Conference on Web Search and Data Mining*, 2018.

The concepts, development of the cohesive hierarchies and algorithms, complexity analysis, implementation and evaluations of this paper are presented in Chapter 4.

[3] W. Xu, M. Rezvani, W. Liang, J. X. Yu, and C. Liu, "Efficient Algorithms for the Identification of Top-k Structural Hole Spanners in Large Social Networks", *IEEE Transactions on Knowledge and Data Engineering*, 2017.

The development of the new models for the top-k structural hole spanners, and complexity analysis of this paper are presented in Chapter 5.

[4] M. Rezvani, W. Liang, W. Xu, and C. Liu, "Indentifying Top-k Structural Hole Spanners in Large-Scale Social Networks", *ACM International Conference on Information and Knowledge Management*, 2015.

The concepts, development of the new model and algorithms, complexity analysis, implementation and evaluations of this paper are presented in Chapter 5.

Under Review

[5] M. Rezvani, Q. Wang, W. Liang, "Community Search in Large-Scale Networks", Submitted to *IEEE Transactions on Knowledge and Data Engineering*, 2018.

The concepts, development of the new models and algorithms, complexity analysis, implementation and evaluations of this paper are presented in Chapter 7.

Secondary Publications

Two other papers were published as a join work with other collaborators. However, contributions from these publications, even though being closely related are not presented in this thesis. These are listed below:

[6] M. Rezvani, M. Rezvani, "A Robust and Fast Reputation System for Online Rating Systems", *International Conference on Web Information Systems Engineering (WISE)*, 2017.

[7] A. Bozorgi, H. Haghighi, M. S. Zahedi, M. Rezvani, "A Community-Based Algorithm for Influence Maximization Problem under the Linear Threshold Model", *Information Processing & Management*, 2016.

Abstract

Vertices in complex networks can be grouped into communities, where vertices inside communities are densely connected to each other and vertices from one community are sparsely connected to vertices in other communities. This is the so-called *community structure* in complex networks. Identifying the community structure of networks has many applications, ranging from data mining, webpage clustering and marketing to extracting proteins with the same functionality in protein-protein-interaction networks and beyond.

This thesis addresses a number of the primary problems surrounding community structure in large-scale networks. These problems generally revolve around two of the principal challenges of the area, accuracy and soundness of modelling and scalability to real-world networks. The problems include identifying top-k structural hole spanners, detecting the hierarchy of communities, detecting overlapping communities, and community search in large-scale complex networks. The thesis formally defines the cohesive hierarchies of communities in complex networks. Since scalability is a major challenge for cohesive hierarchical community detection, the thesis incorporates a network sparsification technique to leverage the network size and finds cohesive hierarchies of communities in large-scale complex networks. The problem of identifying top-k structural hole spanners is formally defined in this thesis and several scalable algorithms have been presented for this problem. Furthermore, the thesis delves into the problem of overlapping community detection and proposes an accurate fitness metric to find overlapping communities in large-scale complex networks. The thesis finally studies the problem of community search and introduces a new algorithm for community search in complex networks.

The thesis develops novel models, algorithms, and evaluation measures for these problems, and presents the experimental results of these algorithms using real-world datasets, which outperform considerably on the scalability and accuracy of the state of the art, in several cases.

Contents

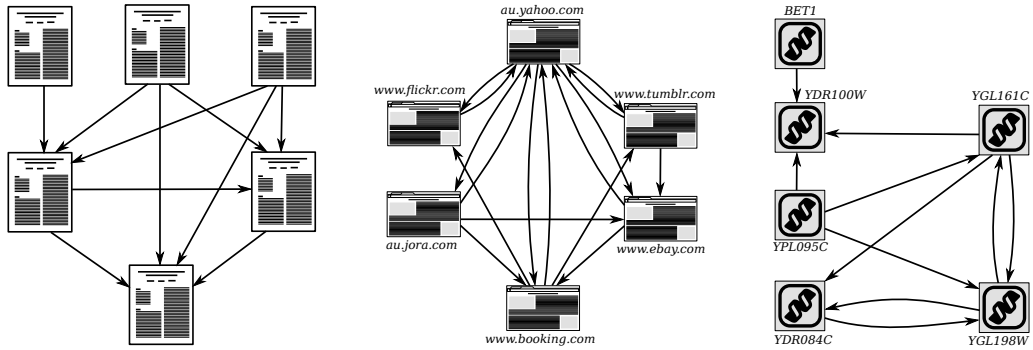
Chapter 1

Introduction

A *network*, which is also referred to as a graph in mathematics, consists of a set of entities, called *vertices*, and the relationships between those entities, called *edges*. Networks were studied as early as 1735 A.D., when Euler solved the Seven Bridges of Königsberg problem using a network model in which each land mass area was represented as a vertex, and each bridge was represented as an edge. Nowadays, networks are used as a modelling tool in a wider range of applications, such as transportation, communication infrastructures, power grids, information flow, social interactions and prediction of prospective friendships between people [Bondy et al. 1976]. The term *complex network* refers to a network that has a non-trivial topological structure which does not appear in simple networks such as lattices and cliques but frequently occurs in real-world networks.

Complex networks are quite prevalent these days, as a networks are used as common representation for a variety of complex systems [Fortunato 2010; Strogatz 2001] such as information networks [Tong et al. 2012], technological networks [Boccaletti et al. 2006], and biological networks [Fell and Wagner 2000]. A social network of people is an instance of a complex network, where members of the social network represent the set of vertices, and different types of relationships between members such as friendship, follower/followee, messaging and endorsement represent the set of edges of this network. The World Wide Web forms a network where webpages are the vertices, and the hyperlinks among webpages form the edges. Fig. 1.1a illustrates a small part of the complex network of webpages in the World Wide Web. In a similar manner, citation among research articles forms a complex network, where each research article represents a vertex, and every citation from one article to another represents an edge between the corresponding vertices. Fig. 1.1b represents a small network of research articles and the citations between them. In biology, researchers have created networks of proteins based on the chemical interactions between proteins, where proteins are represented by vertices, and there is an edge between two proteins if there is a certain chemical interaction between those proteins. The availability of such networked data has provided us with an opportunity to understand the underlying structure of these complex systems.

Despite the differences in the way these complex networks are constructed, they all share several characteristics. One of the interesting characteristics is the small-world phenomenon, which states that the longest distance between any pair of vertices is



(a) Citation network (b) World Wide Web network (c) Protein network

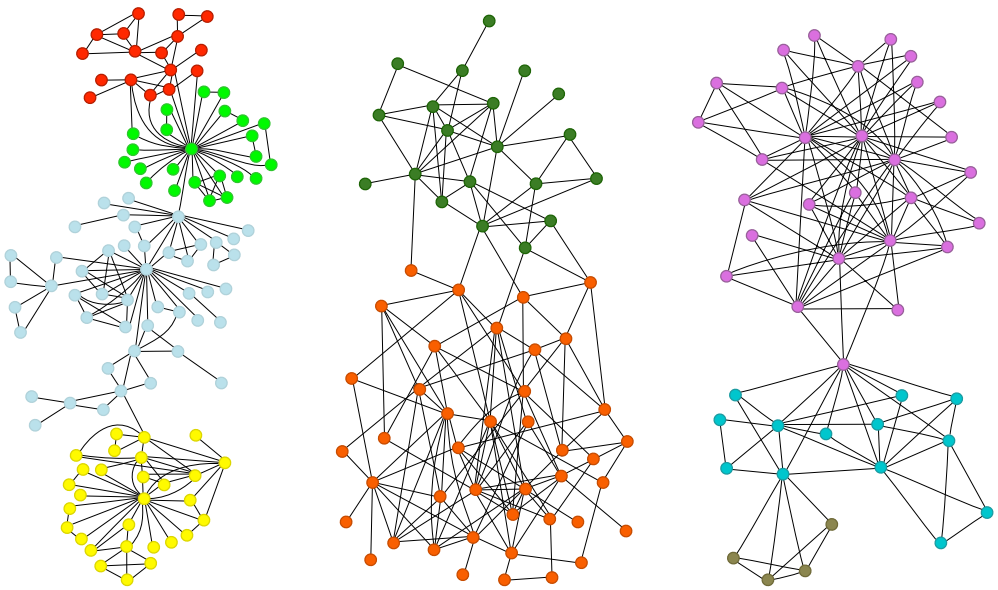
Figure 1.1: Examples of complex networks. (a) shows a citation network, which consists of research papers as vertices and citations as edges; (b) illustrates a world Wide Web network, in which webpages are represented by vertices, and hyperlinks are represented by edges; (c) demonstrates a protein-protein interaction network, where proteins are represented by vertices and there is an edge between two vertices if there is a chemical interaction between proteins.

usually a small constant [Watts and Strogatz 1998]. Furthermore, navigability is one of the important characteristics in small-world networks [Kleinberg 2000]. Complex networks often share other characteristics such as clustering coefficient and power-law degree distribution, which can help us to make sense of the non-trivial nature of such networks.

Unravelling the underlying structure of complex networks can provide us with interesting information about these networks and help us understand the complex nature of such networks. For example, the majority of members of social networks spend a noticeable amount of their time browsing the network and viewing the contents shared by others. Such information can be used in data mining applications, whereas meaningful results can be generated for different purposes including marketing, information propagation analysis and opinion mining. In World Wide Web, the hyperlink network of webpages can be used to extract useful information from the web, including webpages with high content commonality, web ranking for search engines and wbepage clustering. In academia, studying networks of researchers and their interactions can reveal popular research topics and active researchers in communities. Moreover, biologists have started realising the importance of studying the relationships between organisms and the connection patterns between proteins in protein-protein interaction networks.

1.1 Community structure in complex networks

The non-trivial topological structure of complex networks is due to an irregular distribution of edges that creates an interesting structure [Fortunato 2010], since edges appear with high density among groups of vertices, while they appear with less den-



(a) Collaboration network [Girvan and Newman 2002]

(b) Network of dolphins [Arenas et al. 2008]

(c) PPI network [Jonsson et al. 2006]

Figure 1.2: Examples of communities in complex networks.

sity between groups. This leads to the so-called *community structure* characteristic of complex networks. For instance, students in a university and their relationships can form a network, where students within the same discipline tend to be highly connected to each other, while they have a lesser tendency to connect to students in other disciplines. Similarly, studies in a network of bloggers have shown that bloggers can be grouped into communities, where bloggers that advocate a certain political party usually fall in the same community [Kumar et al. 2004]. Metabolic networks of organisms can also be decomposed into highly connected communities, where communities form a hierarchy in which communities at lower levels of the hierarchy are more cohesive, and vertices within those communities are closer to each other [Ravasz et al. 2002]. Such *hierarchical community structure* is commonly observed in other complex networks as well.

Fig. 1.2a shows a small collaboration network of some researchers at the Santa Fe Institute. It can be observed in Fig. 1.2a that different disciplinary research groups can be distinguished using the community structure of this network, since researchers of each discipline are well-separated from researchers in the other disciplines based on their collaborations. Similar communities can be found from collaboration networks of academics across the world to identify the research topics that are being pursued in academia. Fig. 1.2b represents a network of bottlenose dolphins living in New Zealand, studied by Lusseau [Lusseau 2003], where each vertex represents a dolphin, and two vertices are connected by an edge if the correspoding dolphins have been observed together. It can be seen in Fig. 1.2b that the community structure reveals the biological classification of dolphins proposed by Lusseau [Lusseau 2003]. Likewise,

the community structure of a protein-protein interaction network in Fig. 1.2c reveals that functional groups of proteins tend to have more connections to each other, while having less connection to other functional groups.

The existence of community structure in complex networks implies fewer connections between different communities, compared to connections between vertices inside communities. As a result, *structural holes* are formed, which refer to the gaps formed by the lack of connections between communities. Recent studies [Bozorgi et al. 2016; Guille et al. 2013] have shown that information, rumours and disease circulate quickly inside communities and spread to other communities through weak connections. This exposes great opportunities for individuals who bridge different communities to benefit from different sources of information and a chance to manipulate the communication between communities. Such individuals are known as *structural hole spanners* and identifying them is useful in a wide range of applications [Burt 1992].

The emergence of structural hole spanners in complex networks and the membership of vertices in multiple communities show that communities may have overlaps with each other [Xie et al. 2013]. In a social network, a person may take part in several social groups such as family, classmates and friends. However, since overlapping communities are usually highly connected to each other, identifying the overlapping regions between communities in complex networks is yet another challenging problem.

The aforementioned networks are increasing in size, and finding efficient, yet scalable algorithms for studying them is a huge challenge. In the light of the important role that communities play in many applications, it is a crucial task to find a systematic approach to detect communities in a large-scale network. As a result, the community detection problem arises in computer science [Fortunato 2010], as the problem of finding densely connected modules that are sparsely connected to other groups.

Although there is a plethora of methods for community detection in complex networks, the state-of-the-art methods suffer from either inefficiency or inaccuracy in finding communities. Due to the massive growth of networks in size, many of the existing approaches are not feasible in real-world networks. This means that such methods are incapable of finding communities in large networks with millions of vertices and edges [Yang and Leskovec 2013]. Other methods of community detection sacrifice the accuracy of the communities to obtain scalability. As a result, these methods find communities that do not match with the real community structure in networks. This thesis deals with several problems related to the community structure in complex networks, such as detection and searching of communities. The thesis pays a special attention to scalability, as well as accuracy. In the following section, the problems studied in this thesis are described.

1.2 The problems

This thesis aims to find accurate, efficient, yet scalable algorithms for studying the community structure of complex networks. More specifically, the thesis studies the

problems of hierarchical and overlapping structure of communities, identifying structural hole spanners and community search and formal defines each problem. The complexities of these proposed problems are analysed and efficient and accurate algorithms for dealing with these problems in complex networks are devised in this thesis. The proposed algorithms are scalable to networks with millions of vertices and hundreds of millions of edges. The thesis further conducts extensive experiments on a wide range of real-world and synthetic networks to evaluate the performance of the proposed algorithms and discusses the experimental results.

1.2.1 Hierarchical structure of communities

It is well-known that communities in a network often exhibit a hierarchical structure [Fortunato 2010; Ravasz et al. 2002; Sales-Pardo et al. 2007; Schaeffer 2007]. For instance, metabolic networks of organisms can be decomposed into highly connected communities, where communities form a hierarchy in which communities at lower levels of the hierarchy are more cohesive, and vertices within those communities are closer to each other [Ravasz et al. 2002]. Researchers in a collaboration network can be grouped into communities based on their research areas, from general areas such as computer science to more specific ones such as database and data mining, where information circulates more quickly among them. Therefore, small and cohesive communities are nested into larger and less cohesive communities in a hierarchical manner. Despite the indisputable role of hierarchical community detection in many applications, and with the presence of an ever-growing body of research for detecting communities in social networks, existing methods for hierarchical community detection are far from perfect. First of all, the existing methods for hierarchical community detection fail to scale in real-world networks containing millions or billions of vertices. Secondly, the existing methods for flat community detection are not directly applicable for detecting hierarchies of communities, i.e. using a flat community detection algorithm on a community often leads to the same community, instead of other sub-communities.

Traditionally, hierarchical community detection methods find hierarchies by removing/merging edges and vertices of a network one by one, which can be very time consuming. For example, Girvan and Newman [Girvan and Newman 2002; Newman and Girvan 2003] suggested starting with a given network as a community and partitioning the network by removing edges in decreasing order of their betweenness centrality. Similarly, Fortunato et al. [Fortunato et al. 2004] advocated to remove edges in order of impact of their removal on the information centrality (mean distance) of a network. Newman [Newman 2004] proposed an approach, where communities are merged if the modularity of the resulting community can be increased, starting from vertices. However, repeatedly calculating modularity, difference in modularity, betweenness and information centralities in a large network is computationally infeasible.

Furthermore, existing approaches may fail to reveal a cohesive hierarchical structure among communities. In order to capture cohesiveness, existing approaches for

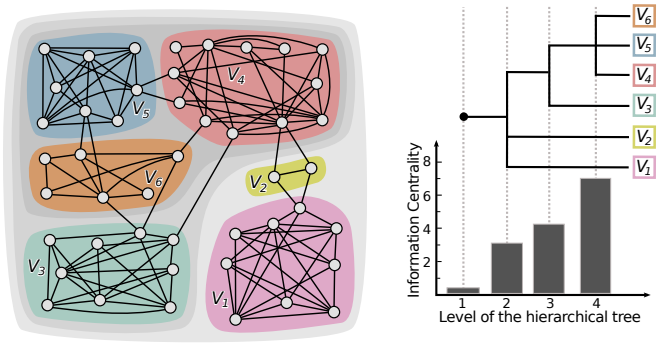


Figure 1.3: A hierarchical structure of communities in Amazon, where from the root to its leaves connections within communities become denser and the values of their information centrality increase.

flat community detection can be adjusted to detect a hierarchical structure among communities. For example in k-truss [Cohen 2008], where every edge in a community forms at least $k-2$ triangles, and k-core [Cheng et al. 2011], where the degree of every vertex in a community is at least k, one can identify a hierarchy of communities by starting with $k = 1$ and increasing the value of k to obtain more cohesive communities at the lower levels of the hierarchy. Fig. 1.3 shows a hierarchical structure of communities detected in a real-world network Amazon, where k-core and k-truss both fail to detect the hierarchical structure of communities. It can be seen that for $k < 4$, the whole network is in a single k-core community, while k-truss is unable to detect any community for $k > 4$ and $k < 3$. Thus, the aforementioned method is unable to guarantee a strong cohesion among vertices in lower levels of the hierarchy, thereby sacrificing the quality of communities for the sake of efficiency.

The first problem studied in this thesis is the problem of identifying a cohesive hierarchical structure among communities in complex networks. The main challenges for solving this problem are accuracy and efficiency in large-scale networks.

1.2.2 Structural hole spanners

In a complex network, vertices that bridge different communities gain the benefit of obtaining access to different sources of resources. It is known that communities play a significant role in information diffusion within a network; information within a community circulates very quickly and diffuses to other communities through community boundaries or bridges. There is a consensus among social scientists [Burt 1992] that a person who plays a bridge role between different communities in social networks can acquire more potential resources from these communities and has more control over the information that is being transmitted. Burt [Burt 1992] studied social structures of many organizations by introducing the notion of *structural holes* as positions that can bridge diverse groups and bring benefits to the beholder. It is shown that information within a single community tends to be homogeneous. Non-redundant information is often obtained through the contacts of people in different communities [Rinia et al.

2001]. Therefore, a person who develops relations with people from multiple communities will gain more benefits. *Structural hole spanners* were studied initially by Lou *et al.* [Lou and Tang 2013], and are referred to as a few people who fill the structural holes can bridge different communities. For example, a community in an academic collaboration network represents the group of people with the similar research interests, and people (structural hole spanners) who bridge different communities are more potent to combine ideas from different research groups and create interdisciplinary works.

Structural hole spanners have a wide range of applications. For example, in community detection, identifying central hubs that connect different groups can help isolate and identify communities [Andersen and Lang 2006; Wang et al. 2011]. In Epidemic diseases and rumors spreading, quarantining structural hole spanners can stop the spread of infection and rumors into other communities [Budak et al. 2011; Guille et al. 2013; Marathe and Vullikanti 2013]. In viral marketing, the most influential structural hole spanners can speed-up new product marketings to different groups [Kempe et al. 2003; Tang et al. 2014; Tang et al. 2013]. In graph compression, structural holes are good candidates for k-shattering [Kang and Faloutsos 2011] as they connect diverse parts of the network together, and their removal results in a network being disconnected.

Existing works on structural hole spanners can be categorised in two groups: (a) community-based approaches and (b) structure-based approaches. On the one hand, community-based approaches are highly dependent on the communities identified by community detection methods. On the other hand, structure-based approaches find structural hole spanners merely based on the topological structure of networks, and it has been shown that these approaches sometimes fail to accurately identify the structural hole spanners. For example, Goyal et al. [Goyal and Vega-Redondo 2007] considered a structural hole spanner as a vertex that lies on a large number of shortest paths, and Kleinberg et al. [Kleinberg et al. 2008] considered a structural hole spanner as a vertex that lies on a large number of shortest paths with length two. However, Fig. 1.4 illustrates that, although these two methods suggest vertex v_1 as the best structural hole spanner, vertex v_2 bridges more communities and is considered as a better structural hole spanner.

The thesis studies the problem of identifying the top-k structural hole spanners, which are the top-k vertices that bridge a wide range of communities, in a complex network with the aim of finding an algorithm that is scalable and independent of the community structure in complex networks.

1.2.3 Overlapping community detection

Recent studies [Rezvani et al. 2015; Xie et al. 2013; Yang and Leskovec 2013] have shown that some vertices in a complex network can join multiple communities to broker ideas and access resources from other communities. As a result, communities in complex networks are overlapping, rather than being exclusive with each other. The detection of overlapping communities from a complex network thus becomes a fundamental problem in the big data era, as it has many real applications. For example, for

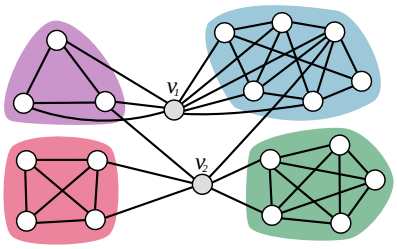
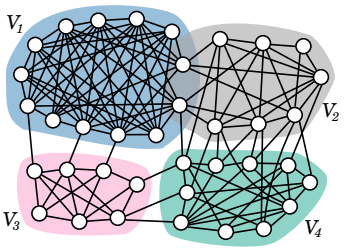


Figure 1.4: Illustration of structural hole spanners; each closed area represents a community, and vertices v_1, v_2 represent structural hole spanners that span multiple communities.

targeted advertisements in a consumer network, detecting overlapping communities helps identifying groups of members with similar shopping preferences, and they will become suitable audiences for an advertisement campaign, as they usually share several shopping preferences [Aggarwal et al. 2004]. In WWW, webpages with high content commonality can be obtained by detecting overlapping communities of hyperlink networks [Dourisboure et al. 2007]. In author-collaboration networks, communities reveal research areas and topics that are pursued by different researchers [Newman 2004]. There are many other applications of overlapping communities, including disease spread controls [Salathé and Jones 2010], product recommendations, and mining of structural hole spanners [Rezvani et al. 2015; Xu et al. 2017].

The key to identifying high-quality overlapping communities in large-scale networks is an accurate *fitness metric*, which measures the quality of identified communities, in terms of the density of internal edges within a community and sparsity of edges leaving the community. Examples of well-known fitness metrics include Classic Density [Saha et al. 2010], Relative Density [Mihail et al. 2002], Conductance [Kannan et al. 2004], Subgraph Modularity [Luo et al. 2008], and Local Modularity [Newman 2006]. We here assume that all fitness metrics measure the strength of communities. Since some of these fitness metrics (such as conductance) measure the weakness of a community (a smaller fitness value is preferred), we here use the reciprocal value of these fitness metrics to maximize the strength of communities.

Existing fitness metrics for overlapping community detection introduce the following two issues. One issue with conductance and local modularity fitness metrics is the *separation effect*, which means that an overlapping region is assigned to only one of the many communities. Another issue is the presence of the *free rider effect* [Wu et al. 2015] or *resolution limit* [Fortunato and Barthelemy 2007], which means that a community can be merged with a denser community, and the fitness value of the resulting merged community is better. Bandyopadhyay *et al.* [Bandyopadhyay et al. 2015] proposed an algorithm `FOCS` that expands neighborhoods of vertices, using the subgraph modularity fitness metric. Wu *et al.* [Wu et al. 2015] however showed that the modularity metric and other existing metrics suffer from the free rider effect [Wu et al. 2015]. Fig. 1.5 illustrates an example in which some of the existing fitness metrics cause either free rider effect or separation effect.



Fitness metrics	Detected communities	Remarks
Classic density (CD)	$V_1, V_1 \cup V_2$	Free rider effect
Relative density (RD)	$V_1, V_1 \cup V_2$	Free rider effect
Subgraph modularity (SM)	$V_1, V_1 \cup V_2$	Free rider effect
Local modularity (LM)	$V_2, V_1 - V_2$	Separation effect
Conductance (CN)	$V_1, V_2 - V_1$	Separation effect

Figure 1.5: A small network and different fitness metrics for its communities. While communities V_1 and V_2 share two vertices, fitness metrics CD, RD and SM obtain larger fitness values for communities $V_1, V_1 \cup V_2$ (free rider effect), and fitness metrics LM and CN obtain larger values for $V_2, V_1 - V_2$ and $V_1, V_2 - V_1$, respectively (separation effect).

The problem of overlapping community detection in complex network is studied in this thesis and a scalable algorithm based on asymmetric triangle-cuts is proposed for this problem that can accurately find overlapping communities, while it minimises the occurrence of free-rider and separation effects.

1.2.4 Community search

Given a number of vertices in a network, how can we find communities (i.e. densely connected subgraphs) that are related to these vertices? This is the so-called *community search* problem, which has attracted an increasing amount of attention in recent years [Barbieri et al. 2015; Cui et al. 2014; Huang et al. 2015; Shan et al. 2015a; Shan et al. 2015b; Sozio and Gionis 2010; Wu et al. 2015]. In practice, community search has a wide range of applications. For instance, in the disease control with a given set of infected members, we are interested in identifying and quarantining a group of members that are most likely to become infected, thereby stopping the spread of diseases. Similarly, in collaboration networks, one might be interested in finding communities around influential researchers in different areas to detect potential interdisciplinary areas that are forming and key researchers who shape the ideas. Communities can also be used to identify the hidden communication patterns between individuals and reveal information flows among users of social networks.

Despite the importance of community search, existing algorithms dealing with this problem have their limitations. Existing community search algorithms focus on identifying densely connected communities that contain the given query vertices. This, however, overlooks the fact that vertices in such communities may relate to the query vertices in rather different ways. For example, k-core [Cui et al. 2014] and k-truss [Huang et al. 2014; Huang et al. 2015] have been widely used to search communities in a network. As depicted in Fig. 1.6, querying k-core ($k = 3$) for two vertices u and v may result in a community E that includes many irrelevant vertices such as vertex z. This is because the minimum degree among all vertices is 3. Similarly, querying k-truss ($k = 2$) for u and v results in a community D that includes vertices that are irrelevant to v such as w, as these vertices from a different community form triangles

with u. Wu et al. [Wu et al. 2015] coined the *free rider effect* to characterize the relevance between query vertices and communities found using a fitness metric that evaluates the quality of communities. Nonetheless, the free rider effect still occurs in community search methods that do not incorporate a fitness metric, such as k-core and k-truss. In order to address the free rider issue in k-truss community search, Huang et al. [Huang et al. 2015] suggested the use of connected k-truss with minimum diameter. However, when query vertices belong to different communities, this approach also suffers from the free rider effect. Fig. 1.6 shows that searching communities for vertices u, v and z, using minimum diameter connected k-truss, returns the community F with all vertices in the figure, including x that is irrelevant to the query vertices.

Furthermore, existing methods for community search either consider queries with only one vertex, or find only one community per search. None of these methods are capable of finding communities accurately when query vertices belong to different communities. Specifically, these methods suffer from two main limitations: (i) the number of query vertices is restricted to one vertex per search [Akbas and Zhao 2017; Cai et al. 2017; Cui et al. 2014; Huang et al. 2014; Lim and Datta 2013; Liu et al. 2016]; or (ii) the number of detected communities is restricted to one community per search [Barbieri et al. 2015; Huang et al. 2015; Shan et al. 2015a; Shan et al. 2015b; Sozio and Gionis 2010; Wu et al. 2015; Zheng et al. 2017]. For example, Cui et al. [Cui et al. 2014] formulated the community search problem with a single-vertex query as finding a k-core that includes the given query vertex with maximum possible value of k. Sozio et al. [Sozio and Gionis 2010] and Barbieri et al. [Barbieri et al. 2015] attempted to address the limitation on query size by defining the problem of community search as finding only one k-core that includes a given set of query vertices, with the maximum value of k. In a similar manner, Huang et al. [Huang et al. 2014] used the notion of a connected k-truss with k-triangle-connectivity, and Shan et al. [Shan et al. 2015b] adopted the notion of γ-quasi k-clique where each edge in the community forms at least a predefined number of triangles with other edges in the community. However, in reality one may be interested in finding all relevant communities (not necessarily only one) for a group of query vertices. For instance, when querying communities for three vertices u, v and z in Fig. 1.6, a single community that contains all query vertices, such as E or F, does not reveal how these query vertices are related to each other, while two communities A and C shed light on the structure of communities for these query vertices.

The thesis finally studies the problem of community search for a given query of vertices, instead of community detection over all vertices of a network. As a result, two accurate, yet scalable algorithms for searching communities over complex networks are proposed.

1.3 Contributions

In this thesis, we study the problems introduced above, with close attention to scalability, as well as to accuracy. Since the shortest path in complex networks, to a great

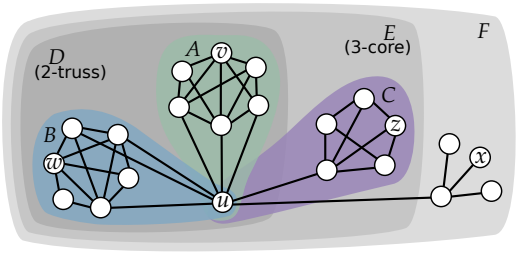


Figure 1.6: An illustrative example that shows querying k-core and k-truss communities for vertices u and v can result in communities D and E, respectively, which include vertices irrelevant to the queried ones such as w and z.

extent, reveals the underlying structure of a network, we explore how to use the shortest paths to solve each of these problems.

Hierarchical community detection. The first problem that we study in this thesis is the detection of a hierarchical community structure for complex networks. Despite the importance of hierarchical communities, existing approaches are not scalable to networks with millions of vertices or fail at accurately modelling the hierarchical structure of communities. We thus define the notion of a *cohesive hierarchy* based on an intuition about the hierarchical structure of communities, that is, communities are more densely connected to each other in lower levels of the hierarchy. Using our intuition about the hierarchical structure of communities, we formally define the problem of hierarchical community detection as the problem of identifying a cohesive hierarchy, where the sum of information centralities of detected communities is maximised. We then show that the problem of identifying a cohesive hierarchy in a network is NP-hard and propose a heuristic baseline algorithm for this problem. Using a fast sparsification technique, we develop yet another scalable algorithm for hierarchical community detection that uses a sparsified network for finding communities at higher levels of the hierarchy. We also develop an approximation algorithm for the information centrality of a given network. We finally conduct extensive experiments on real-world networks. Our experimental results show that the proposed algorithms are able to scale to networks with hundreds of millions of vertices, while outperforming existing algorithms in the accuracy of communities found and running time.

Identifying top-k structural hole spanners. Since structural hole spanners emerge in complex networks as a result of the community structure, the second problem that we study in this thesis is to identify the top-k structural hole spanners in a given network. Existing works on structural hole spanners either rely on a given set of communities in the network or fail at modelling the structural hole spanners accurately. Thus, we propose an accurate model for the top-k structural hole spanners to spot structural hole spanners without the knowledge of communities in a network. Structural hole spanners usually sit on the shortest paths between vertices in different communities. As a result, upon removal of a structural hole spanner, the lengths of shortest paths between different vertices are expected to increase significantly. We therefore define

the top-k structural hole spanners problem as identifying the top-k vertices such that their removal can result in the maximum increase in the information centrality of a network. We show that the problem of identifying top-k structural hole spanners is NP-hard. We propose an efficient algorithm for this problem using the bounded information centrality of vertices. Since articulation points are vertices that their removal can make a network disconnected, are bridging points between communities, we propose a second algorithm based on articulation points which runs in almost linear time. We validate the performance of the proposed algorithms in both real and synthetic datasets and show that the proposed approaches outperform the existing ones.

Overlapping community detection. Upon studying the overlapping community detection problem in complex networks, we realised that existing methods cause two undesirable effects in overlapping community detection, i.e. the free rider effect and the separation effect. The former causes communities to be merged with each other, instead of having overlapping regions, while the latter causes overlapping regions of two communities to be assigned to only one of the two communities. We explore these two effects in overlapping community detection and propose a fitness metric for the overlapping community detection problem based on the asymmetric triangle cuts in a network, which minimises these two effects. We then propose a two-stage algorithm for finding overlapping communities in large-scale networks. Finally, we quantitatively analyse the performance of the proposed algorithm and show that our proposed approach is capable of handling networks with hundreds of millions of vertices and finding communities with an accuracy that is significantly higher than the state-of-the-art algorithms.

Community search. Last but not least, we study the problem of community search in complex networks, in which a set of query vertices is given and we are interested in finding a community that consists of vertices that are mostly related to the queried vertices. We show that when query vertices belong to different communities, the existing works on the community search problem have the following two limitations: (1) they can find only one community for the query vertices, (2) they cause the free rider effect. We first define the notion of propinquity between two vertices, which shows how closely related two vertices are. We then formally define the community search problem, using the notion of propinquity, where two query vertices belong to the same community if the propinquity between them is larger than a given threshold. We show that the existing works on community search are special cases of the defined community search problem, by different means of propinquity. We propose a novel propinquity measure, which leads to accurate guesses about whether two query vertices belong to the same community. Since the shortest path between two query vertices in the same community is unlikely to leave the community, we use the top-k shortest paths between query vertices to expand their communities. Next, we propose an efficient algorithm that is capable of finding more than one community per search, by discovering the top-k shortest paths between query vertices. We finally evaluate the performance of the proposed algorithm in real-world networks and show its superior results in realistic settings.

1.4 Outline of this thesis

This thesis is organised as follows. In Chapter 2, we introduce the notations, based on which this thesis is written. In Chapter 3, we provide a comprehensive literature review of the problems studied in this thesis by outlining the strengths and weaknesses of the existing approaches. In Chapter 4, we study the problem of hierarchical community detection in large-scale networks. In Chapter 5, we study the problem of identifying the top-k structural hole spanners in large-scale networks and in Chapter 6, we study the overlapping community detection problem. In Chapter 7, we study the community search problem. We finally conclude the thesis in Chapter 8.

In each chapter, we start with an overview of the specific problem studied. We then propose the models and the algorithms for solving the problems. We finish each chapter by presenting our experimental results and a summary of the chapter.

Chapter 2

Preliminaries

We model a network as an undirected graph $G = (V, E)$, where V is a set of vertices, and E is a set of edges representing relationships between vertices. Let $n = |V|$ and $m = |E|$. We denote by $N(v)$ the set of neighbours of vertex $v \in V$, i.e. $N(v) = \{u : u \in V, (u, v) \in E\}$, and $\deg(v) = |N(v)|$ the degree of vertex v in G. We denote by $\Delta(G)$ the largest degree of vertices in graph G. Given a subset of vertices $V' \subseteq V$, the induced subgraph of V', denoted by $G[V']$, is a subgraph of G that its set of vertices is V' and there is an edge between two vertices in $G[V']$, if and only if there is an edge between corresponding vertices in G. For the community search problem, a query Q is given that consists of vertices, i.e. $Q \subseteq V$. Given a set of vertices Q, $\mathcal{C} = \{V_1, \cdots, V_t\}$ is called a *cover* of Q if and only if $Q \subseteq \bigcup_{V_i \in \mathcal{C}} V_i$. A cover $\mathcal{C}$ is called *minimum*, if its cardinality is smallest among all possible covers of Q.

Given two vertices u and v, a path P between them is said to be *simple* if it does not contain a self-loop or a cycle. We assume that G is an unweighted graph, and each edge has a weight of 1. The *distance* d_{uv} between two vertices u and v in G is the length of the shortest path between them. We have $d_{vv} = 0$ for any vertex $v \in V$. The *inverse closeness centrality* of a vertex v in G is the average distance between vertex v and other vertices [Beauchamp 1965], i.e.,

$$d(v) = \frac{\sum\limits_{u \in V} d_{uv}}{n - 1}. \tag{2.1}$$

The *mean distance* that is the average of the distances between all pairs of vertices in a graph G is defined as follows.

$$d(G) = \frac{\sum\limits_{v \in V} d(v)}{|V|} = \frac{\sum\limits_{v \in V} \sum\limits_{u \in V} d_{uv}}{(n-1)|V|} = \frac{\sum\limits_{v \in V} \sum\limits_{u \in V} d_{uv}}{n(n-1)}. \tag{2.2}$$

Note that if G is disconnected, the distance between two vertices in different connected components is defined by a sufficiently large value ζ to avoid the infinite distance. This value should be larger than the sum of lengths of all pairs of shortest paths in any connected component of G, e.g., $\zeta = n^3$, as the upper bound on the sum of lengths of all pairs of shortest paths in a n-vertex graph is no more than $\zeta/3$ [Plesník 1984].

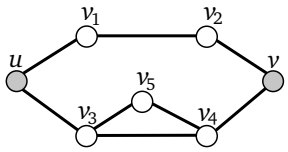


Figure 2.1: Top-3 shortest paths between two vertices u and v.

The sum of lengths of all pairs shortest paths in G is

$$D(G) = \sum_{u \in V} \sum_{v \in V} d_{uv} = n(n-1)d(G). \tag{2.3}$$

The *information centrality* of G, denoted by $\mathcal{D}(G)$, is the inverse of mean distance between every pair of vertices u and v [Fortunato et al. 2004], i.e.,

$$\mathcal{D}(G) = \frac{n(n-1)}{\sum\limits_{v \in V} \sum\limits_{u \in V} d_{uv}}. \tag{2.4}$$

The information centrality of a partition of vertices P is defined as the sum of information centralities of all subgraphs induced by each subset of vertices in the partition, i.e. $\mathcal{D}(P) = \sum_{S \in P} \mathcal{D}(S)$. We use $\mathcal{P}_{st}$ to refer to the set of all simple paths between s and t. The *k-th shortest path* between s and t, denoted by $P_{st}^{(k)}$, refers to the k-th path in $\mathcal{P}_{st}$, ordered by length of paths, where ties are broken arbitrarily. We denote by $d_{st}^{(k)}$ the length of the k-th shortest path between vertices s and t in G, i.e. $d_{st}^{(k)} = |P_{st}^{(k)}|$. It is noted that the top-k shortest paths between s and t in G can share edges and vertices. Fig. 2.1 shows an example of the top-k shortest paths between vertices in a network. The *diameter* of a network G is the greatest distance between a pair of vertices in G. The diameter of a set of vertices V' is defined as the diameter of the induced subgraph $G[V']$.

Example 1. *Fig. 2.1 shows top-3 shortest paths between two vertices u and v, including $P_{st}^{(1)} = (u, v_1, v_2, v)$, $P_{st}^{(2)} = (u, v_3, v_4, v)$, and $P_{uv}^{(3)} = (u, v_3, v_4, v_5, v)$ with lengths $d_{uv}^{(1)} = 3$, $d_{uv}^{(2)} = 3$, and $d_{uv}^{(3)} = 4$. Note that the first two shortest paths $P_{uv}^{(1)}$ and $P_{uv}^{(2)}$ have the same length 3, while the third shortest path $P_{uv}^{(3)}$ has the length 4. The edges (u, v_3) and (v_4, v) in $P_{uv}^{(3)}$ overlap with $P_{uv}^{(2)}$, as well as vertices v_3 and v_4.*

Two simple paths P_1 and P_2 between u and v are said to be *edge-disjoint* (resp. *vertex-disjoint*) if they do not share any edges (resp. vertices), except u and v. The number of edge-disjoint paths between two vertices u and v is the *edge-connectivity* between them, denoted by $\lambda(u, v)$. Two vertices u and v are said to be *k-edge-connected* if $\lambda(u, v) \geq k$. Note that two k-edge-connected vertices remain connected whenever fewer than k edges are removed from G. The edge-connectivity $\lambda(u, v)$ also represents the minimum edge-cut, which is the minimum number of edges whose removal can disconnect two vertices u and v. Similarly, the *vertex connectivity* $\kappa(u, v)$ in G between two vertices u and v is the number of vertex-disjoint paths between them, and a subgraph is called *k-vertex-connected* if the minimum vertex-connectivity among its

Table 2.1: Summary of notations used in the thesis.

Notation	Definition
$G = (V, E)$	Network G with the set of vertices V and edges E
n	Number of vertices in a network V
m	Number of edges in a network E
$N(v)$	Set of vertices adjacent to vertex v
$deg(v)$	Degree of vertex v
$\Delta(G)$	The largest degree of vertices in graph G
$G[V']$	Subgraph of G induced by a subset of vertices V'
$\mathcal{P}_{uv}$	Set of simple paths between u and v
$P_{uv}^{(k)}$	The k-th shortest path between u and v
d_{uv}	The length of the shortest path between u and v
$d(v)$	The average distance between vertex v and other vertices
$d(G)$	The average length of shortest path between vertices in G
$D(G)$	The sum of length of shortest paths between vertices in G
$\mathcal{D}(G)$	The information centrality of G
$\lambda(u, v)$	Edge-connectivity between u and v
$\kappa(u, v)$	Vertex-connectivity between u and v
$E(V', V'')$	Cut-set between subsets of vertices V' and V''
$e(V', V'')$	Size of cut-set between subsets of vertices V' and V''
$E(V')$	Set of edges between vertices in V'
$e(V')$	Number of edges between vertices in V'
$sup_G(e)$	Number of triangles formed by edge e in graph G
$\Delta_G(V')$	Set of triangles formed by vertices in V'
$\Delta_G(V', V'')$	Set of triangles that have two vertices in V' and one vertex in V''
$vol(V')$	Sum of degrees of all vertices in V'

vertices is no smaller than k. A vertex is an *articulation point* of G if its removal will disconnect the graph.

Let $E(V', V'')$ be a *cut-set* between subsets V' and V'' of vertices, which is the set of edges that have one vertex in V' and the other vertex in V'', and $e(V', V'')$ the size of such cut-set, i.e., $e(V', V'') = |E(V', V'')|$. Let $e(V')$ represent the number of edges in the induced subgraph $G[V'] = (V', E[V'])$ of a graph G by the vertices in V'. The *volume* of a subset of vertices $V' \subseteq V$ is defined as $vol(V') = \sum_{v \in V'} |N(v)|$. Denote by Δ_{uvw} the triangle that consists of vertices u, v and w, and denote by Δ_G the set of triangles in G. Given a subset $V' \subseteq V$ of vertices, $\Delta_G(V')$ represents the set of triangles formed by the vertices in V'. The *asymmetric triangle cut* $\Delta_G(V', V'')$ between two sets of vertices V' and V'' is defined as the set of triangles with each having two vertices in V' and one vertex in V''. Note that $\Delta_G(V', V'')$ is not necessarily equal to $\Delta_G(V'', V')$.

A p-clique K_p is a complete graph of p vertices. A *triangle* is a cycle of length three. Table 2.1 summarises the definitions of all notations used in this thesis.

The key to identifying communities in a complex network is an accurate *fitness metric*, which measures the quality of identified communities, in terms of the density

of internal edges within a community and sparsity of edges leaving the community. We here assume that all fitness metrics measure the strength of communities. Since some fitness metrics (such as conductance) measure the weakness of a community (a smaller fitness value is preferred), we here use the reciprocal value of these fitness metrics to maximize the strength of communities. In the following, we summarise some of the existing fitness metrics for community detection.

The vertices in a graph $G = (V, E)$ can be allocated to different communities. Let $\mathcal{V} = \{V_1, ..., V_q\}$ be a collection of all communities in G, where $V_i \subseteq V$ is a community, $\cup_{i=1}^{q} V_i = V$, and $V_i \cap V_j$ may and may not be empty ($i \neq j$), $1 \leq i, j \leq q$. The *fitness metric* of a community $V_i \in \mathcal{V}$, denoted by $f(V_i)$, will determine the degree to which vertices inside V_i are connected with each other, while separating from the vertices in $V \setminus V_i$. The fitness metric of a collection of communities $\mathcal{V}$ thus is defined as a summation over the fitness values of all communities in the collection, i.e., $f(\mathcal{V}) = \sum_{V_i \in \mathcal{V}} f(V_i)$. In the following we introduce several widely-adopted fitness metrics for community detection [Fortunato 2010; Xie et al. 2013].

- *Classic density* $\delta(V_i)$ of a community V_i [Saha et al. 2010] is referred to as the average degree of vertices within the community V_i, i.e., $\delta(V_i) = e(V_i)/|V_i|$, where $e(V_i)$ is the number of edges in the subgraph induced by vertices in V_i.

- *Relative density* $\rho(V_i)$ of a community V_i [Mihail et al. 2002] is referred to as the ratio $e(V_i)$ of the number of edges in community V_i to the number of edges that have at least one vertex in V_i, i.e., $\rho(V_i) = e(V_i)/(e(V_i) + e(V_i, V \setminus V_i))$.

- *Subgraph modularity* $\psi(V_i)$ of a community C [Luo et al. 2008; Wu et al. 2015] is referred to as the ratio of the number of edges in community C to the number of edges between vertices in V_i and the vertices in $V \setminus V_i$, i.e., $\psi(V_i) = e(V_i)/e(V_i, V \setminus V_i)$. Note that this subgraph modularity [Luo et al. 2008] is a variant of the traditional modularity [Newman 2006].

- *Local modularity* $\mu(V_i)$ of a community V_i [Newman 2006] is the ratio of the number of edges in V_i between boundary vertices and other vertices in V_i to the number of edges between boundary vertices in V_i and all other vertices in the network, i.e., $\mu(V_i) = e(B(V_i), V_i)/(B(V_i), V)$, where $B(V_i)$ is the boundary set of vertices in V_i, which are adjacent to at least one vertex outside V_i.

- *Conductance* $\sigma(V_i)$ of a community V_i [Kannan et al. 2004] is referred to as the ratio of the size of the edge cut to the minimum of the number of edges that have at least one endpoint in V_i and number of edges that have at least one endpoint in $V \setminus V_i$, i.e., $\sigma(V_i) = e(V_i, V \setminus V_i)/\min\{vol(V_i), vol(V \setminus V_i)\}$. Unlike other fitness metrics, smaller values of conductance are preferred for a community. Therefore, we consider the inverse of this value, and throughout this chapter we refer to conductance as $\sigma'(V_i) = \min\{vol(V_i), vol(V \setminus V_i)\}/e(V_i, V \setminus V_i)$.

Chapter 3

Literature Review

It has been shown that one of the key characteristics of complex networks is the community structure among vertices, by which vertices form modular groups, called communities, where vertices within the same community are highly connected to each other, while vertices in different community are rarely connected to each other. Due to the importance of community structure, several problems arise in complex network analysis related to the community structure. One of the problems is the community detection problem, which deals with identifying communities of a given networks. Since communities in a complex network can inherit hierarchical and overlapping structures, community detection has been studied in a variety of forms, e.g. hierarchical community detection and overlapping community detection. Since community structure in complex networks leads to the existence of structural hole spanners, another problem that is related to community structure is the problem of identifying structural hole spanners in a complex networks. Community search is yet another problem that arise as a result of community structure in complex networks. While the community detection problem is concerned with finding communities with respect to the whole set of vertices in a complex network, the community search problem is concerned with finding communities that are related to a given set of query vertices. In the following, we provide a comprehensive literature review on hierarchical and overlapping community detection, identifying structural hole spanners and community search. We analyse the weaknesses and strengths of the existing works on these problems.

3.1 Community detection

Community detection in complex networks has received an enormous amount of attention in recent years [Fortunato 2010; Xie et al. 2013], which has led to a large number of methods available for community detection. Each of these methods has its own merits and demerits. Fortunato [Fortunato 2010] surveyed some of the pioneering works on community detection and related problems, including overlapping community detection and hierarchical community detection. However, the proliferation of complex networks and the wide range of applications of community detection have posed new challenges, such as scalability, which have lead to an ever increasing

growth of research on the community detection.

Among the plethora of works on community detection, the main focus has mostly been on either the accuracy of communities, or the efficiency and scalability of the community detection algorithms. Due to the massive growth of networks in size, many of the existing approaches are not feasible in real-world networks, or they sacrifice the accuracy of the communities to obtain scalability. In this thesis, we study the community detection problem with a special attention to the scalability of the proposed algorithms. Although we care about the accuracy of the proposed models, we also care about the efficiency of the proposed algorithms and feasibility of using the proposed approaches in real-world networks.

There is a variety of problems rooted in community detection. For instance, it is widely known that communities in complex networks exhibit a hierarchical structure. Although a large number of attempts have been made to develop hierarchical community detection algorithms, existing methods fail in scalability and efficiency. Another inherent characteristic of communities is the weak connectivity between vertices in different communities. As a result, structural hole spanners bridge different communities. However, existing works on identifying structural hole spanners mainly rely on a given community structure in networks. Structural hole spanners also lead to overlapping regions between communities and poses the problem of overlapping community detection. In this section, we review the existing works on three problems as follows: (1) hierarchical community detection, (2) identifying structural hole spanners, and (3) overlapping community detection.

3.1.1 Hierarchical structural community detection

In recent years, considerable efforts have been taken in building efficient metrics and models that can accurately capture the properties of communities in real complex networks. In their comprehensive surveys, Xie *et al.* [Xie et al. 2013], Furtano [Fortunato 2010] and Shaeffer [Schaeffer 2007], surveyed state-of-the-art algorithms for community detection. Existing methods for detectiing the hierarchical structure of communities can be divided into two categories: *top-down approaches* and *bottom-up approaches* [Schaeffer 2007]. While the top-down approaches start with a given graph and partition the graph into successively denser communities, the bottom-up approaches start with seeds, i.e. vertex seeds [Whang et al. 2013] or community seeds [Lancichinetti et al. 2009], and expand those seeds gradually by merging them into each other. In the following, we review some of the most influential works in each of these categories.

Top-down approaches. Given an undirected graph G, top-down approaches start with the whole network as a community and break the network into communities by removing edges or vertices, until a certain condition is met. For example, Newman et al. [Girvan and Newman 2002; Newman and Girvan 2003] suggested to use the betweenness centrality score of edges as a criterion for removing edges and breaking a graph into partitions. However, the time-complexity of calculating betweenness centrality in a network is quite high $\Theta(nm)$, thus the overall time-complexity of this

approach is $\Theta(nm^2)$, which is not practical in real networks. Similarly, Fortunato et al. [Fortunato et al. 2004] exploited the information centrality of a network as a criteria for deciding which edges to be removed at each iteration. Fortunato et al. [Fortunato et al. 2004] suggested to iteratively remove the edges, whose removal will result in the maximum decrease in the information centrality of a network. While the measure used for the information centrality is inaccurate, the time-complexity of the proposed algorithm is $\Theta(nm^3)$, which is infeasible in networks that contain more than a few thousands of vertices.

Another line of work in the top-down approaches used several density measures as a criteria for deciding which edges to be removed in a network. For example, Cohen [Cohen 2008] suggested the notion of a k-truss, where every edge in a community forms at least k triangles with other edges in the community, and suggested an algorithm with the overall time complexity of $O(nm^{3/2})$. However, several types of networks (such as product-buyer networks) do not have any triangles. In order to tackle this problem, Zhou et al. [Zhou et al. 2012] suggested the notion of k-edge-connectivity in a network, and defined a community as a subgraph, in which every pair of vertices are k-edge-connected. While the k-edge-connectivity is general enough to be applicable to several types of networks, the distance in a k-edge-connected community is not bounded compared with the size of the detected community.

Bottom-up approaches. Unlike top-down approaches, bottom-up approaches start with seeds and expand those seeds gradually, until a certain threshold is met. However, one challenge is to choose appropriate seeds for the community expansion phase. For example, the clique expansion method [Lee et al. 2010] identifies distinct cliques as initial seeds, and then expands the seeds greedily using a local fitness metric. In clique percolations [Palla et al. 2005], a community is defined as the maximal union of maximal cliques that can reach each other through a series of adjacent maximal cliques. However, since some complex networks, such as collaboration networks, are funamentally a union of cliques, this model may consider the whole network as a single community. To tackle this issue, Shen et al. proposed an algorithm called EAGLE [Shen et al. 2009], which merges two communities with the maximum similarity into one, where the similarity between two communities is proportional to the number of edges between them. Du et al. proposed the algorithm COCD [Du et al. 2008], in which seeds are a set of maximal cliques and two maximal cliques are merged if their similarity is positive, where the similarity between two cliques is a proportional to the number of edges between non-overlapping vertices of those cliques. Even though cliques can guarantee a very strong connectivity among its members, it is considered as a very strict condition for real-world communities. Therefore, Lancichinetti et al. proposed the algorithm LFM [Lancichinetti et al. 2009], where random seeds are expanded until the value of a fitness function based on the number of edges in the community is locally maximal. Similarly, Whang *et al.* [Whang et al. 2013] used a personalized PageRank algorithm for finding cuts between communities, where a random walk in a network can start from vertex seeds only. Whang *et al.* [Whang et al. 2013] suggested the use of vertices with maximum degree, and dominating sets and

random vertices as seeds for community expansion. Considering that networks contain many vertices that act as hubs and connect several communities, using vertices as seeds can affect the outcome of the algorithms. Therefore, finding an appropriate seed in bottom-up approaches ends up in an judgement call that is difficult to make, due to difference in topology of networks.

3.1.2 Identifying structural hole spanners

While real complex networks are increasing in size at an exponential rate, building accurate models that truly reflect the properties of structural hole spanners is crucial for identifying influential positions in a network. Nevertheless, researchers have taken lots of efforts towards this aim [Ahuja 2000; Burt 2004; Burt 2007; Goyal and Vega-Redondo 2007; Kleinberg et al. 2008; Tang et al. 2012; Tong et al. 2012].

The notion of structural hole spanners was first introduced by Burt [Burt 1992] to find the key employees in organizations for integrating operations across functional and business boundaries. This concept later was further refined in [Ahuja 2000; Burt 2004; Burt 2007]. A few studies have exploited the concept of structural holes in order to design strategic games for network formation [Goyal and Vega-Redondo 2007; Kleinberg et al. 2008]. Goyal *et al.* [Goyal and Vega-Redondo 2007] presented a network formation model that a vertex u serves as an intermediary between many vertices. However, this strategy leads to the star network and real networks do not follow a star topology. In order to tackle this problem, Kleinberg *et al.* [Kleinberg et al. 2008] designed a game by building a model of the payoffs that arise from filling structural holes. This payoff is a decreasing function of the number of paths with length two between each pair of neighbors to avoid the star topology. One of the limitations of the model presented by Kleinberg *et al.* [Kleinberg et al. 2008] is that this model needs careful tuning of parameters such as the link maintenance cost that cannot easily be achieved in large-scale networks. Another line of research is to find structural hole spanners in order to incorporate them in contagion. These works can be divided in two categories as follows.

Structural-based Models. Goyal *et al.* [Goyal and Vega-Redondo 2007] formulated a structural hole spanner as a vertex that resides on more shortest paths between different pairs of vertices. Since counting the number of shortest paths in large networks is time-consuming, Tang *et al.* [Tang et al. 2012] proposed to only count the number of shortest paths with length two on which a vertex lies. In this model, any shortest path of length greater than two will be ignored, thus the model suggests candidates that are connected to smaller rather than larger, richer and more influential communities. A fairly common case under this model is its failure of finding good quality structural hole spanners when a vertex is densely connected to two communities. In order to address this problem, Ugander *et al.* [Ugander et al. 2012] defined the *structural diversity* of an individual as the number of connected components in its contact neighborhood, which is a similar notion as structural hole spanners, and studied the role of structural diversity in contagion of information within real social networks. This work was further complemented by Huang *et al.* [Huang et al. 2013], who studied the top-k

structural diversity search in large networks and developed scalable algorithms for it. However, the contact neighbourhood subgraph may contain only a fraction of the vertices that lie in each community, and due to the incomplete community structure, these vertices can form multiple connected components. Similarly, Tong *et al.* [Tong et al. 2012] defined the gateway-ness of a vertex v, proportional to the paths between source vertices S and target vertices T, on which v lies. In addition, each path is given a score, which is inversely proportional to its length.

Community-based Models. Lou *et al.* [Lou and Tang 2013] proposed the very first model to find structural holes in a social network, assuming that communities in the network are given. The objective in their model is to maximize a utility function that measures the degree to which structural hole spanners span communities. One instantiation of their utility function is to find a set of vertices whose removal leads to the maximum decreases on the number of inter-community edges. One major concern about this model is that communities usually are not known, thus the quality of the solution relies on the quality of communities found.

3.1.3 Overlapping community detection

Xie *et al.* [Xie et al. 2013] surveyed state-of-the-art algorithms for overlapping community detection and categorized the algorithms into five categories: link partitioning, fuzzy community detection, agent-based algorithms, and local expansion as follows.

Given an undirected graph $G = (V, E)$, link partitioning algorithms [Ahn et al. 2010] construct the line graph of G, and find vertex-disjoint communities in the line graph that correspond to the overlapping communities in G. However, the line graph of a graph is usually very large, for example, the line graph of a star graph is a clique, link partitioning algorithms are not scalable for a reasonable size network in practice due to the massive size of the line graph.

Fuzzy community detection algorithms employ a soft clustering technique for identifying communities [Xie et al. 2013]. One representative example is algorithm `Bigclam` [Yang and Leskovec 2013] which exploits a non-negative matrix factorization framework for finding overlapping communities. Nepusz *et al.* [Nepusz et al. 2008] modeled the overlapping community detection problem as a nonlinear constrained optimization problem, and solved the problem using simulated annealing. One noticeable drawback of such algorithms is the need to determine the number of communities in a network [Eustace et al. 2015].

In agent-based methods, each vertex is considered as an agent that transmits messages to other vertices and/or receives messages and joins communities. For instance, Xie et al. proposed algorithm SLPA, in which each vertex can be a listener or a speaker. It spreads vertex labels across the network based on pairwise interaction rules and the probability of observing a vertex label in another vertex's memory determines the community membership. Coscia et al. [Coscia et al. 2012] proposed another agent-based method algorithm called Demon, which extracts the ego network of each vertex and applies a label propagation algorithm on this structure, ignoring the presence of the ego itself, then each vertex acquires a label that appears most frequently among

its neighbors to extract the communities [Coscia et al. 2012]. However, agent-based algorithms are very time consuming in real-world networks. Beside the running time, if a vertex v belongs to two communities, it is more likely to be assigned to the community which is more densely connected to v, as the labels of that community appear in most of v's friends.

Local expansion methods are based on growing a community using a community fitness metric that measures the quality of the community. Whang et al. [Whang et al. 2013] used a personalized PageRank algorithm for finding cuts between communities, where a random walk in a network can start from seeds only. Since the vertices close by a seed are more likely to be visited, thereby receive higher ranks and join the same communities. Among the methods, Algorithm LFM [Lancichinetti et al. 2009] chooses random seeds and then expands the seeds until the value of fitness function based on the number of edges in the community is locally maximal. While the fitness metric used in the local expansion methods can capture the community density, it suffers from free rider [Wu et al. 2015] or separation effects.

The clique expansion method [Lee et al. 2010] identifies distinct cliques as initial seeds, and then expands the seeds greedily using a local fitness metric. In clique percolations [Palla et al. 2005], a community is defined as the maximal union of maximal cliques that can reach each other through a series of adjacent maximal cliques. However, some social networks, i.e., collaboration networks, are essentially a union of cliques can be considered as a single community. To tackle this issue, algorithm EAGLE [Shen et al. 2009] it is proposed to merge two communities with the maximum similarity into one, where the similarity between two communities is proportional to the number of edges between them. In algorithm COCD [Du et al. 2008], seeds are a set of maximal cliques and two maximal cliques are merged if their similarity is positive, where the similarity between two cliques is a proportional to the number of edges between non-overlapping vertices of those cliques. Even though cliques can guarantee a very strong connectivity among its members, it is considered as a very strict condition for real-world communities.

3.2 Community search

Community detection in real-world complex networks has been the focus of many scholars in computer science [Fortunato 2010; Xie et al. 2013] and several algorithms have been developed for identifying communities from complex networks that are surveyed in [Fortunato 2010; Xie et al. 2013]. Studies in this area have mainly considered the problem of identifying all non-overlapping communities [Fortunato 2010] and overlapping communities [Xie et al. 2013] from complex networks. However, in real-life applications, we are usually interested in identifying communities around a given set of vertices of a networks $Q \subseteq V$, called query vertices. To this end, the community search problem has been defined [Cui et al. 2014] and several algorithms for solving this problem have been proposed [Barbieri et al. 2015; Cui et al. 2014; Huang et al. 2014; Huang et al. 2015; Wu et al. 2015]. In this section, we review these state-of-

the-art techniques for identifying communities of a given set of query vertices.

Several papers have considered the problem of local community detection [Chen et al. 2009; Gleich and Seshadhri 2012; Yang et al. 2014], where the objective is to identify a set of communities that exist in the neighbourhood of the query Q. Such techniques fail to detect communities that span beyond the borders of neighbourhood induced subgraph, since the search space of such methods is limited to the neighbourhood of the query vertices.

Cui et al. [Cui et al. 2014] introduced two definitions for single vertex community search problem: Community Search with Maximality constraint (CSM) and Community Search with Threshold constraint (CST). In CSM, the objective is to find a connected subgraph that contains a given query vertex q and has the largest minimum degree, while in CST, the objective is to find a connected subgraph that contains the query vertex and its minimum degree is no less than a given threshold. In both CSM and CST, the input query is restricted to only one vertex, which is not a practical condition for many real-world applications. Barbieri et al. [Barbieri et al. 2015] proposed an extended model of CSM, which is capable of handling queries with more than one vertex, based on minimum degree. However, there is a drawback in employing minimum degree for measuring density of communities, since minimum degree fails to guarantee a strong edge-connectivity among vertices in the same community. In [Wu et al. 2015], Wu et al. studied the free rider effect in community detection as the problem of communities being merged during the process of community detection, due to use of inappropriate fitness metrics. In order to tackle the free rider effect, Wu et al. [Wu et al. 2015] proposed a weighting scheme for vertices based on the random walk model and associated a penalty for co-membership in communities, which is relative to the distance between vertices. In order to tackle both free rider effect and edge-connectivity problems, Huang et al. [Huang et al. 2015] proposed an algorithm that avoids the free rider effect by minimizing the diameter of a community and gurarantees a strong edge-connectivity by the notion of k-truss. In the method proposed by Huang et al. [Huang et al. 2015], the number of triangles formed by each edge is considered as a measure of density for communities. Shan et al. [Shan et al. 2015b] defined the community search problem as finding all subgraphs G' with respect to three parameters k, α and γ, such that G' is a γ-quasi-k-clique, and every edge in G' forms $\alpha - 2$ triangles with other vertices in G'. However, it is noted that such techniques are not able to find communities in networks that do not have any triangles, such as bipartite networks. Another issue with the studies mentioned above is that they are restricted to find only one single community associated with each set of query vertices, which is not a realistic assumption in real situations.

Another way to detect communities is to assign a weight w_{ij} to each pair of vertices (i, j) that represents how closely connected i and j are in the network [Girvan and Newman 2002], and then use these weights w_{ij} to determine if two vertices i and j are in the same community [Girvan and Newman 2002]. One possible definition of the weight is the number of edge-disjoint paths between vertices [Zhou et al. 2012]. It is known that the number of edge-disjoint paths between two vertices i and j in a graph is equal to the minimum number of edges that must be removed from the graph to

disconnect i and j from one another (calculated by max-flow algorithms) [Zhou et al. 2012]. However, there is no bound on the length of edge-disjoint paths in this method. Thus, very long paths can be counted, while vertices on those paths are not relevant to the endpoints. Another possible way to define weights between vertices is to count the total number of paths that run between two vertices [Katz 1953] (all paths, not just those that are vertex- or edge-disjoint). However, because the length of paths between any two vertices can be very large, one typically weights paths of length ℓ by a factor α^ℓ ($0 < \alpha < 1$), so that long paths contribute exponentially less weight than those that are short.

$$w_{ij} = \sum_{\ell=1}^{\infty} \alpha^\ell [A^\ell]_{ij} \tag{3.1}$$

For the sum to converge, we must choose α smaller than the reciprocal of the largest eigenvalue of the adjacency matrix A [Katz 1953]. Although this definition of the weights gives reasonable results for community structure in some cases, they fail to be effective in many real-world networks. The reason is that non-simple paths are taken into account, which means that in the presence of a loop, there will be an infinite number of paths between a pair of vertices. Moreover, the scaling factor reduces exponentially, which makes it hard to distinguish between paths that are actually short (for example, difference of factors for paths of length 1 and 3 is very significant).

Chapter 4

Cohesive Hierarchies of Communities in Complex Networks

4.1 Overview

In this chapter, we study the problem of hierarchical community detection based on the notion of cohesiveness among communities in a hierarchy, which dictates communities at lower levels of a cohesive hierarchy are more densely connected with each other. We define the notion of *cohesive hierarchy* that is a rooted tree of communities, where communities at the k-th level of the hierarchy are connected to each other through weak cuts with no more than k edges. For example in Fig. 4.1, communities at the second level (V_1, V_2 and $V_3 \cup ... \cup V_6$) are connected to each other by two edges, and communities at the third level (V_3 and $V_4 \cup V_5 \cup V_6$) are connected to each other by 3 edges. As we move towards the leaves of this hierarchy, the connectivity becomes stronger and communities become more densely connected.

We propose an efficient, yet scalable cut-based algorithm for detecting a cohesive hierarchy in a complex network. However, finding cuts in a network with hundreds of millions of edges is a painstaking task. Particularly at high levels of a hierarchy, when a network has not been broken into smaller subgraphs, finding cuts in the network is a challenge. Based upon this observation, we optimize the minimum cut detection [Stoer and Wagner 1994] algorithm by sparsifying the network in early iterations. Specifically, when finding communities at the k-th level of a cohesive hierarchy, we remove unnecessary edges that are not part of any edge-cut with size k. Therefore, when detecting communities at top levels of the hierarchy, the number of edges is significantly reduced, i.e. at most $\min\{m, (k+1)(n-1)\}$ at level k, where n and m are the numbers of edges and vertices, respectively.

One of the key ingredients of the proposed cohesive hierarchical community structure is the information centrality of communities, which is increased from communities at one level to communities at a lower level of the hierarchy. However, the information centrality can be calculated in time $O(nm)$ using an exhaustive brute-force approach, which is unrealistic for the scale of real-world networks such as Facebook and LiveJournal. Moreover, the effective diameter in real-world networks is usually small, as they exhibit a small-world characteristic. Therefore, we propose a random-

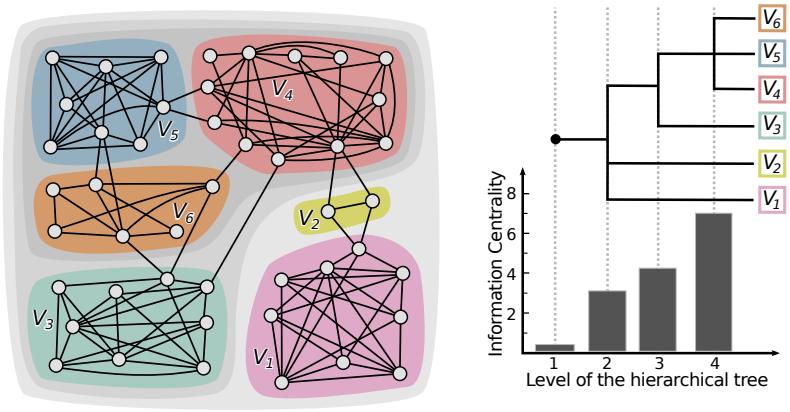


Figure 4.1: A hierarchical structure of communities in Amazon, where from the root to its leaves connections within communities become denser and the values of their information centrality increase.

ized approximation algorithm for calculating the information centrality with an error bound that is no larger than a fraction ϵ of the network diameter, with high probability (at least $1 - \frac{1}{n}$).

The contributions of this chapter are as follows,

- The problem of cohesive hierarchical community detection is formalized, where the granularity of a community is measured using information centrality, and the NP-hardness of this problem is proved.
- An efficient algorithm for the hierarchical community detection problem is developed and the scalability of this algorithm is enhanced by incorporating a fast sparsification for efficiently finding less granular levels of a hierarchy.
- A randomized approximation algorithm is proposed for calculating the information centrality of a network with a guaranteed approximation ratio.
- The effectiveness and efficiency of the proposed algorithms are evaluated by quantitatively analysing their performance using five large real-world datasets.

The rest of this chapter is organized as follows. Section 4.2 introduces the problem definition. Section 4.3 presents two novel algorithms for the hierarchical community detection problem and Section 4.4 presents an approximation algorithm for information centrality. Section 4.5 discusses the experimental results, which compare the performance of our proposed algorithms against the benchmark algorithms. Section 4.6 summarises the chapter.

4.2 Problem definition

Traditionally, communities are perceived as subsets of vertices of a graph G among which the number of edges (density of connections) is large [Fortunato 2010]. Follow-

ing this perception, it is possible to find hierarchical communities by recursively increasing the density threshold, and consequently finding denser communities. In this chapter, we take the relationships among communities into account for finding hierarchical communities of a network, while previous studies only focused on the relationships among vertices. Specifically, we here represent the hierarchy of communities as a rooted tree of subsets of vertices in G. Given two partitions $P = \{V_1, \cdots, V_{|P|}\}$ and $P' = \{V'_1, \cdots, V'_{|P'|}\}$ of V , we say that P has a higher *hierarchical order* than P', denoted by $P \succ P'$, if for every set $V'_i \in P'$ there is a superset $V_j \in P$ that includes V'_i, i.e. $V'_i \subset V_j$. However, density of connections among communities is not uniform across different levels of a cohesive hierarchy. In fact, density of connections among communities becomes larger at lower levels of a cohesive hierarchy. Therefore, we say that P has a higher *hierarchical order at degree k* compared to P', denoted by $P \succ_k P'$, if for every set $V'_i \in P'$ there is a set $V_j \in P$ such that $V'_i \subset V_j$ and for every set $V'_j \in P'$, $E[V'_i, V'_j] \leq k$. A sequence of partitions of V, e.g. $\mathcal{P} = \langle P_1, \cdots, P_{|\mathcal{P}|} \rangle$ is said to be a *hierarchy*, if $P_{i-1} \succ P_i$ $(1 < i \leq |\mathcal{P}|)$. We refer to $\mathcal{P}' = \langle P'_1, \cdots, P'_{|\mathcal{P}'|} \rangle$ as a *cohesive hierarchy* if $P'_{i-1} \succ_i P'_i$ $(1 < i \leq |\mathcal{P}'|)$. We note that every cohesive hierarchy is a hierarchy, but a hierarchy is not necessarily cohesive. Given a hierarchy $\mathcal{P}$, we refer to $P_{|\mathcal{P}|}$ the lowest level of hierarchy and we refer to P_1 as the root of the hierarchy. For every partition $P_i \in \mathcal{P}$, we say that $P_j \in \mathcal{P}$ is at a lower level if $j > i$.

Example 2. *Let us revisit the network illustrated in Fig. 4.1. Assume that V is the set of vertices of the network and V_i is the set of vertices in a subgraph G_i $(1 \leq i \leq 6)$. We show that $\mathcal{P} = \langle P_1, P_2, P_3, P_4 \rangle$ is a cohesive hierarchy, where $P_1 = \{V\}$, $P_2 = \{V_1, V_2, V_3 \cup V_4 \cup V_5 \cup V_6\}$, $P_3 = \{V_1, V_2, V_3, V_4 \cup V_5 \cup V_6\}$, and $P_4 = \{V_1, V_2, V_3, V_4, V_5, V_6\}$. It can be seen that $P_1 \succ_2 P_2$, since vertices in V_1, V_2 and $V_3 \cup V_4 \cup V_5 \cup V_6$ are connected to each other by at most 2 edges. Furthermore, $P_2 \succ_3 P_3$, since V_3 is connected to $V_4 \cup V_5 \cup V_6$ by at most 3 edges. Similarly, $P_3 \succ_4 P_4$, as V_4, V_5 and V_6 are connected to their siblings by at most 4 edges, which means that $P_4 = \{V_1, V_2, V_3, V_4, V_5, V_6\}$. As a result, $\mathcal{P} = \langle P_1, P_2, P_3, P_4 \rangle$ is a cohesive hierarchy. It is also noted that a cohesive hierarchy such as $\mathcal{P}$ results in a decomposition tree that is illustrated in Fig. 4.2, where siblings at level k of the decomposition tree are connected to each other by at most k edges.*

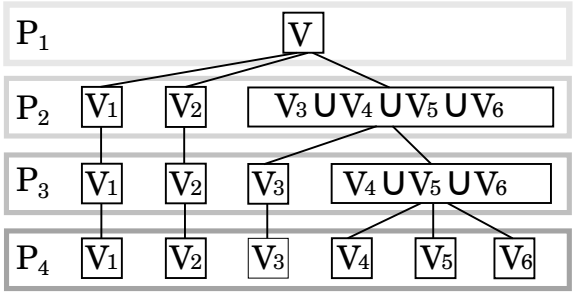


Figure 4.2: The cohesive hierarchy of the network illustrated in Fig. 4.1.

Since information centrality has been widely adopted to capture the closeness of vertices in a network [Fortunato et al. 2004; Rezvani et al. 2015], we formally define

the problem of hierarchical community detection, based on information centrality.

Definition 1 (HCDP). *Given a network G, the hierarchical community detection problem is to find a cohesive hierarchy $\mathcal{P}$ of maximal communities, where the sum of information centralities of the subgraphs induced by all partitions in $\mathcal{P}$ is maximized, i.e.*

$$\textit{maximize} \sum_{P_i \in \mathcal{P}} \sum_{V' \in P_i \setminus P_{i-1}} \mathcal{D}\left(G[V']\right). \tag{4.1}$$

The flat community detection is a special case of this hierarchical community detection problem, when $|\mathcal{P}| = 1$. In the following, we show that the hierarchical community detection problem is NP-hard by a reduction from the Maximum Clique problem [Bron and Kerbosch 1973] to its special case, i.e. the flat community detection problem. We first define the decision version of the flat community detection problem.

Definition 2 (CDP DECISION). *Given a network $G = (V, E)$, a rational number $\epsilon > 0$ and a positive integer k, the community detection decision problem is to determine whether there is a subset $V' \subset V$ of vertices with size k, whose information centrality is no less than ϵ, i.e. $\mathcal{D}\left(G[V']\right) \geq \epsilon$.*

The following lemma shows that the CDP DECISION problem is NP-hard, using a reduction from the MaximumClique problem.

Lemma 1. CDP DECISION *problem is NP-Complete.*

Proof. We first show that CDP DECISION is in NP. Given a certificate of CDP DECISION, which consists of a network G, and a set of vertices $V' \subseteq V$ with $|V'| = k$, we can use all-pairs shortest paths algorithm [Cormen et al. 2001] to determine if $\mathcal{D}(G[V']) \geq \epsilon$, in polynomial time. Thus, CDP DECISION is in NP.

We now show that CDP DECISION is NP-hard by a reduction from the MaximumClique problem. Note that a subset of vertices V' form a clique in G, if and only if the information centrality of the induced subgraph $G[V']$ is 1, i.e. $\mathcal{D}(G[V']) = 1$. Therefore, it is implied that the MaximumClique problem is a special case of CDP DECISION, where $\epsilon = 1$. Thus, given a network G and a positive integer k, one can decide the existence of a clique of the given size k by solving the CDP DECISION. □

4.3 Cohesive hierarchical community detection

In this section, we propose two efficient, yet scalable algorithms for the cohesive hierarchical community detection problem: (i) CHD is a basic cut-based algorithm which iteratively partitions the network into densely connected communities, and (ii) FACH is an optimized cut-based algorithm that relies on a sparsification technique to find sparse communities at high levels of a hierarchy.

4.3.1 The basic algorithm (CHD)

The CHD algorithm proceeds iteratively to identify a cohesive hierarchy $\mathcal{P}$ from a given network G. In each iteration, it finds one level of the hierarchy by creating a partition of each subset of vertices at its parent level. In iteration k, the algorithm finds a partition P_k by decomposing subsets of vertices in P_{k-1} into several subsets, where k is 1 initially, and incremented in each iteration.

Let $\mathcal{P}$ be a detected cohesive hierarchy of communities in network G, which is initialized by $\emptyset$ and let P_k be the k-th level of hierarchy $\mathcal{P}$. We assume that P_0 consists of all vertices as a community. Let k be the size of cuts detected by CHD in iteration k, which is initialized to 1 in the first iteration. In iteration k of the algorithm, for every set of vertices V' in P_{k-1}, which is currently a leaf in $\mathcal{P}$, an induced subgraph $G' = G[V']$ is constructed. Then a multi-cut of size no larger than k is detected in G' using the following procedure, called MAS-Decompose, and the result of removing this cut from G' is stored in another subgraph G''. The procedure MAS-Decompose [Chang et al. 2013] decomposes the subgraph G' into several connected components in G'', such that each connected component in G'' is connected to other vertices by at most k edges. The CHD algorithm then calculates the information centrality of G' and G'', and if the information centrality of G'' is no less than that of the initial subgraph G', it adds vertices in connected components of G'' to the k-th level of the hierarchy, i.e. P_k. Otherwise, the initial set of vertices V' is added to P_k. The CHD algorithm increments k and constructs levels of the hierarchy, until for all $V' \in P_{k-1}$, the algorithm cannot find a multi-cut of size k in G' that can increase the information centrality of G''. The detailed description of steps is given by Algorithm 1.

We now show the time complexity of Algorithm 1 as follows.

Theorem 1. *Given a network $G = (V, E)$, the algorithm* CHD *delivers a feasible solution for the hierarchical community detection problem in time $O(n^2m + n^3 \log(n))$.*

Proof. The CHD algorithm proceeds iteratively. Within each iteration, for every subgraph $G[V']$ ($V' \in P_{k-1}$), a MAS-Decompose is applied in time $O(nm + n^2 \log(n))$, and the calculation of information centrality takes $O(nm)$ time.

Therefore, the time complexity of MAS-Decompose is dominant at each iteration of the CHD algorithm. It is also noted that the hierarchy $\mathcal{P}$ has at most n leaves, as the number of subgraphs is no larger than the number of vertices in the network (due to the Pigeon-hole principle). Therefore, MAS-Decompose is called at most n times in each iteration of the CHD algorithm. However, in each iteration of the CHD algorithm, MAS-Decompose is run on partitions of the actual graph. Let n' and m' be the number of vertices and edges in the induced subgraph by each leaf V', i.e. $G[V']$, respectively. It is then implied that $\sum_{V' \in P_{k-1}} n'm' \leq nm$ as $\sum_{V' \in P_{k-1}} n' \leq n$ and $\sum_{V' \in P_{k-1}} m' \leq m$. This means that the overall time complexity of each iteration is no larger than $O(nm + n^2 \log(n))$. Since the edge-connectivity of a network is at most $n - 1$, the time complexity of the CHD algorithm is $O(n^2m + n^3 \log(n))$. □

Algorithm 1 CHD(G)

Input: $G = (V, E)$
Output: A cohesive hierarchy $\mathcal{P}$

/* Initialize the cohesive hierarchy */
$\mathcal{P} \leftarrow \emptyset$;
$k \leftarrow 0$;
/* P_k is the value of k-th level of the hierarchy */
$P_k \leftarrow \{V\}$;
$P_{k-1} \leftarrow \emptyset$;
while $(\mathcal{D}(P_k) - \mathcal{D}(P_{k-1}) > 0)$ **do**
    $k \leftarrow k + 1$;
    $P_k \leftarrow \emptyset$;
    for each $V' \in P_{k-1}$ **do**
        $G' \leftarrow G[V']$;
        /* Let G'' be a graph created by removing the edges
            in the multi-cut of size k found by running MAS on G' */
        $G'' \leftarrow$ MAS-Decompose(G', k);
        $CC \leftarrow$ the set of connected components of G'';
        if $\mathcal{D}(G') \leq \sum_{U \in CC} \mathcal{D}(G[U])$ **then**
            Add connected components of G'' to P_k;
        else
            $P_k \leftarrow P_k \cup \{V'\}$;
    $\mathcal{P} \leftarrow \{P_k\}$;
return $\mathcal{P}$;

4.3.2 The FACH algorithm

We here propose an algorithm, called FACH, that can reduce the number of edges from m in iteration k to $\min\{m, (k+1)(n-1)\}$.

In the following, we describe the construction of a sparse network $G_i = (V, E_i)$ $(1 \leq i \leq m)$ from $G = (V, E)$ which is noticeably smaller than G, while it can be used to find minimum cuts of G.

Lemma 2. *[Nagamochi and Ibaraki 1992] Given a network $G = (V, E)$ and an integer $k > 0$, let $F_1 = (V, E_1)$ be a spanning forest in G and $F_i = (V, E_i)$ be a spanning forest in $G \setminus E_1 \cup E_2 \cdots E_{i-1}$ $(1 < i \leq k-1)$. For any two vertices $s \in V$ and $t \in V$, if they are k-edge-connected in G, then they must be k-edge-connected in $G_k = F_1 \cup \cdots \cup F_k$.*

Lemma 2 states that in the second iteration, where $k = 2$, we can find a cut with size k in a network with at most $3(n-1)$ edges, instead of the entire network with m edges. Similarly, in iteration k of the algorithm, a cut of size k can be found in a network with $\min((k+1)(n-1), m)$ edges.

Motivated by Lemma 2, we propose the FACH algorithm that runs significantly faster than the CHD algorithm by starting from a sparse network and gradually increasing the network size as a factor of k. Specifically, in iteration k of this algorithm, the MAS-Decompose is run on a subgraph of the initial network with at most $\min\{(k+1)(n-1), m\}$ edges. In other words, the network size in the first iteration is no larger than $2(n-1)$. In the second iteration, the network size is no larger than $3(n-1)$ and in iteration k, the network size is no larger than $(k+1)(n-1)$. Let G_k be the network in iteration k, on which the MAS-Decompose is run. The FACH algorithm starts with $G_0 = MSF(G)$, where $MSF(G)$ is the minimum spanning forest of the network G. The algorithm then constructs G_k by adding the minimum spanning forest of the residual network $G \setminus G_{k-1}$ to G_{k-1}.

Let $\mathcal{P}$ be a cohesive hierarchy, which is initially empty. The algorithm iteratively increments the value of k and finds P_k, the partition representing the k-th level of the cohesive hierarchy $\mathcal{P}$, until the information centrality of P_k cannot be increased. Let $P_0 = \{V\}$. In iteration k, the FACH algorithm finds a minimum spanning forest F_k and form $G \setminus G_{k-1}$, where G_{k-1} is the union of spanning forests found in iterations 1 to $k-1$. For each community $V' \in P_{k-1}$, the FACH algorithm creates an induced subgraph $G_k[V']$ and finds a multi-cut in it using the MAS-Decompose procedure. If the removal of the multi-cut can increase the information centrality, the FACH algorithm applies the cut; otherwise, it proceeds to the next community in P_{k-1}. The detailed steps of the FACH algorithm is given in Algorithm 2.

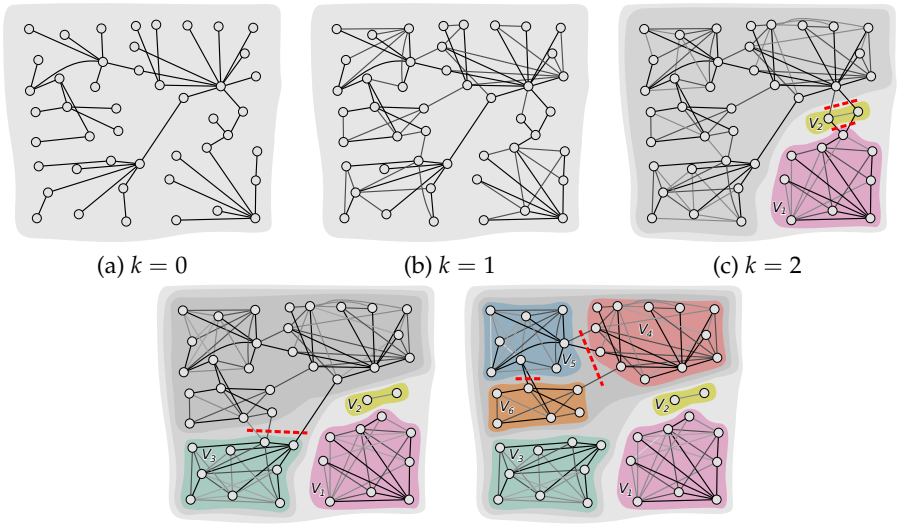
Example 3. *Fig. 4.3 illustrates a running example of the* FACH *algorithm, in which a network is sparsified in early iterations for finding cohesive communities. In Fig. 4.3b, it can be seen that in the first iteration ($k = 1$), the edge-cuts are found in a network that has only $81 = 43 + 38$ edges, which is sparser than the original network that has 124 edges. The number of edges is increased to $105 = 77 + 28$ and $113 = 102 + 11$ in the second and third iterations, respectively.*

We now provide the time complexity of the FACH algorithm.

Algorithm 2 FACH(G)

Input: $G = (V, E)$
Output: A cohesive hierarchy $\mathcal{P}$

```
1:  /* Initialize the cohesive hierarchy */
2:  P ← ∅;
3:  k ← 0;
4:  /* Find a Minimum Spanning Forest of G */
5:  G_k ← MSF(G);
6:  /* Let P_k be the k-th level of the hierarchy */
7:  P_k ← {V};
8:  P_{k-1} ← ∅;
9:  while (D(P_k) − D(P_{k-1}) > 0) do
10:     k ← k + 1;
11:     /* Find a Minimum Spanning Forest F_k of G \ G_{k-1} */
12:     F_k ← MSF(G \ G_{k-1});
13:     G_k ← F_k ∪ G_{k-1};
14:     P_k ← ∅;
15:     for each V' ∈ P_{k-1} do
16:         G' ← G_k[V'];
17:         G'' ← MAS-Decompose(G', k);
18:         CC ← the set of connected components of G'';
19:         if D(G') ≤ Σ_{U∈CC} D(G[U]) then
20:             Add connected components of G'' to P_k;
21:         else
22:             P_k ← P_k ∪ {V'};
23:     P ← {P_k};
    return P;
```



(a) $k = 0$ (b) $k = 1$ (c) $k = 2$

(d) $k = 3$ (e) $k = 4$

Figure 4.3: A running example of the FACH algorithm which illustrates the sparsification technique.

Theorem 2. *Given a network $G = (V, E)$, there is an algorithm for the hierarchical community detection problem, i.e. the* FACH *algorithm, which delivers a feasible solution in time $O(nm)$.*

Proof. The FACH algorithm consists of several iterations, where in iteration k, the union of spanning forests for levels 1 to $k-1$ (i.e. G_i), is constructed and for every induced subgraph $G_i[V']$ ($V' \in P_{k-1}$), a MAS-Decompose is applied, and the value of the information centrality is calculated in $O(nm)$ time.

The time complexity of the MAS-Decompose is $O(nm + n^2)$ and the time complexity of information centrality estimation is $O(nm)$. Since the network is connected and thus $m \geq n - 1$, the time complexity of the FACH algorithm is $O(nm)$. □

4.4 Approximating information centrality

To efficiently detect a cohesive hierarchy of communities in a complex network, one challenge is to calculate the value of information centrality. One straightforward way to calculate the information centrality is to discover all-pairs shortest paths, which is very time-consuming or even infeasible for real-world networks. Thus, we devise a simple, yet scalable algorithm for approximating the information centrality in polylogarithmic time.

We here build on the results obtained by Eppstein et al. [Eppstein and Wang 2001] and devise a randomized algorithm, Algorithm 3, which finds the information centrality of a network using a small number of randomly selected vertices from the network. The algorithm first selects a set S ($|S| = s$) of vertices from V, uniformly at random. It then runs the single-source shortest path algorithm (BFS), starting from each ran-

domly selected vertex in S. Finally, it estimates the value of information centrality of the network using the following equation,

$$\hat{\mathcal{D}}(G) = \frac{s \cdot (n-1)}{\sum_{u \in S} \sum_{v \in V} d_{uv}^G}. \tag{4.2}$$

Algorithm 3 describes an overview of the approximation algorithm for information centrality in a given network.

Algorithm 3 Estimated information centrality

Input: $G = (V, E)$, s
Output: $\hat{\mathcal{D}}(G)$

```
Let S be a set of s vertices selected uniformly at random;
for each vertex u ∈ S do
    Calculate shortest paths d_uv^G from u to all v ∈ V;
for each vertex v ∈ V do
    Ĉ(v) ← Σ_{u∈S} d_uv^G;
D̂(G) ← s · (n − 1) / Σ_{v∈V} Ĉ(v);
return D̂(G);
```

4.4.1 Theoretical analysis

We now analyse the approximation algorithm for information centrality. We first review the Hoeffding lemma [Hoeffding 1963] as follows.

Lemma 3 (Hoeffding [Hoeffding 1963])**.** *If $x_1, x_2, \cdots, x_n$ are independent variables, such that variable x_i ($1 \leq i \leq n$) is bounded by a_i and b_i, and $\mu = E[\sum x_i/n]$ is the expected mean, then for $\xi > 0$,*

$$\Pr\left\{\left|\frac{\sum_{i=1}^n x_i}{n} - \mu\right| \geq \xi\right\} \leq 2e^{-2n^2\xi^2/\sum_{i=1}^n (b_i - a_i)^2}. \tag{4.3}$$

Due to the small-world characteristic of complex networks [Watts and Strogatz 1998], the diameter in such networks is usually small. Therefore, we show that the error bound of Algorithm 3 for approximating the information centrality of networks is no larger than $\epsilon\Delta$ with a high probability, where Δ is the diameter of network and ϵ is a given approximation ratio.

Theorem 3. *Given a network $G = (V, E)$, and a set S of randomly selected vertices with size $\Theta(\log(n)/\epsilon^2)$, Algorithm 3 approximates the reciprocal of information centrality such that $|1/\hat{\mathcal{D}}(G) - 1/\mathcal{D}(G)| \leq \epsilon\Delta$, with a high probability of at least $1 - 1/n$.*

Proof. Recall that Algorithm 3 chooses $s = |S|$ random samples from vertices and the approximated value of information centrality is

$$\hat{\mathcal{D}}(G) = \frac{s \cdot (n-1)}{\sum_{u \in S} \sum_{v \in V} d_{uv}^G} = \frac{s \cdot n \cdot (n-1)}{n \cdot \sum_{u \in S} \sum_{v \in V} d_{uv}^G}. \tag{4.4}$$

It is noted that the expected reciprocal of estimation, i.e. $1/\hat{\mathcal{D}}(G)$, is the reciprocal of actual information centrality, i.e. $1/\mathcal{D}(G)$. Thus, in Hoeffding lemma, considering $x_i = \frac{n\sum_{v\in V} d_{iv}^G}{n(n-1)}$, we can safely assume that $\mu = 1/\mathcal{D}(G)$, $a_i = 0$ and $b_i = n\Delta/(n-1)$. Therefore, the probability that the difference between the reciprocal estimated information centrality $1/\hat{\mathcal{D}}(G)$ and the actual reciprocal information centrality $1/\mathcal{D}(G)$ being more than ξ is

$$\Pr\left\{\left|\frac{1}{\hat{\mathcal{D}}(G)} - \frac{1}{\mathcal{D}(G)}\right| \geq \xi\right\} \quad \leq \quad 2e^{-2s^2\xi^2/s(\frac{n\Delta}{n-1})^2}.$$

Considering the error being a small fraction of the diameter of G, i.e. $\xi = \epsilon\Delta \ll \Delta$, and using $s = \frac{\log n}{\epsilon^2}$ random samples, it can be seen that the probability of error is bounded above by $1/n$. Therefore, the approximation error of Algorithm 3 is smaller than $\epsilon\Delta$ with a high probability of at least $(n-1)/n$. □

The following theorem shows that the time complexity of the FACH algorithm with the proposed randomized approximation algorithm can be reduced to $O(n^2)$.

Theorem 4. *Given a network $G = (V, E)$, where $m = cn$ with constant c, the* FACH *algorithm delivers a feasible solution for the hierarchical community detection problem (HCDP), in time $O(n^2)$.*

Proof. The FACH algorithm proceeds iteratively, where in iteration k, the union of spanning forests for levels 1 to $k-1$, i.e. G_i, is constructed and for every induced subgraph $G_i[V']$ ($V' \in P_{k-1}$), a MAS-Decompose is applied, and the value of the information centrality is calculated, using Algorithm 3.

As mentioned above, the time complexity of the MAS-Decompose is $O(nm + n^2)$ and the time complexity of information centrality estimation is $O(m \log n + n \log^2 n)$. It can be seen that MAS-Decompose dominates the time-complexities of each iteration of the FACH algorithm. In iteration k, the network size is $k(n-1)$. Since the number of edges in real-world complex networks is usually a constant factor of the number of vertices, i.e. $m = cn$, for a constant c, the hierarchies $\mathcal{P}$ will have at most a constant number $k \leq c$ of levels. Therefore, the overall time complexity of the FACH algorithm is $O(n^2)$. □

4.5 Experimental results

In this section, we discuss the performance of the proposed algorithms, i.e. CHD and FACH, on several real datasets by comparing against several state-of-the-art algorithms. We first describe the experimental settings and then evaluate the performance of the proposed algorithms in detecting the hierarchical structure of communities. We finally investigate the performance of the CHD and FACH algorithms in finding different levels of a hierarchy.

Table 4.1: Real datasets with their characteristics.

Dataset	# vertices	# edges	# communities	Diameter
Facebook	4,039	88,234	308	8
Amazon	334,863	925,872	14,529	44
DBLP	317,080	1,049,866	7,556	21
LiveJournal	3,997,962	34,681,189	12,115	17
Orkut	3,072,441	117,185,083	9,120	9

4.5.1 Experimental settings

We introduce the benchmark algorithms, datasets and measures that were adopted for evaluating the proposed algorithms.

Benchmark algorithms. We compare the performance of the proposed algorithms, i.e. `CHD` and `FACH` (both of them using the randomized algorithm for approximating the information centralities of communities), with the following state-of-the-art algorithms for hierarchical community detection: `LinkCluster` [Ahn et al. 2010], `CNM` [Clauset et al. 2004], `InfoMap` [Rosvall and Bergstrom 2011], and `OSLOM` [Lancichinetti et al. 2011].

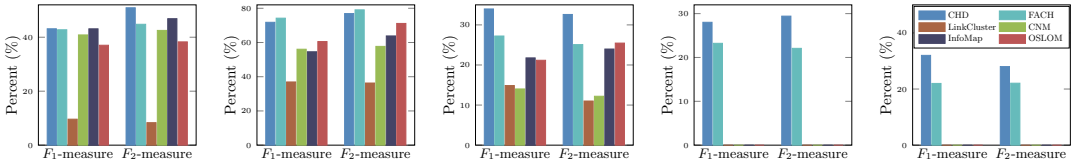
Datasets. We used five real datasets that are publicly available[1], and have been widely used in the literature [Xie et al. 2013]: (1) Facebook is a subgraph of the social network facebook, where communities are groups of members identified by surveyed users, (2) Amazon is a network in which vertices are products and there is an edge between two vertices i and j if product i is frequently co-purchased with product j. Products in each category are considered as ground-truth communities, (3) DBLP is a collaboration network of researchers, where communities are defined as journals and conferences, (4) LiveJournal is a friendship network of users in the LiveJournal website. Users can create groups, and these groups are considered as the ground-truth communities. (5) Orkut is the friendship network of Orkut members. Communities in this network are groups created by users, where users can join each group.

Evaluation measures. Measuring the quality of detected communities is challenging, as different metrics lead to different interpretations of communities. We employ F-measure that is widely-adopted in the literature [Gopalan and Blei 2013; Whang et al. 2013; Xie et al. 2013; Yang and Leskovec 2013; Zhao et al. 2005] for quantifying the accuracy of detected communities. Let $\mathcal{C}^*$ be the set of ground-truth communities and let C be a detected community. The F-measure of C compared to $C^* \in \mathcal{C}^*$ is defined as follows,

$$F_k(C) = \max_{C^* \in \mathcal{C}^*} \left\{ \frac{(k+1) \cdot p(C, C^*) \cdot r(C, C^*)}{k \cdot p(C, C^*) + r(C, C^*)} \right\}, \tag{4.5}$$

where k is the level in the hierarchy, and $p(C, C^*) = |C \cap C^*|/|C|$ and $r(C, C^*) =$

[1] http://snap.stanford.edu/data/index.html



(a) Facebook (b) Amazon (c) DBLP (d) LiveJournal (e) Orkut

Figure 4.4: F_1 and F_2-measures of the hierarchical community detection algorithms.

$|C \cap C^*|/|C^*|$ are the precision and recall, respectively. To calculate the accuracy of a flat community detection algorithm, one may calculate the average of F_1-measure and F_2-measure for all detected communities [Gopalan and Blei 2013; Whang et al. 2013; Xie et al. 2013; Yang and Leskovec 2013; Zhao et al. 2005]. However, the situation is different for hierarchical community detection algorithms, as communities detected at each level of a hierarchy can have different characteristics and be interpreted differently. One general rule in hierarchical community detection is that communities at the lower levels are smaller, more connected and more cohesive than the ones at the higher levels. Therefore, we suggest a weighting method in calculating the F-measure of communities at different levels of a hierarchy, which provides us with the ability to put more weight on communities at lower levels. Specifically, we incorporate a weight α_i, called the weight of level i, into F-measure of communities at level i of a hierarchy. Given a detected hierarchy $\mathcal{P} = \{P_1, \cdots, P_{|\mathcal{P}|}\}$, we define the F-measure of $\mathcal{P}$ as follows,

$$F_k(\mathcal{P}) = \sum_{1 \le i \le |\mathcal{P}|} \frac{1}{|\mathcal{P}|} \sum_{C \in P_i} \alpha_i \frac{F_k(C)}{|P_i|}, \tag{4.6}$$

$$\text{where } \alpha_i = \frac{i}{\sum_{1 \le j \le |\mathcal{P}|} j}.$$

Notice that the term α_i is used to emphasize on the accuracy of communities at the lower levels of the hierarchy. Although there are an infinite number of ways to define the weights α_i, we use a simple intuitive definition that makes both empirical and analytical sense.

All our experiments were run on a desktop computer with an Intel(R) Core(TM) i7-3770 CPU (3.40GHz) and 32GB of RAM.

4.5.2 Accuracy and efficiency

We study the accuracy and efficiency of our proposed algorithms `CHD` and `FACH` in hierarchical community detection by calculating the F-measures of communities found in each network and comparing them against the benchmark algorithms.

Fig. 4.4 shows the accuracy of different hierarchical community detection algorithms in terms of F_1-measure and F_2-measure. It can be seen in Fig. 4.4a that algorithms `CHD` and `FACH` outperform the benchmark algorithms by nearly 1% in F_1-measure for dataset Facebook. Fig. 4.4a also shows that algorithm `CHD` outperforms

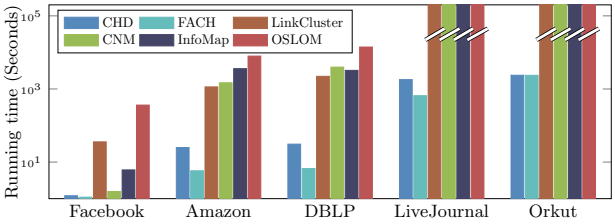


Figure 4.5: Running times of hierarchical community detection algorithms, where bars with parallel lines indicate that the corresponding algorithm did not terminate in 150 hours.

all other algorithms by at least 5% in F_2-measure using dataset Facebook. In Fig. 4.4b, it can be seen that the F_1 measures of algorithms CHD and FACH are at least 10% higher than the other algorithms in the benchmark using dataset Amazon. Similarly, F_2-measures for both CHD and FACH in this dataset are at least 5% higher than all other algorithms for dataset Amazon. Fig 4.4c plots the results for dataset DBLP, where it can be observed that both F_1-measure and F_2-measure of algorithm CHD is at least 20% higher than all other benchmark algorithms. However, the accuracy of algorithm FACH is slightly less than that of algorithm CHD for dataset DBLP. Fig. 4.4d presents the results for dataset LiveJournal and Fig. 4.4e presents the results for dataset Orkut, where due to the large size of the datasets only algorithms CHD and FACH terminated. It is noted that we waited for 150 hours for all algorithms to terminate, but only CHD and FACH found the results. Yet, the accuracy of the results is reasonable (above 20% for both datasets LiveJournal and Orkut). It is noted in Fig. 4.4 that the performance of algorithm CHD is better than that of algorithm FACH in datasets LiveJournal and Orkut. The reason is that algorithm CHD spends more time in finding edge-cuts in the networks.

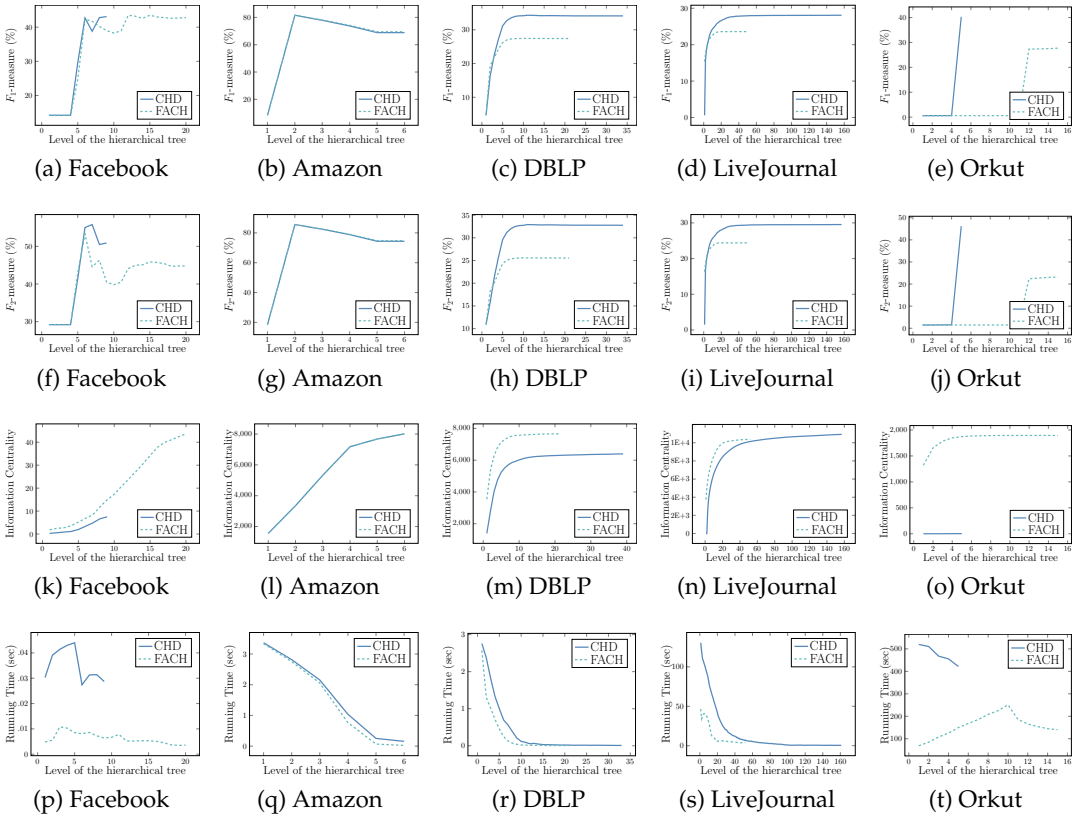
Fig. 4.5 depicts the running times of different hierarchical community detection algorithms. Although dataset Facebook is small, Fig. 4.5 shows that the running time of algorithms CHD and FACH is only a fraction of all other benchmark algorithms. In Fig. 4.5, for dataset Amazon, the running time of algorithm CHD is at most 1/50 of that of all benchmark algorithms, and the running time of algorithm FACH is only 80% of the running time of algorithm CHD. Similarly, for dataset DBLP, the running time of algorithm CHD is at most 1/75 of the running time of benchmark algorithms, and the running time of algorithm FACH is only 60% of the running time of algorithm CHD. However, for datasets LiveJournal and Orkut, only algorithms CHD and FACH terminated within 150 hours, and their running time was less than 30 minutes for such a large datasets. Fig. 4.5 illustrates that the running time of algorithm FACH is less than that of algorithm CHD, because of the sparsification technique. However, for dataset Orkut, both algorithms CHD and FACH have similar running times, because these algorithms discovers more levels of the hierarchy.

4.5.3 Hierarchies: level by level

Fig. 4.6 illustrates the values of F_1-measure, F_2-measure, information centrality and running times for the communities at each level of hierarchies detected by algorithms CHD and FACH. Fig. 4.6a shows that for dataset facebook, the value of F_1-measure increases as we move towards the lower levels of a hierarchy using algorithms CHD and FACH, with minor fluctuations. However, it can be seen in Fig. 4.6f that the value of $F2$-measure has a sudden drop at level 5 of the hierarchy, which indicates that the communities beyond level 5 of the hierarchy are too fine. In Fig. 4.6b, the values of F_1-measure of communities detected by algorithm CHD for dataset Amazon are increasing, as we move towards lower levels of hierarchy of communities. While there are some fluctuations in F_1-measure for dataset Amazon, the values are stable at the last two levels of hierarchy, i.e. levels 5 and 6. It is noted that the number of levels of the hierarchy for dataset Amazon is 6. Similarly, Fig. 4.6c shows that for dataset DBLP, the values of F_1-measure and F_2-measure have an increasing trend as we move towards the bottom of a hierarchy.

Fig. 4.6k-4.6o present the values of information centrality for communities at different levels of a hierarchy. Fig. 4.6k shows that for dataset facebook, the value of information centrality by algorithm CHD has a sudden increase and it stabilizes after that. However, in the same figure, the value of information centrality has a slower increase using algorithm FACH. It is noted in Fig. 7.7a that as the algorithm iterates, it finds more levels of the hierarchy and increases the values of the information centrality of communities, for dataset Amazon. Similarly for dataset DBLP, Fig. 7.7b shows that the information centrality of the detected communities is strictly increasing. In Fig. 7.7d, for dataset LiveJournal, the value of information centrality is increasing, as we move downwards in the hierarchy.

Fig. 4.6p-4.6t plot the amounts of time that algorithms CHD and FACH need for detecting each level of a hierarchy in different datasets. Fig. 4.6p shows that the running time of algorithm CHD is mainly dominated by its early iterations, where k is small. However, the running time of the algorithm FACH is almost stable with minor fluctuations across different levels of the hierarchy, which is due to the fast sparsification in algorithm FACH. It can be seen in Fig. 4.6q that for dataset Amazon, the running time of algorithm CHD drops significantly, as the algorithm moves to lower levels of the hierarchy. The reason behind is that algorithm CHD breaks the network into smaller and smaller subgraphs at each level and therefore, the time spent for detecting lower levels of a hierarchy is much less than that of the higher one. Fig. 4.6r shows the running times of algorithm CHD for different levels of the hierarchy in dataset DBLP, from which it can be seen that the running time significantly drops. Similarly, Fig. 4.6s shows that for dataset LiveJournal, the running time drops, as we move downwards the hierarchy of communities. However algorithm FACH spends significantly less amount of time in both early and late iterations, resulting in a faster outcome. In Fig. 4.6t, while the running time of algorithm CHD is dominated by its early iterations, algorithm FACH performs very fast in those early iterations. The conclusion from Fig. 4.6p-4.6t is that algorithm CHD is promising in hierarchical community detection,



(a) Facebook (b) Amazon (c) DBLP (d) LiveJournal (e) Orkut

(f) Facebook (g) Amazon (h) DBLP (i) LiveJournal (j) Orkut

(k) Facebook (l) Amazon (m) DBLP (n) LiveJournal (o) Orkut

(p) Facebook (q) Amazon (r) DBLP (s) LiveJournal (t) Orkut

Figure 4.6: Running times for each level of hierarchies detected by algorithms `CHD` and `FACH`

but algorithm `FACH` is much more efficient, particularly in early iterations.

4.6 Summary

We studied the hierarchical community detection problem in this chapter. We formally defined the problem of hierarchical community detection, as finding a rooted tree of communities where each community is a subset of its parent in the tree, and the information centrality of communities is no less than that of their parent in the hierarchical tree. We showed that the problem of finding hierarchical communities is NP-hard and devised an efficient and scalable heuristic algorithms for this problem. We further incorporated a fast sparsification method to reduce the network size for finding global cuts. We also proposed a fast randomized algorithm to estimate the value of information centrality in large-scale networks. We finally validate the effectiveness of our proposed algorithms using extensive experiments over five large-scale real-world datasets.

Chapter 5

Structural Holes Spanners in Complex Networks

5.1 Overview

One of the implications of structural hole spanners is that they bridge different communities and the shortest paths between those communities go through them. Therefore, their removal will increase the lengths of shortest paths between vertices of the network. For example, vertex v_1 in Fig. 5.1 plays a key role in the shortest paths between the vertices in different communities and its removal can significantly change the length between the other vertices, while the impact of the removal of other vertices on shortest paths is not as significant as vertex v_1. In this chapter, we propose a model based on the distance between all pairs of vertices of a network [Beauchamp 1965] for modelling structural hole spanners. We consider the structural hole spanners as a set of vertices whose removal will result in the maximum increase on the distances between the resulting network, and we term *the top-k structural holes problem* as the problem of finding a set of k vertices whose removal will maximize the increase on the distances in the resulting graph. The significant difference between our work and the state-of-the-art is that our model relies on only the network topological structure, while most existing works that assume that either all communities are given or rely on community detection algorithms.

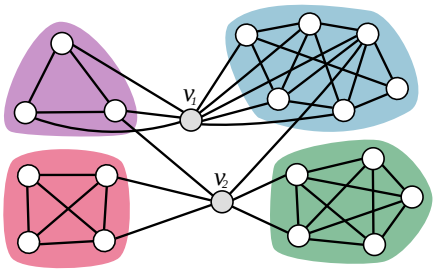


Figure 5.1: Illustration of structural hole spanners; each closed area represents a community, and vertices v_1, v_2 represent structural hole spanners that span multiple communities.

The main contributions of this chapter are as follows. We study the top-k structural hole spanner problem in a large-scale complex network. We first formulate the problem as an optimization problem and show its NP-hardness. We then devise two efficient, yet scalable algorithms, by exploring the bounded inverse of closeness centrality of vertices and articulation points, as well as the small-world phenomenon in complex networks. We finally evaluate the performance of the proposed algorithms by extensive experiments on real and synthetic datasets. Experimental results show that the proposed model can capture the structural spanners with high confidence and the structural hole spanners delivered by the proposed algorithms can connect more and larger communities in comparison with that by existing methods in real datasets. Moreover, using a synthetic datasets that contains structural hole spanners, we show that the proposed algorithms can accurately find the structural hole spanners and the precision of the solution is better than that of other methods. Furthermore, the proposed algorithms outperform the comparison heuristics by several orders of magnitude in terms of accuracy and running time. We also evaluate the impact of parameters on the proposed algorithms and demonstrate the role of small-world phenomenon in the performance of bounded inverse closeness centrality.

The rest of this chapter is organized as follows. Section 5.2 formally defines the problem of identifying top-k structural hole spanners. Section 5.3 shows the NP-hardness of the problem. Section 5.4 proposes algorithms for solving the problem. Section 5.5 evaluates the performance of the proposed algorithms, using real and synthetic datasets. Section 5.6 summarises the chapter.

5.2 Problem definition

Since structural hole spanners are fundamental in many applications, building a model that can accurately capture the structural properties of structural hole spanners is crucial. We here propose a model to identify the top-k structural hole spanners of a complex network.

Given a network $G = (V, E)$ and a positive integer k, the *top-k structural hole spanner problem* in G is to find a subset of vertices V_S ($V_S \subset V$) with $|V_S| = k$, such that the removal of the vertices in V_S from G will result in the maximum increase on the sum of the distances between all pairs of vertices in the induced subgraph $G \setminus V_S$, i.e., the problem objective is to

$$\max_{V_S \subset V,\ |V_S|=k} \{D(G \setminus V_S) - D(G)\}, \tag{5.1}$$

which is equivalent to

$$\max_{V_S \subset V,\ |V_S|=k} \{D(G \setminus V_S)\}. \tag{5.2}$$

Communities in a network are defined as the groups of vertices with dense connections internally and sparser connections externally [Girvan and Newman 2002]. Since

network communities are dense and vertices inside communities are well-connected, the distance between vertices within each community is small and the removal of community members does not change the distance between the other members of the community considerably. In contrast, structural hole spanners bridge different communities, thus their removal can increase the distance between vertices in those communities [Burt 1992]. The removal of top-k structural hole spanners in a network will result in the maximum number of communities disconnected in comparison with other k-vertex removals, thereby significantly increasing the sum of distances between all pairs of vertices in the network. Fig. 5.1 illustrates the impact of the removal of structural hole spanners on the distances between vertices. Specifically, the proposed model and the problem definition based on the model capture three important characteristics of structural hole spanners.

1. **Inter-community connections:** Given a vertex u that bridges multiple communities and another vertex v that has direct edge to vertices only in its community, vertex u is considered by the model to be a better structural hole spanner than vertex v, since the connections among vertices within the communities to which vertex v belongs are strong, and the absence of v only slightly increases the distance among other vertices in the network. In contrast, vertex u connects vertices from different communities, thus the absence of u can dramatically increase the distances between other vertices.

2. **Connections to important communities:** Given a vertex u that bridges large communities and a vertex v that bridges small communities, vertex u is considered to be a better structural hole spanner than vertex v, since the removal of u will disconnect more vertices in the network.

3. **Connections to many communities:** Given a vertex u that bridges many communities and a vertex v that bridges only a few, vertex u is considered to be a better structural hole spanner, since the removal of u can increase the distance between more communities (even if they are smaller).

Note that the proposed model relies only on the network structure and does not need to identify communities, thus it describes the properties of structural hole spanners in a unified way that is applicable to many real complex networks.

5.3 NP-hardness

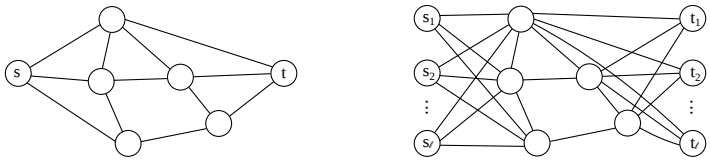
In this section we show that the top-k structural hole spanner problem is NP-hard by a reduction from an NP-hard problem - the *Most Vital Node Problem* (MVNP) [Bar-Noy et al. 1998], which is defined as follows.

Given an undirected network $G = (V \cup \{s, t\}, E)$, a pair of nodes s and t, and a positive integer k, assume that there is no edge between vertices s and t and the vertex connectivity $\kappa^G(s, t)$ between vertices s and t is no less than $k + 1$, the problem is to find a subset V_S of V with $|V_S| = k$ such that the length of the shortest path between s

and t in subgraph $G[(V \setminus V_S) \cup \{s,t\}]$ of G is maximized. The rest is to show that the problem is NP-hard by a reduction from the MVNP by the following theorem.

Theorem 5. *The top-k structural hole spanner problem in G is NP-hard.*

Proof. Given an instance of the MVNP in an undirected graph $G = (V \cup \{s,t\}, E)$ with $n = |V \cup \{s,t\}|$, a pair of vertices s and t in G, and a positive integer k, an instance of the top-k structural hole spanner problem in another undirected graph $G' = (V \cup S \cup T, E')$ can be constructed as follows. Let $l = 4n^6$. Sets of vertices S and T are obtained by duplicating vertices s and t with l copies, i.e., $S = \{s_1, s_2, \ldots, s_l\}$ and $T = \{t_1, t_2, \ldots, t_l\}$. For any two different vertices $u, v \in V$, an edge (u,v) is added to E' if an edge $(u,v) \in E$. For each vertex $v \in V$, l edges $(v,s_1), (v,s_2), \ldots,$ and (v,s_l) (or $(v,t_1), (v,t_2), \ldots,$ and (v,t_l)) are added to E' if edge (v,s) (or (v,t)) is contained in E. The construction of G' is illustrated in Fig. 5.2. Clearly, it can be verified that $\kappa^{G'}(s_i,t_j) = \kappa^G(s,t)$ for any vertex $s_i \in S$ and any vertex $t_j \in T$ and $d_{uv}^{G'} = d_{uv}^G$ for every pair of vertices u and v.



(a) G (b) G'

Figure 5.2: G' is constructed from G by replicating vertices s and t and their incident edges l times.

The MVNP in $G = (V \cup \{s,t\}, E)$ can be reduced to the structural hole spanner problem in G' as follows. We first show that an optimal solution to the problem in G' does not contain any vertex s_i or t_i, thus it is a feasible solution for the MVNP. We then prove that the optimal solution to the structural hole spanner problem is indeed the optimal solution to the MVNP.

Assume $V_S \subset V \cup S \cup T$ is an optimal solution to the top-k structural hole spanner problem in G', i.e., $D(G' \setminus V_S) = \max_{V'_S \subset V \cup S \cup T, |V'_S| = k}\{D(G' \setminus V'_S)\}$. We claim that V_S is an optimal solution to the MVNP in G. Following Lemma 11 in Appendix, V_S does not contain any vertices in S or T, therefore, V_S is a feasible solution to the MVNP. We now show that V_S is indeed the optimal solution to the MVNP in G too. To this end, let

$$x = \frac{D(G' \setminus V_S) - 4l(n_V - k) - 4l(l-1) - (n_V - k)(n_V - k - 1)}{2l^2}, \tag{5.3}$$

where $n_V = |V| = n - 2$. In the following we show that (i) $d_{st}^{G \setminus V_S} = \lfloor x \rfloor$, and (ii) $\lfloor x \rfloor = \max_{V'_S \subset V, |V'_S| = k}\{d_{st}^{G \setminus V'_S}\} = d_{st}^{G \setminus V_S}$. Then, set V_S is an optimal solution to the MVNP in G.

We start by showing Case (i): $d_{st}^{G\setminus V_S} = \lfloor x \rfloor$. We can see that $d_{st}^{G\setminus V_S} = d_{s_it_j}^{G'\setminus V_S}$ for any pair of vertices $s_i \in S$ and $t_j \in T$. To this end, we show that $d_{s_it_j}^{G'\setminus V_S} \leq \lfloor x \rfloor$ and $d_{s_it_j}^{G'\setminus V_S} \geq \lfloor x \rfloor$. Following Lemma 12 in Appendix, $D(G' \setminus V_S) \geq 4l(n_V - k) + 4l(l-1) + (n_V - k)(n_V - k - 1) + 2l^2 d_{s_it_j}^{G'\setminus V_S}$. Thus, $d_{s_it_j}^{G'\setminus V_S} \leq x$. Since the value of $d_{s_it_j}^{G'\setminus V_S}$ is a positive integer, $d_{s_it_j}^{G'\setminus V_S} \leq \lfloor x \rfloor$. Now, we show that $d_{s_it_j}^{G'\setminus V_S} \geq \lfloor x \rfloor$ by contradiction. Assume that $d_{s_it_j}^{G'\setminus V_S} < \lfloor x \rfloor$, then $d_{s_it_j}^{G'\setminus V_S} \leq \lfloor x \rfloor - 1$, following Lemma 12, we have

$$D(G' \setminus V_S) \leq 4l(n_V - k)\zeta + 4l(l-1) + (n_V - k)(n_V - k - 1)\zeta + 2l^2 d_{s_jt_j}^{G'\setminus V_S},$$

since we assumed $d_{s_it_j}^{G'\setminus V_S} \leq \lfloor x \rfloor - 1$,

$$\begin{aligned} D(G' \setminus V_S) &\leq 4l(n_V - k)\zeta + 4l(l-1) + (n_V - k)(n_V - k - 1)\zeta + 2l^2(\lfloor x \rfloor - 1) \\ &\leq 4l(n_V - k)\zeta + 4l(l-1) + (n_V - k)(n_V - k - 1)\zeta + 2l^2(x-1) \end{aligned}$$

then, we simply substitute x by its value from Eq. (5.3),

$$\begin{aligned} D(G' \setminus V_S) &\leq 4l(n_V - k)\zeta + 4l(l-1) + (n_V - k)(n_V - k - 1)\zeta - 2l^2 + \\ &\quad D(G' \setminus V_S) - 4l(n_V - k) - 4l(l-1) - (n_V - k)(n_V - k - 1) \\ &< D(G' \setminus V_S) + 4l(n_V - k)\zeta + (n_V - k)(n_V - k - 1)\zeta - 2l^2 \end{aligned}$$

since $n_V - k - 1 < n_V - k < n$ and $\zeta = n^3$, therefore,

$$D(G' \setminus V_S) < D(G' \setminus V_S) + 4ln^4 + n^5 - 2l^2$$

as $l = 4n^6$,

$$\begin{aligned} D(G' \setminus V_S) &< D(G' \setminus V_S) + 16n^{10} + n^5 - 32n^{12} \\ &< D(G' \setminus V_S), \end{aligned} \tag{5.4}$$

where inequality (5.4) is a contradiction. Therefore, $d_{s_jt_j}^{G'\setminus V_S} \geq \lfloor x \rfloor$. In conclusion, we have $d_{s_jt_j}^{G'\setminus V_S} = \lfloor x \rfloor$.

We then show Case (ii) $\lfloor x \rfloor = \max_{V'_S \subset V, |V'_S| = k}\{d_{st}^{G\setminus V'_S}\} = d_{st}^{G\setminus V_S}$. Denote by V_S^* the optimal solution to the MVNP in G, i.e., $d_{st}^{G\setminus V_S^*} = \max_{V'_S \subset V, |V'_S| = k}\{d_{st}^{G\setminus V'_S}\}$. We can see that $d_{st}^{G\setminus V_S} = d_{s_it_j}^{G'\setminus V_S}$ and $d_{st}^{G\setminus V_S^*} = d_{s_it_j}^{G'\setminus V_S^*}$, for any pair of vertices $s_i \in S$ and $t_j \in T$. Assume that $d_{s_it_j}^{G'\setminus V_S^*} > d_{s_it_j}^{G'\setminus V_S}$, then $d_{s_it_j}^{G'\setminus V_S^*} \geq d_{s_it_j}^{G'\setminus V_S} + 1 = \lfloor x \rfloor + 1 > x$. Following

Lemma 12, we have

$$\begin{aligned} D(G' \setminus V_S^*) &\geq 4l(n_V - k) + 4l(l-1) + (n_V - k)(n_V - k - 1) + 2l^2 d_{s_j t_j}^{G' \setminus V_S^*} \\ &> 4l(n_V - k) + 4l(l-1) + (n_V - k)(n_V - k - 1) + 2l^2 \cdot x \\ &= D(G' \setminus V_S), \end{aligned} \tag{5.5}$$

i.e., $D(G' \setminus V_S^*) > D(G' \setminus V_S)$, which contradicts the assumption that V_S is an optimal solution.

In conclusion, V_S is an optimal solution to the MVNP in G too. The top-k structural hole spanners problem is NP-hard. □

5.4 Algorithms for top-k structural hole spanner problem

In this section, we devise efficient algorithms for the top-k structural hole spanner problem. We first consider a greedy approach for the problem and improve its efficiency by exploring the bounded inverse closeness centrality using the small-world property of complex networks. Specifically, we start with a basic algorithm, using the inverse closeness centrality of vertices. We then develop a faster algorithm, by exploring the bounded inverse closeness centrality of vertices. We finally propose a fast, yet scalable algorithm by jointly considering both articulation points and the bounded inverse closeness centrality of vertices.

5.4.1 The basic algorithm

A structural hole spanner in a complex network usually spans multiple communities, thus the sum of distances between the spanner and the other vertices should not be larger than the sum of distances between an ordinary vertex and the other vertices in the network. The mean distance of the network after the removal of a vertex $v \in V$ thus is

$$d(G \setminus \{v\}) = \frac{n(n-1)d(G) - 2\sum_{u \in V} d_{uv}^G}{(n-1)(n-2)} + \frac{\sum_{u,w \in V} (d_{uw}^{G \setminus \{v\}} - d_{uw}^G)}{(n-1)(n-2)}, \tag{5.6}$$

where the value of $n(n-1)d(G)$ is the same for every vertex in G. If we only consider the first term in Eq. (5.6) as the dominant term, it can be implied that the shorter the distance between v and others, the more likely vertex v is to maximize the distances between vertices in the graph. We will use the inverse closeness centrality of vertices as an approximate measurement to find the top-k structural hole spanners in G. Specifically, the algorithm proceeds iteratively. The set of hole spanners V_S is empty initially, within each iteration, a new hole spanner $v \in V \setminus V_S$ is found and added to V_S, if its inverse closeness centrality $d(v)$ is the smallest among the vertices in $V \setminus V_S$. This procedure continues until the number of vertices in V_S becomes k. The detailed algorithm is described as follows.

We refer to algorithm 4 based on the Inverse Closeness Centrality of vertices as algorithm `ICC` for short. The dominant running time of algorithm 4 is to find a single source shortest path tree for each source vertex $v \in V$, which takes $O(m+n)$ time, using the BFS traversal on G. algorithm 4 thus takes $O(nm + n \log k) = O(mn)$ time, where the $\log k$ is the time of each priority operation in priority queue Q. Despite algorithm 4 is efficient, its time complexity is still quite high for a large-scale network that contains millions or billion of vertices. A challenging question then is whether this time complexity can be further significantly improved, e.g. a linear-time complexity, while the solution quality is not inversely compromised. In the following we answer this question affirmatively by devising two efficient algorithms for the problem.

5.4.2 Algorithm based on the bounded inverse closeness centrality

Most real-world complex networks follow two important facts: one is the sparsity. The number of neighbors of each vertex in a network is constant, which does not proportionally grow with the network size [Barabási and Albert 1999]; another follows the small world law. That is, the expected distance between any pair of vertices is a small constant, not proportional to the network size [Kleinberg 2000; Watts and Strogatz 1998]. Thus, instead of finding the single source shortest path tree for each vertex that includes all vertices in G, it suffices to find a *partial shortest path tree* for the vertex that reaches up to a given level of neighbors, where the neighbors of a vertex is its *level-1 neighbors*, the neighbors of its neighbors is its level-2 neighbors, and so on. We term the partial shortest path tree spanning up to level-l neighbors of v as the *l-bounded shortest tree* $T_l(v)$, and the *l-bounded inverse closeness centrality* of v thus is defined as

$$d_l(v) = \sum_{u \in T_l(v)} d_{uv}^G / (n-1). \tag{5.7}$$

We here adopt the similar metric as in algorithm 4, the only difference between them is to choose K vertices with top-K *largest* l-bounded inverse closeness centrality, rather than k vertices with top-k *smallest* inverse closeness centralities in algorithm 4, assuming that $K \geq k$. The rationale behind is that if a vertex (as a source) can reach a larger portion of vertices in a network within a small distance l, then its average distance to other vertices is shorter. To explore the diversity among vertices and to mitigate two neighbors chosen as the top-k structural hole spanners at the same time, the number of candidates K for the top-k hole spanners can be larger than k, e.g., $K = ck$ ($c \geq 1$). We then calculate the inverse closeness centralities of these K vertices in G, and choose the top-k smallest ones as the top-k structural hole spanners of the network. The proposed algorithm proceeds as follows.

It first identifies K vertices with top-K largest l-bounded inverse closeness centralities, starting at each vertex $v \in V$, using the BFS traversal on G. Assume that $d_l(v)$ is the sum of the lengths of shortest paths from each vertex within the l-neighborhood of vertex v. Let H be the set of top-K vertices with top-K largest bounded inverse closeness centralities. It then calculates the inverse closeness centrality $d(v)$ of v, for each

Algorithm 4 ICC

Input: $G = (V, E), k$
Output: The set V_S of top-k structural hole spanners
1: $V_S \leftarrow \emptyset$;
2: Let Q be a priority queue of top k hole spanners where the key of each element is its inverse closeness centrality;
3: **for** each vertex $v \in V$ **do**
4: calculate the inverse closeness centrality $d(v)$ of v;
5: **if** $|Q| < k$ **then**
6: add v to Q and the key of v is $d(v)$;
7: **else if** $d(v)$ is less than the largest key in Q **then**
8: remove the element with the largest key from Q;
9: add v to Q and the key of v is $d(v)$;
10: $V_S \leftarrow Q$.

Algorithm 5 BICC

Input: $G = (V, E), k, K, l$
Output: The set of top-k structural hope spanners V_S
1: Build a priority queue H with the bounded inverse closeness as the key of each element in H;
2: Build a priority queue V_S with the inverse closeness as the key of each element in V_S;
3: **for** each vertex $v \in V$ **do**
4: calculate the bounded inverse closeness centrality $d_l(v)$ of v, using BFS search;
5: **if** $|H| < K$ **then**
6: add v to H;
7: **else if** $d_l(v)$ is larger than the smallest key in H **then**
8: remove the smallest key element from H;
9: add v to H;
10: **for** each vertex $v \in H$ **do**
11: calculate the inverse closeness centrality $d(v)$ of v;
12: **if** $|V_S| < k$ **then**
13: add v to V_S;
14: **else if** $d(v)$ is less than the largest key in V_S **then**
15: remove the largest key element from V_S;
16: add v to V_S;
return V_S.

vertex $v \in H$, using the BFS technique on G. It finally identifies the k vertices from the K chosen vertices with top-k smallest inverse closeness centralities of the vertices. We refer to algorithm 5 based the Bounded Inverse Closeness Centrality of vertices as algorithm `BICC` for short. The rest is to analyze its time complexity by the following theorem.

Theorem 6. *Given an undirected connected graph $G = (V, E)$ with constant maximum degree and positive integers k and l, there is an algorithm, Algorithm 5, for the top-k structural hole spanners in G, which takes $O(m + n)$ time, where $n = |V|$ and $m = |E|$.*

Proof. Assume that G is represented by adjacency lists. Following algorithm 5, it first constructs a partial shortest path tree $T_l(v)$ rooted at v for each vertex $v \in V$ using the BFS technique, which takes $O(\sum_{i=1}^{l} d_{max}^i) = O(d_{max}^{l+1})$ time, where d_{max} is the maximum degree of vertices in G. It then identifies the top-K vertices with the largest bounded inverse closeness centrality, which takes $O(\log K)$ time for each vertex insertion into the priority queue H. Therefore, it takes $O(nd_{max}^{l+1}) = O(m + n)$ time for finding the set H as both d_{max} and l are small constants, in comparison with the network size n. Identifying set V_S takes $O(K(n + m) + K \log k)$ time, due to the BFS search in G for each candidate vertex in H, and the addition of the candidate vertex to set V_S, where V_S is maintained as a priority queue. Therefore, the time complexity of algorithm 5 is $O(K(n + m) + K \log K) = O(m + n)$ as $K = ck$ usually is constant. □

5.4.3 A fast and scalable algorithm

So far, we have provided an algorithm based on the inverse closeness centrality and devised an efficient algorithm by approximating the inverse closeness centrality of each vertex, using the l-bounded inverse closeness centrality concept. We now take the second term of Eq. (5.6) into account and devise another efficient algorithm which further speeds up the running time in practice, by exploring articulation points of G.

One of the properties of structural hole spanners in most complex networks is their tendency to connect multiple isolated communities. Such hole spanners are referred to *articulation points* in graph theory. Thus, a top-k structural hole spanner usually is an articulation point too. However, the number of articulation points in real-world complex networks is quite large, e.g., the number of articulation points in each network of Table 5.1 is at least 10% of the number of vertices in the network. How to identify top-k structural hole spanners from all articulation points in a large-scale network is a challenging issue. In the following, we shall devise a fast yet scalable algorithm for the top-k structural hole spanner problem, by exploring articulation points and using the bounded inverse closeness centrality of vertices.

Lemma 4. *[Plesník 1984] Let G be an unweighted connected graph $G = (V, E)$ with $n = |V|$ vertices, then the sum of lengths of all pairs shortest paths in G is no more than $n^3/3$.*

Given two vertices u and v that are not in the same connected component of G, we assume that there is a *virtual edge* in G between them with weight larger than the sum of lengths of all pairs shortest paths in G, and we assign this virtual edge with

a weight $w(u,v) = cn^3$ with $c \geq 1/3$. For the sake of convenience, we set $c = 1$ in the rest of discussion. Assume that v is an articulation point in G, we distinguish two cases as follows.

- Case one: if the removal of v results in two connected components CC_1 and CC_2. Let CC_i contain n_i vertices with $i = 1, 2$. The weighted sum of all virtual edges resulting from the removal of v is $n_1 \times (n - n_1)cn^3 + n_2 \times (n - n_2)cn^3 = cn^3(nn_1 + nn_2 - n_1^2 - n_2^2)$. Clearly, when $n_1 \approx n_2$, the weighted sum is maximized. This implies that an articulation point is likely to be a top-k hole spanner if its removal results in two large connected components.
- Case two: if the removal of v results in l connected components $CC_1, CC_2, \ldots, CC_l$ with $l > 2$. Let CC_i contain n_i vertices ($1 \leq i \leq l$). Let $n = \sum_{i=1}^{l} n_i$. Then, the weighted sum of virtual edges between vertices in CC_i and CC_j is $\sum_{i=1}^{l} \sum_{j=1}^{l} n_i n_j cn^3 = cn^3 \sum_{i=1}^{l} n_i(n - n_i)$, which is maximized when all components have roughly equal sizes.

Following the analysis of these two cases, it can be seen that an articulation point in G is likely to be one of the top-k structural hole spanner if its inverse closeness centrality is maximized, which approximately equals the weighted sum of virtual edges resulting from its removal, since the sum of all pairs shortest paths in each connected component is much less than the weight of each virtual edge. We now propose a fast, scalable algorithm by exploring the inverse closeness centralities of articulation points. Specifically, the algorithm consists of two stages. Let A be the set of articulation points in G. If $|A| < k$, the algorithm will proceed to the second stage after the first stage. Within the first stage, there are a number of iterations. An articulation point within each iteration will be chosen as a top-k hole spanner. In the second stage, algorithm 5 will be invoked to find the rest of top-k structural hole spanners. The detailed algorithm is described in algorithm 6.

We refer to algorithm 6 based on Articulation Points and Bounded Inverse Closeness Centrality of vertices as algorithm `AP_BICC` for short. We now show that articulation points and the approximate inverse closeness centrality of all articulation points can be done efficiently, using an extension of the Depth-First Search (DFS) traversal on G [Hopcroft and Tarjan 1973].

Let v be an articulation point of G, in the DFS tree construction starting from a vertex v, assume that $u_1, u_2, \ldots, u_p$ are the children of vertex v in the DFS tree. Let V_i be the set of vertices in the subtree T_i rooted at u_i and CC_i the connected component of G induced by the vertices in V_i with $1 \leq i \leq p$. Let CC_0 be the connected component containing the ancestors of v in the DFS tree. Following the DFS search property, all edges in G can be partitioned into two types of edges: "tree edges" and "non-tree edges", respectively. All non-tree edges are "back edges", which means that one endpoint of the edge is a descendant while another endpoint of the edge is a proper ancestor of v in the DFS tree. Clearly, there is no edges between any two connected components CC_i and CC_j with $i \neq j$ and $1 \leq i, j \leq p$, by the DFS traversal property as shown by Fig. 5.3. If there is a back edge between a vertex in V_i and a vertex in CC_0,

Algorithm 6 AP_BICC

Input: $G = (V, E), k, K, l$
Output: The set of top-k structural hole spanners V_S

build a priority queue V_S of top-K approximate inverse closeness candidates with the key of each element in V_S;
let A be the set of articulation points in G, obtained by invoking **Procedure 1**;
for each vertex $v \in A$ **do**
    find the approximate inverse closeness centrality $d'(v)$;
    if $|V_S| < k$ **then**
        add v to V_S;
    else if $d'(v)$ is larger than the smallest key in V_S **then**
        remove the smallest key element from V_S;
        add v to V_S;
if $|V_S| < k$ **then** /* the number of articulation points is less than k */
    $U \leftarrow V \setminus A$;
    $k' \leftarrow k - |V_S|$;
    $V'_S \leftarrow \emptyset$;
    build a priority queue Q of K elements with the key of each element in U being its l-bounded inverse closeness centrality;
    for each vertex $v \in Q$ **do**
        calculate the inverse closeness centrality $d(v)$ of v;
        if $|V'_S| + |V_S| < k$ **then**
            add v to V'_S;
        else if $d(v)$ is less than the largest key in V'_S **then**
            remove the largest key element from V'_S;
            add v to V_S;
$V_S \leftarrow V_S \cup V'_S$; **return** V_S.

Procedure 1 Articulation points and their approximate inverse closeness centrality calculation

for each vertex $u \in V$ **do**
    /* the number of children of u */
    $u.child \leftarrow 0$;
    /* each vertex has 3 colors white/grey/black */
    $u.color \leftarrow white$;
    $d'(u) \leftarrow 0$;
$time \leftarrow 0$;
for each vertex $u \in V$ **do**
    if $u.color == white$ **then**
        call **Modified-DFS(**G**,**u**)** /* **Procedure 2** */;

Procedure 2 Modified-DFS(G, u)

```
1: u.color ← black;
2: time ← time + 1;
3: /* the discovered time of vertex u */
4: u.discovered ← time;
5: /* the smallest discovered time among neighbors of u's
       descendants (through a back-edge) */
6: u.lowest ← time;
7: /* the number of vertices in CC_0 after removing u */
8: cc_0 ← n;
9: /* the number of descendants of u in DFS tree */
10: u.descendant ← 0;
11: for all (u, v) ∈ E do
12:     if u.color == white then
13:         u.color ← grey;
14:         v.π ← u /* u is the parent of v */;
15:         u.child ← u.child + 1;
16:         call Modified-DFS(G,v);
17:         u.descendant ← u.descendant + v.descendant;
18:         u.lowest ← min (u.lowest, v.lowest);
19:         if (v.lowest ≥ u.discovered) OR (u is root AND u.child > 1) then
20:             /* v will be disconnected without u */
21:             d'(u) ← d'(u) + (v.descendant × (n − v.descendant − 1));
22:             cc_0 ← cc_0 − v.descendant /* subtree of v is not part of CC_0 */;
23:     else if v ≠ u.π then
24:         u.lowest ← min (u.lowest, v.discovered);
25: d'(u) ← d'(u) + (cc_0 × (n − cc_0 − 1)).
```

then both CC_i and CC_0 are still the same connected component even if v is removed from G, $1 \leq i \leq p$. An illustration of this case is shown in Fig 5.3.

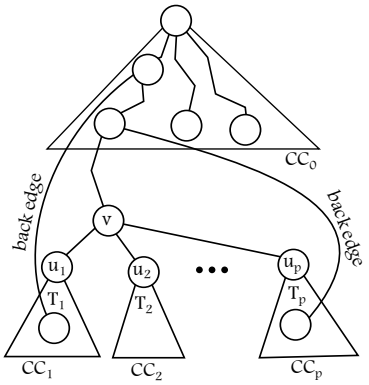


Figure 5.3: An illustration of exploring an articulation point v and its p children $u_1, u_2, \ldots, u_p$ during a DFS traversal on G.

Assume that there are p' CCs containing back edges among the p CCs derived from the p children of v. Then, the removal of v will result in $p - p'$ CCs. For the sake of convenience, we assume that these $p - p'$ CCs are $CC'_1, CC'_2, \ldots, CC'_{p-p'}$ with each having n'_i vertices. The approximate inverse closeness centrality of v then is

$$d'(v) \approx \sum_{i=1}^{p-p'} |CC'_i| \cdot (n - |CC'_i| - 1) \cdot n^3. \tag{5.8}$$

The linear-time procedure for detecting each articulation point and the calculation of its approximate inverse closeness centrality is then detailed as follows.

A vertex v is identified as an articulation point of G if a subtree rooted at one of its children does not contain any back edges to CC_0. The induced subgraph by the set of vertices in this subtree is a connected component after the removal of vertex v from G. The number of vertices contained in each such connected component is the number of descendants of that child in the DFS tree. To keep track of the number of descendants of each vertex when performing the DFS traversal on G and to identify those children of the vertex without any back edges, the approximate inverse closeness centrality of v (as an articulation point) can be easily calculated. The detailed implementation of this is given in Procedure 1 and Procedure 2, respectively.

Theorem 7. *Given a graph $G(V, E)$ with constant maximum degree and an integer $k > 0$, there is an efficient algorithm for the top-k structural hole spanners problem, algorithm 6, which takes time $O(n \log k + m) = O(m + n)$ as k is a constant.*

Proof. Following algorithm 6, the detection of all articulation points and the calculation of their approximate inverse closeness centralities takes $O(n + m)$ time by Procedure 1. For each vertex u, its adjacency list is traversed exactly once and the number of descendants and children are calculated in the post-traversal in DFS. The maintenance of the priority queue Q takes $O(|A| \log k) = O(n \log k)$ time, where A is the

Table 5.1: Six different real datasets.

Dataset	$\|V\|$	$\|E\|$	Articulation Points (%)	$\Delta(G)$	Diameter
GR-QC	5,242	28,980	15%	81	17
Epinions	75,879	508,837	14%	1,551	14
Twitter	92,180	188,971	14%	233	26
Email-euAll	265,214	420,045	2%	7,636	14
DBLP-2011	986,324	6,707,236	9%	979	12
LiveJournal	5,363,260	79,023,142	16%	2,469	14

set of all articulation points in G. The total amount of time for calculating the number of descendants of each vertex in the DFS tree is $O(n)$. Thus, the time complexity of algorithm 6 is $O(n+m)$. □

5.5 Performance evaluation

In this section we evaluate the performance of the proposed algorithms for solving the structural hole spanner problem. We start with presenting the experimental environment settings, we then investigate the effectiveness of the proposed algorithm of structural hole spanners, compared with other existing algorithms, using both real and synthetic datasets. We finally study the performance of the proposed algorithms and the impacts of parameters on the performance, using the datasets in Table 5.1 and in the end, we discuss the results.

5.5.1 Experimental environment setting

To evaluate the performance of the proposed algorithms, we adopt the real-world datasets, which are listed in Table 5.1, where `GR-QC` is the collaboration network from arXiv[1], covering collaborations between authors of chapters submitted to General Relativity and Quantum Cosmology category. `Epinions` is an online social network of a general consumer review site Epinions[2]. The `Twitter` dataset was obtained from [Lou and Tang 2013]. `Email-EuAll` is the anonymous email network of a large European research institution for an 18-month period [Leskovec and Krevl 2014]. The `DBLP-2011` dataset is the collaboration network obtained from the DBLP web site[3], and the `LiveJournal` dataset describes the social network of free on-line blogging community[4].

Recall that algorithms 4, 5, and 6 are denoted by `ICC`, `BICC` and `AP_BICC`, respectively. To evaluate their performance we will use the following state-of-the-art algorithms for benchmark purposes.

[1]http://arxiv.org/
[2]http://epinions.com/
[3]http://www.informatik.uni-trier.de/∼ley/db/
[4]http://livejournal.com/

- Algorithm `PathCount` [Goyal and Vega-Redondo 2007] is similar to the betweenness centrality and assigns each vertex a score that is the average number of shortest paths (between all pairs of vertices) on which the vertex lies, then selects the top-k vertices with the highest scores.
- Algorithm `2-Step` [Tang et al. 2012] assigns each vertex a score that is the number of pairs of its neighbors without edges between them, then selects the top-k highest scores.
- Algorithm `PageRank` [Page et al. 1999] assigns each vertex v a PageRank score $r(v)$ that is the visiting probability of v by a random surfer, $r(v) = 1/n$ initially. The algorithm then updates $r(v)$ with a new value $r(v) = (1-\alpha)/n + \alpha \sum_{(u,v)\in E} r(u)/\deg(u)$, where $\alpha = 0.85$ is the random jump parameter. It finally chooses the top-k vertices with the highest PageRank scores.
- Algorithm `MaxD` [Lou and Tang 2013] is to find a set of k vertices such that the minimum cut of communities will be reduced significantly, after removing these vertices, assuming that l communities are given. For any pair of communities, the algorithm selects $\lceil 2k/(l(l-1)) \rceil$ vertices as structural hole spanners using a greedy strategy. In each round, it chooses the vertex whose removal will result in a maximum decrease of the minimum cut.
- Algorithm `HIS` [Lou and Tang 2013] assigns each vertex v a score that simulates the likelihood of v as a structural hole spanner across the given subset of communities, assuming that l communities are given.

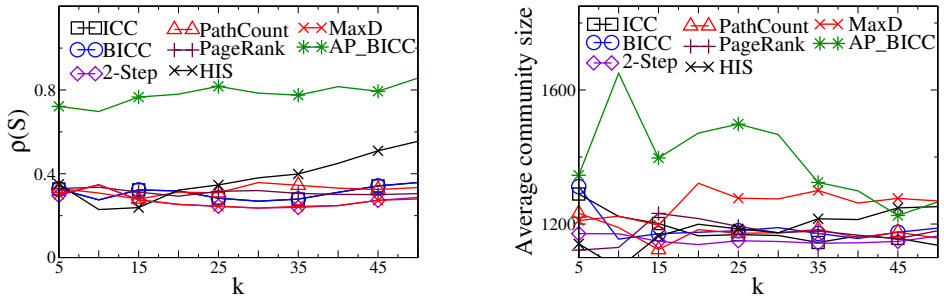
Notice that all our experiments were conducted based on a Linux desktop with GenuineIntel Core i7-3370 (3.40GHz) CPU and 8GB main memory.

5.5.2 Effectiveness of the proposed model

We first evaluate the effectiveness of the proposed model using the definition proposed by Burt [Burt 1992] such as: (1) the size of communities that each vertex spans, (2) the number of communities and (3) the number of neighbours of that vertices. Burt [Burt 1992] suggested that a good structural hole spanner is connected to many communities, but to be influential and efficient, the ratio of the number of its communities to the number of its neighbors should be large. This definition simply implies a metric for evaluating structural hole spanners in a setting where the communities in a network are given in advance. Given a graph $G = (V, E)$, suppose S is the set of structural hole spanners found by an algorithm, then, the quality of the solution S is

$$\rho(S) = \frac{\sum_{v\in S} (\#\text{ of communities that } v \text{ is connected to})/\deg(v)}{|S|}. \tag{5.9}$$

We evaluate the performance of different algorithms using this metric in order to find out the degree to which our model maps the real structural hole spanners. We use the `DBLP` dataset which has been used for the same purpose in [Lou and Tang



(a) `Community membership` (b) `Average community size`

Figure 5.4: Effectiveness of different algorithms on dataset `DBLP` using various quality metrics.

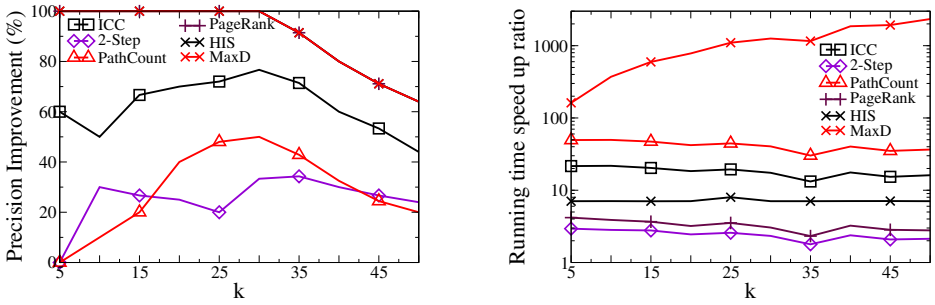
2013]. The communities in this network are publication venues, e.g, journal or conference; authors who published in certain journals or conferences form a community. We evaluate our algorithms against `MaxD` and `HIS` proposed by Lou *et al.* [Lou and Tang 2013].

Fig. 5.4 shows the performance of different algorithms, using the `DBLP` dataset. Fig. 5.4b illustrates that algorithm `AP_BICC` outperforms the other algorithms in the benchmark at least 50%, using the metric in Eq. (5.9). In other words, in the solution of algorithm `AP_BICC` the ratio of the number of communities to which vertices are connected to the number of their neighbours is more than 50% higher than that of other methods. This means that the structural hole spanners found by algorithm `AP_BICC` are efficiently connected to different communities and through a reasonable number of contacts. Similarly, it can be observed in Fig. 5.4a that algorithm `AP_BICC` significantly outperforms all the other algorithms in terms of the average community size by varying k. Fig. 5.4b verifies our claim in Section 5.2 that our model can identify the vertices connecting with larger communities. In a nutshell, algorithm `AP_BICC` can guarantee that the found structural hole spanners are connected to larger communities in this dataset, while the other algorithms find structural hole spanners that are in fewer communities, compared to the number of their adjacent vertices. It is worth to mention that the running time of algorithm `AP_BICC` is just a matter of milliseconds for dataset `DBLP`, while both algorithm `MaxD` and algorithm `HIS` take from a few seconds to a few minutes, which is several orders of magnitude of the proposed algorithms in running time.

5.5.3 Performance on synthetic datasets

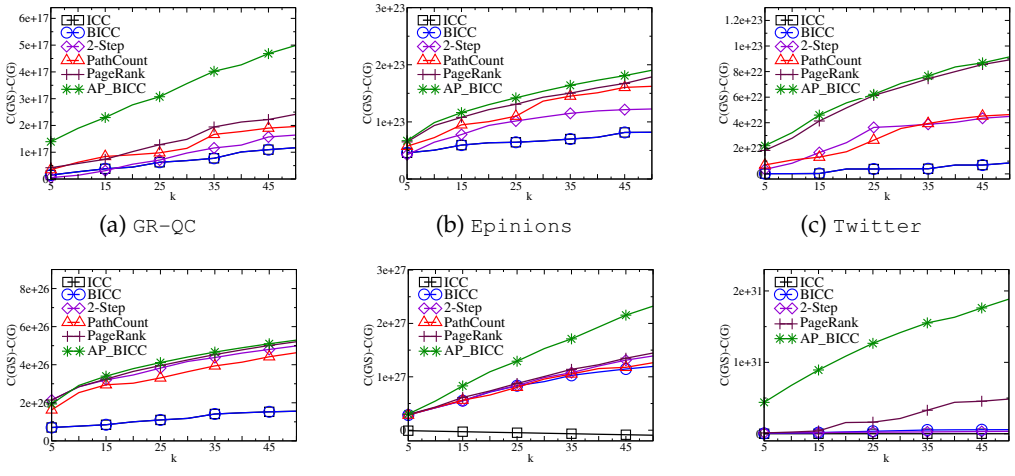
We then evaluate the quality of structural hole spanners found by different algorithms, using a synthetic dataset with known structural hole spanners. We generate a random graph of 2^{11} vertices, using the SSCA method[5], which generates cliques of random size, where the average size of cliques in this graph is 2^7, and randomly places inter-clique edges. We then place a ground-truth structural hole spanner s_i for every clique i. For every edge (u, v), such that u is in clique i and v is in clique j, we

[5]http://www.cse.psu.edu/~madduri/software/GTgraph/



(a) `Precision improvement` (b) `Speed up ratio`

Figure 5.5: Running time and precision improvement by algorithm `AP_BICC` using synthetic dataset.



(a) `GR-QC` (b) `Epinions` (c) `Twitter`

(d) `Email-EuAll` (e) `DBLP-2011` (f) `LiveJournal`

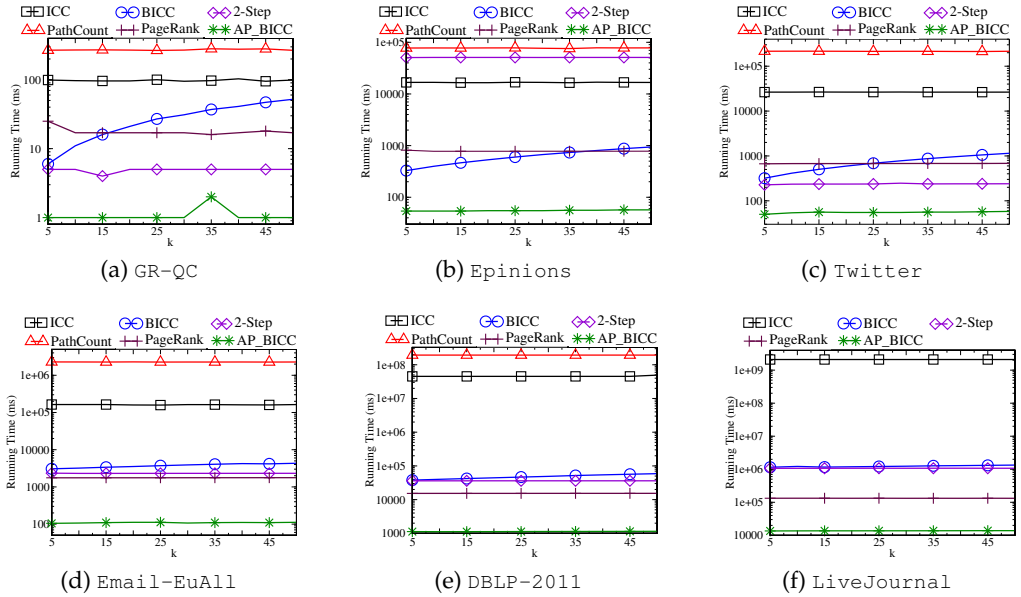
Figure 5.6: Performance of various algorithms, using various datasets.

replace it with two edges (u, s_i) and (s_i, v) in the graph (alternatively, since the graph is undirected, we place two other edges (s_j, u) and (v, s_j) in the network). We evaluate the structural hole spanners found by each method and calculate the precision based on the number of structural hole spanners that were found accurately.

Fig. 5.5 shows the performance improvement made by algorithm `AP_BICC` in empirical result, and demonstrates that the accuracy of the solution delivered by algorithm `AP_BICC` is at least 20% better off, compared with the others for $k > 5$. The reason behind is that algorithm `AP_BICC` finds more meaningful structural hole spanners that play a significant roles in the connectivity of communities (cliques) and vertices in the network.

5.5.4 Performance on real datasets

We also evaluate the performance of the mentioned algorithms `ICC`, `BICC`, and `AP_BICC`, and the structure-based benchmark algorithms `PathCount`, `PageRank`, and `2-Step` against various real datasets listed in Table 5.1. In this experiment, we avoid compar-



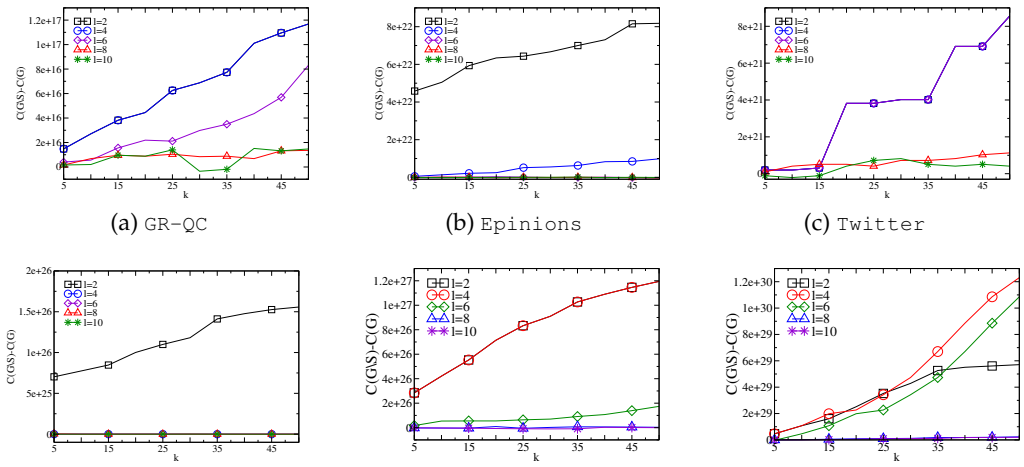
(a) GR-QC (b) Epinions (c) Twitter

(d) Email-EuAll (e) DBLP-2011 (f) LiveJournal

Figure 5.7: Running times of various algorithms, using various datasets.

ing community-based algorithms such as MaxD and HIS, since using different community detection algorithms can lead to different results and thus, can be unfair.

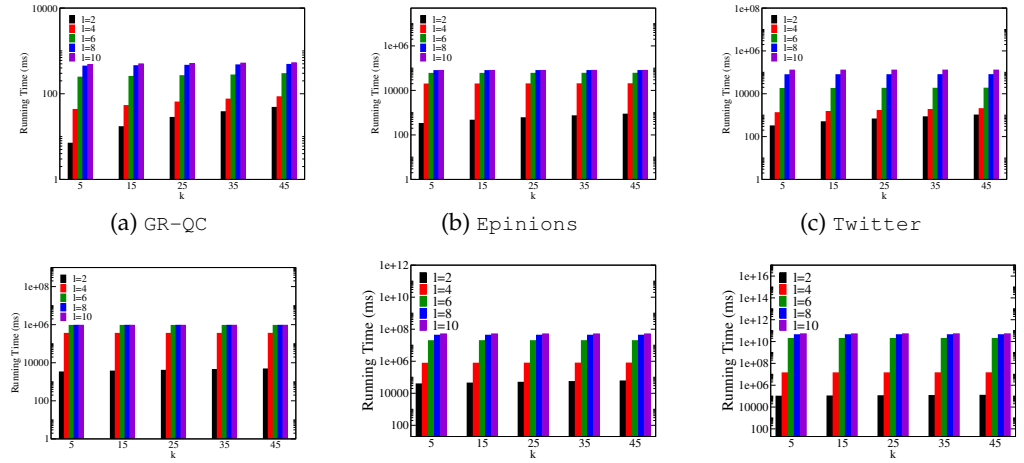
Fig. 5.6 and Fig. 5.7 show the performance of different algorithms in terms of the optimization objective in Eq. (5.1), i.e., $D(G \setminus S) - D(G)$ and the running time. Specifically, it can be seen from Fig. 5.6a that algorithm AP_BICC significantly outperforms all the other algorithms by at least 100% for dataset GR-QC, while its running time is only 6% of the lowest one among the three benchmark algorithms as shown in Fig. 5.7a. It can be observed from Fig. 5.6e to Fig. 5.6f that algorithm AP_BICC has the similar performance and running time for other datasets. That is, it outperforms all the other algorithms at least 50% while its running time is only around 7% and 65% of the fastest benchmark algorithm for datasets DBLP-2011 and LiveJournal, respectively. Fig. 5.6b and Fig. 5.6d further demonstrate that it is superior in comparison with benchmark algorithms such as PageRank and PathCount on both performance and running time for datasets Epinions and Email-EuAll when k is small, (e.g. $k < 20$). With the increase of k, the performance gap between algorithm AP_BICC and the other algorithms increases, too. Furthermore, it can also be seen from Fig. 5.7b and Fig. 5.7d that the running time of algorithm AP_BICC is less than 7% of the fastest benchmark algorithm PageRank and only 0.01% of the running times of algorithms ICC and PathCount for dataset DBLP-2011 as shown in Fig. 5.7e. However, it is interestingly noticed from Fig. 5.6f that algorithm AP_BICC significantly outperforms all other algorithms for dataset LiveJournal while its running time is less than 10% of algorithm PageRank. More interestingly, Fig. 5.7f depicts that the running time of algorithm BICC is no more than 0.1% of algorithm ICC. It can be seen from Fig. 5.6 that that algorithm AP_BICC significantly outperforms algorithm BICC, which means that incorporating articulation points and the bounded inverse closeness centrality is



(a) GR-QC (b) Epinions (c) Twitter

(d) Email-EuAll (e) DBLP-2011 (f) LiveJournal

Figure 5.8: Impact of parameter l on performance of algorithm BICC.



(a) GR-QC (b) Epinions (c) Twitter

(d) Email-EuAll (e) DBLP-2011 (f) LiveJournal

Figure 5.9: Impact of parameter l on running time of algorithm BICC.

considerably effective. Furthermore, since the number of articulation points in real datasets is considerably larger than the desired value of k, algorithm AP_BICC never reaches the second phase, making it to run significantly faster than all other methods.

5.5.5 Impact of parameters on the performance

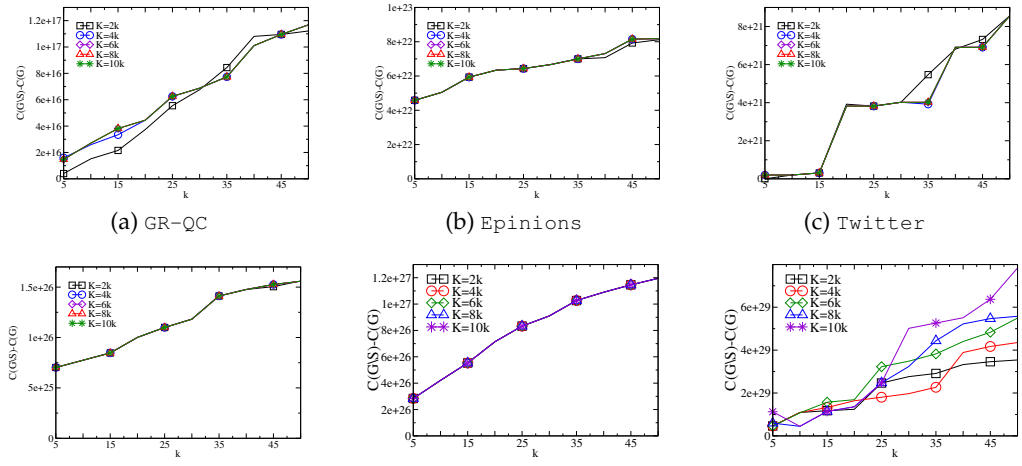
We finally evaluate the impact of parameters on the performance of the proposed algorithms AP_BICC and BICC. As the number of articulation points in each real dataset is far larger than k (please refer to Table 5.1 for the number of articulation points in each dataset), the second stage of algorithm AP_BICC, on the bounded inverse closeness centrality of vertices will not be invoked. Therefore, we only investigate the impact of parameters l and K on the performance of algorithm BICC.

Fig. 5.8 and Fig. 5.9 plot the performance curves of algorithm BICC by varying

the value of l, based on the datasets described in Table 5.1. Fig.5.8a shows that for dataset GR-QC, when l is small ($2 \leq l \leq 4$), the performance of algorithm BICC is at least 30% higher than that of larger values of l. It can be seen from Fig. 5.8b that when parameter l is large, algorithm BICC performs less than 20% of the case where l is small ($l = 2$) using dataset Epinions. Fig. 5.8c illustrates that for dataset Twitter, the performance of small values of parameter l is almost identical, while the performance drops when the value of parameter l is increased ($l > 6$). Fig. 5.8f implies that when l is small with $2 \leq l \leq 6$, algorithm BICC has the best performance for dataset LiveJournal. It also exhibits similar behaviours for dataset DBLP-2011 as shown in Fig. 5.8e. However, its performance degrades chromatically when $l = 6$, i.e., its performance decreases by at least 80% compared with its performance when $l = 2$. This implies that its performance dramatically drops with the growth of l. The reason behind is that algorithm BICC always chooses the top-K vertices with the largest l-bounded inverse closeness $d_l(v)$. By continuously increasing the value of l, the value of $d_l(v)$ will be closer to the value of its inverse closeness centrality $d(v)$. Since the top-K vertices with the "largest" l-bounded inverse closeness centrality (that is indeed almost the same as inverse closeness centrality for large l) will be chosen, the algorithm performance drops. Fig. 5.8 suggests that the value of l in practice should be small, otherwise the quality of the solution is not promising, this further verifies the small-world phenomenon that causes l-bounded closeness centrality to be very close to inverse closeness centrality, and the largest distance in the network is expected to be a constant.

Fig. 5.9 plots the running times of algorithm BICC by varying parameter l. It can be seen from Fig. 5.9a that for dataset GR=QC, the running time of algorithm BICC grows very fast by increasing the value of l. Specifically, when l is small ($l = 2$), the performance of the algorithm is less than 50% of the case where l is large ($l = 10$). Fig. 5.9b shows the running time of the algorithm using dataset Epinions, where it can be seen that the performance drops considerably when l is large. However, it is noticed from Fig. 5.9b that when l is large ($8 \leq l \leq 10$), the running time of the algorithm is identical. The reason behind such performance is that the diameter of dataset Epinions is very small and the algorithm traverses almost every vertex in the graph until level 8 of the shortest path tree. Fig. 5.9f shows the running time of the algorithm for dataset LiveJournal, from which it can be seen that it takes more time on finding a solution with the growth of l. Specifically, its running time when $l = 2$ is a tiny fraction of its running time when $l = 10$ (0.001%). Fig. 5.9e further shows that it has the similar behaviours for dataset DBLP-2011.

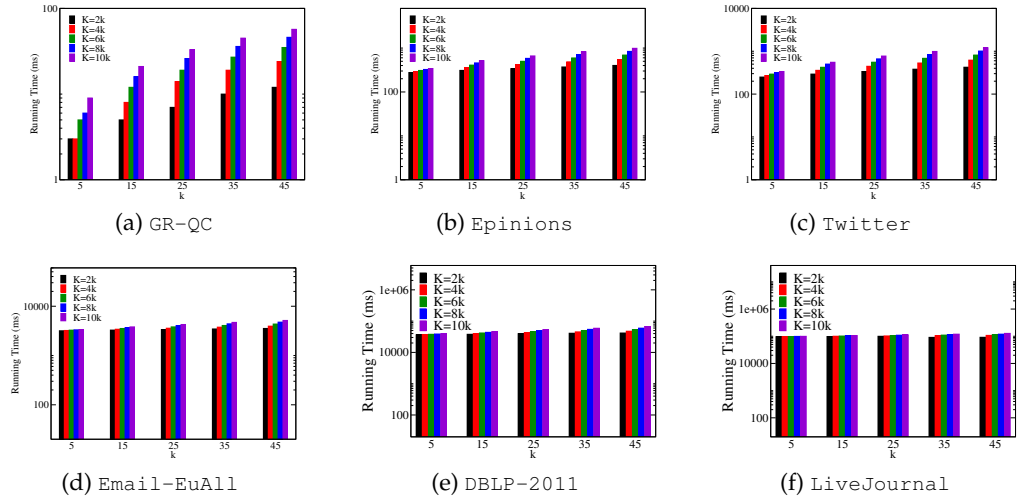
We now study the impact of parameter K on the performance of algorithm BICC. Fig. 5.10a shows that for dataset GR-QC, the performance of the algorithm increases with the increase of K. It is noticed that only for some values of k ($30 \leq k \leq 45$), the performance with $K = 2k$ is promising. The reason behind this observation is that by increasing K, more vertices will be examined by the algorithm and the performance will be improved. Fig. 5.10b shows that performance of algorithm BICC for dataset Epinions is 5% improved by increasing K, for large values of k. From Fig. 5.10d and Fig. 5.10e, it can be seen that the performance of algorithm BICC for datasets



(a) GR-QC (b) Epinions (c) Twitter

(d) Email-EuAll (e) DBLP-2011 (f) LiveJournal

Figure 5.10: Impact of parameter K on performance of algorithm BICC.



(a) GR-QC (b) Epinions (c) Twitter

(d) Email-EuAll (e) DBLP-2011 (f) LiveJournal

Figure 5.11: Impact of parameter K on running time of algorithm BICC.

Email-EuAll and DBLP-2011 is stable, with the increase of K. The rationale behind is that most real complex networks follow the small-world law that vertices can reach each other with a few number of hops, thus the l-bounded inverse closeness centrality of vertices can approximately represent the degree to which a vertex is connected to other vertices. Fig. 5.10f demonstrates that algorithm BICC leads to a better performance for dataset LiveJournal with the growth of K.

Fig. 5.11 plots the running time curves of algorithm BICC, using different values of K for different datasets. Fig. 5.11a shows that by increasing K, the running time of algorithm BICC increases by at least 10% for dataset GR-QC. Fig. 5.11b illustrates that for dataset Epinions, the running time of the algorithm increases by at least 5% when parameter K is increased. Similarly, from Fig. 5.11c it can be seen that the running time of algorithm BICC increases by at least 5%. Fig. 5.11f shows the running

time for dataset LiveJournal, from which it can be seen that the running time of algorithm BICC linearly increases, with the growth of K. The reason is that by increasing the value of K, algorithm BICC will examine more candidate vertices that takes linear amount of time. From Fig. 5.11, it can be interestingly noticed that when the size of dataset is larger, the increase in the running time that is caused by increase in parameter K is less significant, which shows that the running time increases linearly for a larger parameter K.

5.5.6 Discussion of experimental results

We now turn to discussing of the experimental results. Fig. 5.6 demonstrates that in co-authorship datasets GR-QC and DBLP-2011, the performance gain of algorithm AP_BICC over algorithm PageRank is significant. The reason is that algorithm AP_BICC finds vertices who bridge different communities, while algorithm PageRank detects vertices with high reputation. In other words, within co-authorship networks, opinion leaders have a high reputation as they collaborate with many people in their own community; however, the structural hole spanners connect different communities and their absence is more tangible. Similarly, in dataset LiveJournal, bloggers produce contents within their specific category of interest and gain high reputation. The bloggers who are active in multiple categories, however, have a wider perspective, more sources of information and hold more significant positions.

Moreover, Fig. 5.6 plots that the performance of algorithm AP_BICC is similar to PageRank for dataset Email-EuAll. Email-EuAll is the email network of an organization in which high level managers gain a high reputation. The high level managers communicate with the operation managers in each separate division and act as the only communication link between each separate division. Thus, they are also structural hole spanners. A similar explanation can be applied to dataset Twitter, where people who have higher reputation are directly followed by a larger portion of users and their content is forwarded by their followers.

5.6 Summary

We studied the top-k structural hole spanner problem in a large-scale complex network. We first proposed a novel model to measure the quality of structural hole spanners. We then formulated a novel top-k structural hole spanner problem and showed its NP-hardness. After that, we devised two fast yet scalable linear-time algorithms for the problem by exploring the bounded inverse closeness centrality of vertices and articulation points in a network. We finally validated the effectiveness of the proposed model and evaluated the performance of the proposed algorithms through extensive experiments on real and synthetic datasets. Experimental results demonstrated that the proposed model can capture the characteristics of structural hole spanners accurately, and the proposed algorithms are promising.

Chapter 6

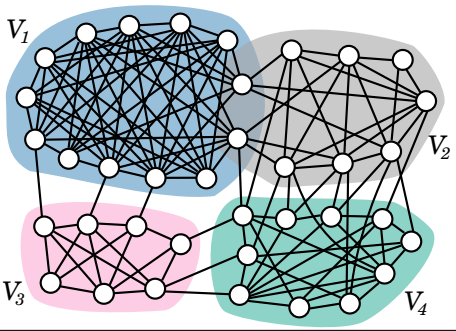
Overlapping Community Structure in Complex Networks

6.1 Overview

Existing fitness metrics for overlapping community detection cause two issues in finding communities. One issue with existing fitness metrics, e.g. conductance and local modularity, is the *separation effect* that the overlapping region is assigned to only one of the two communities. Fig. 6.1 shows that the conductance value of community $V_2 - V_1$ is larger than that of community V_2. Thus, vertices in the overlapping region will be assigned to community V_1 only, as this will result in a larger conductance value. In fact, the vertices in the overlapping region should be assigned to both V_1 and V_2.

Another issue is the presence of the *free rider effect* [Wu et al. 2015]. Bandyopadhyay *et al.* [Bandyopadhyay et al. 2015] proposed an algorithm `FOCS` that expands neighborhoods of vertices, using the subgraph modularity fitness metric. Wu *et al.* [Wu et al. 2015] however showed that the modularity metric and other existing metrics suffer from *free rider effect* [Wu et al. 2015] or *resolution limit* [Fortunato and Barthelemy 2007] on found communities, where a community that is always merged with its densest neighboring community will result in a better community in terms of the adopted fitness metric. Fig. 6.1 illustrates that the values of classic density, relative density and subgraph modularity of community $V_1 \cup V_2$ are larger than those of community V_2, which means that these fitness metrics can cause free rider effect. We later show that fitness metrics relying on only the internal density usually result in free riders, while fitness metrics relying on only the external sparsity usually cause separation effects.

Despite the importance of both free rider and separation effects on overlapping communities, they have not thoroughly been studied yet. The traditional free rider effect [Wu et al. 2015] for non-overlapping communities states that the fitness metric value of the resulting community by merging two communities is better than that of either of the communities. This definition, however, may not apply to overlapping communities, due to the fact that the overlapping region of the two communities should belong to both of them. Furthermore, although triangle-based approaches entail a strong cohesion among vertices in communities [Cohen 2008; Sariyuce et al.



Fitness metrics	$f(V_1)$	$f(V_2)$	$f(V_1 \cup V_2)$	$f(V_1 - V_2)$	$f(V_1 - V_2)$	Detected communities	Remarks
Classic density (CD)	4.38	2.44	3.6	2.28	4.09	$V_1, V_1 \cup V_2$	Free rider effect
Relative density (RD)	0.85	0.47	0.84	0.48	0.73	$V_1, V_1 \cup V_2$	Free rider effect
Subgraph modularity (SM)	5.7	0.91	5.26	0.94	2.81	$V_1, V_1 \cup V_2$	Free rider effect
Local modularity (LM)	0.73	0.61	0.41	0.57	0.81	$V_2, V_1 - V_2$	Separation effect
Conductance (CN)	0.85	0.47	0.26	0.94	0.73	$V_2, V_2 - V_1$	Separation effect

Figure 6.1: A small network and different fitness values for its communities. While communities V_1 and V_2 share two vertices, fitness metrics CD, RD and SM obtain larger fitness values for communities $V_1, V_1 \cup V_2$ (free rider effect), and fitness metrics LM and CN obtain larger values for $V_2, V_1 - V_2$ and $V_1, V_2 - V_1$, respectively (separation effect).

2015; Wang et al. 2010; Zhang and Parthasarathy 2012], the aforementioned fitness metrics measure the quality of communities based on the most obvious structure in networks - the edges, while ignoring more inherent structures within networks such as triangles. Benson *et al.* [Benson et al. 2016] and Tsourakakis *et al.* [Tsourakakis et al. 2016] recently extended the conductance metric by introducing motif concepts, and made use of a given motif as the building block to identify communities, where the number of motif instances in a subgraph is the density of the subgraph, and the number of instances that are partially included in the subgraph is its external sparsity. Particularly, these studies show the effectiveness of triangle motifs in the structure of communities. However, the mentioned conductance metric relies heavily on the external sparsity of communities. Under this metric, the overlapping region of two communities is always assigned to the one with more edge connections.

In this chapter, we will study the free rider effect on overlapping communities, which has not been studied previously [Fortunato and Barthelemy 2007; Huang et al. 2015; Wu et al. 2015]. We will also study the separation effect on overlapping communities. To the best of our knowledge, this is the first overlapping community detection work that considers the quality of overlapping communities, while minimizing both free rider and separation effects on overlapping communities. The main contributions of this chapter are as follows.

- We first introduce a new definition of internal density and external sparsity of communities based on 'asymmetric triangle cuts', and propose a new fitness metric for overlapping community detection that mitigates both free rider and separation effects on communities.

- We then formulate a novel overlapping detection problem based on the proposed fitness metric and show its NP-hardness. We devise an efficient yet scalable algorithm for the problem.
- We finally conduct experiments to evaluate the performance of the proposed algorithm using real-world datasets. Experimental results demonstrate that the proposed algorithm outperforms existing methods, in comparison with the ground-truth communities.

The rest of this chapter is organized as follows. We first propose a new fitness metric for overlapping communities, and define the overlapping community detection problem in Section 6.2. We then show the NP-hardness of the defined problem and devise an algorithm and analyze the time complexity of the algorithm for the problem in Section 6.3 and Section 6.4, respectively. We also evaluate the performance of the proposed algorithm in Section 7.4. We finally summarise the chapter in Section 6.6.

6.2 Problem definition

In this section, we introduce fitness metrics of overlapping communities. We then define the problem of overlapping community detection precisely.

6.2.1 Overlapping community fitness metrics

The aforementioned community fitness metrics are appropriate for non-overlapping community detection. However, they may fail to detect overlapping communities as they may cause *free rider effect* [Wu et al. 2015] and *separation effect* on the found communities. In other words, these metrics dismiss the overlapping region between two communities by either assigning it to only one of them (separation effect) or merging them into a single community (free rider effect). For example, if we adopt the conductance metric in the social network in Fig. 6.1, it can only detect community $V_2 - (V_1 \cap V_2)$ but exclude $V_1 \cap V_2$ that is densely connected to V_2. This implies that the fitness value $\sigma'(V_1) + \sigma'(V_2 - (V_1 \cap V_2))$ is no less than that of $\sigma'(V_1) + \sigma'(V_2)$, using the conductance metric. This will result in the separation effect. In the following, we define free rider and separation effects on overlapping communities formally.

Definition 3 (Separation Effect)**.** *Given a social network $G = (V, E)$ and a fitness metric $f(\cdot)$, the separation effect happens when for any two communities V_1 and V_2,*

$$f(V_1) + f(V_2) < \max\{f(V_1) + f(V_2 - V_1 \cap V_2), f(V_2) + f(V_1 - V_1 \cap V_2)\}. \tag{6.1}$$

It is noticed that community fitness metrics that rely on the number of edges between communities tend to assign the overlapping region of two communities to only one of them, i.e., to the one with more edges connected. Examples of such fitness metrics include conductance and local modularity metrics in Fig. 6.1. On the other hand, the primary cause of free rider effects of existing fitness metrics is that the quality of

a community is measured in terms of the density of its edges, which can be averaged when merging two communities. That is why after merging community V_2 with the denser community V_1 in Fig. 6.1, the fitness value of community $V_1 \cup V_2$ becomes larger than that of community V_2. However, we observe that the fitness value of the merged community $V_1 \cup V_2$ is not necessarily greater than that of the dense community V_1 since they are weakly connected to each other except their overlapping region. We use this intuition to extend the free rider effect on overlapping communities as follows.

Definition 4 (Free Rider Effect)**.** *Given a social network $G = (V, E)$ and a fitness metric $f(\cdot)$, the free rider effect happens when there are two communities V_1 and V_2,*

$$f(V_1) \geq f(V_1 \cup V_2) \quad \wedge \quad f(V_2) < f(V_1 \cup V_2). \tag{6.2}$$

This definition of the free rider effect implies that if a community V_2 is merged with V_1, they will form a new community $V_1 \cup V_2$ with a larger fitness value. However, if the fitness value of the resulting community is larger than only one of them, the free rider will happen. Otherwise, the merge will become valid, and such a merge will not incur the free rider effect. Fig. 6.1 illustrates that fitness metrics such as classic density and relative density can identify community V_1 successfully, but they fail to identify community V_2. They identify community $V_1 \cup V_2$, instead of V_2, which means that they cause free rider effect.

6.2.2 A new fitness metric based on triangle cuts for overlapping community detection

We here propose a new fitness metric for overlapping community detection that can minimize the free rider and separation effects on overlapping communities.

Recent studies [Cohen 2008; Huang et al. 2014; Huang et al. 2015] have shown that triangles can guarantee a strong cohesion among vertices in a community, by which the vertices in the same community are close to each other (the diameter of a subgraph induced by a community is small [Huang et al. 2015]), and the connectivity among the vertices in the community is robust (vertices in a community exhibit a strong edge-connectivity [Cohen 2008; Huang et al. 2015]). Therefore, a good community fitness metric should favour a large number of triangles within a community. Inversely, the connectivity between communities is not a preferred property, since a large number of edges between two communities may not represent strong connections of vertices between the two communities. We thus make use of the number of triangles, instead of the number of edges, between different communities as a proper fitness metric to measure the connectivity among the communities.

Let $\mathcal{V} = \{V_1, ..., Vq\}$ be the collection of overlapping communities in $G(V, E)$. The *overlapping triangle connectivity* of a community $V_i \in \mathcal{V}$ ($1 \leq i \leq q$) is the degree to which the vertices in V_i are connected with each other and sparsely connected to the rest of vertices in G. Thus, all triangles in G can be categorized into two types: *community triangles* and *cut triangles*, where a *community triangle* is a triangle that has all

its three vertices in the community, while a *cut triangle* is a triangle that at least one of its three vertices does not lie in the same community as the other two vertices. Since the number of community triangles indicates the strength of cohesion among the vertices in a community, the number of community triangles will be put in the numerator while the number of asymmetric cut triangles will be put in the denominator of the fitness metric. To balance the overlapping region between communities and avoid the free rider effect, the overlapping size of a community with other communities will be put in the denominator of the fitness metric. Finally, the number of vertices contained in a community will be put in the denominator of the fitness metric to avoid oversized communities and balance the fitness values of communities. We thus define the overlapping triangle connectivity $\tau(V_i)$, as the ratio of the number of community triangles in V_i to the sum of the number of cut triangles, the number of vertices in V_i, and the size of overlapping region with the other communities, i.e.,

$$\tau(V_i) = \frac{|\Delta_G(V_i)|}{|\{\Delta_{uvw} \,:\, u,v \in V_i, w \notin V_i \;\&\; \nexists_{V_j \in \mathcal{V}} u,v,w \in V_j\}| + \sum_{V_j \in \mathcal{V}} |V_i \cap V_j| + |V_i|}. \tag{6.3}$$

Notice that the number of vertices is applied in the denominator of the fitness metric in Eq.(6.3) to normalize the value and the ratio of the number of triangles to the number of vertices. Without the term of the number of vertices in the denominator, a larger community would have a larger fitness value. It can be seen from Fig. 6.1 that the merge of two communities V_1 and V_2 does not increase the fitness value of the resulting community (since the number of vertices increases and the number of triangles per vertex does not increase), using the proposed overlapping triangle connectivity fitness metric $\tau(\cdot)$. Fig. 6.1 also shows that the fitness value of the resulting community does not necessarily increase by merging a community V_2 to another denser community V_1. Thus, the fitness metric $\tau(\cdot)$ does not result in free rider effect on overlapping communities in this example.

A fitness metric $f(\cdot)$ is said to be *monotonically increasing* if for any two subsets $V_1 \subseteq V$ and $V_2 \subseteq V \setminus V_1$, $f(V_1 \cup V_2) \geq f(V_1)$ always holds. Similarly, $f(\cdot)$ is said to be *monotonically decreasing* if $f(V_1 \cup V_2) \leq f(V_1)$. A fitness function is *non-monotonic* if it neither monotonically increases nor monotonically decreases. Monotonicity of a fitness metric $f(\cdot)$ has several implications such as the occurrence of free rider effect [Wu et al. 2015] and the existence of an approximation algorithm for community detection under the fitness metric using hill climbing algorithms [Nemhauser et al. 1978]. In the following we show the defined fitness metric $\tau(\cdot)$ is non-monotonic.

Lemma 5. *The defined fitness metric function $\tau(\cdot)$ is a non-monotonic function.*

Proof. We show the non-monotonicity of function $\tau(\cdot)$, by proving that for a given community $V_j \in \mathcal{V}$, there exist vertices $v, u \notin V_j$ such that $\tau(V_j) \geq \tau(V_j \cup \{v\})$ while $\tau(V_j) \leq \tau(\{V_j \cup \{u\}\})$ as follows.

Let V_j be a community that consists of a clique K_{n-1} ($n > 4$). We first show that there is a vertex $v \in V \setminus V_j$ such that $\tau(V_j) \geq \tau(V_j \cup \{v\})$. The fitness value of clique

K_{n-1} is $\tau(K_{n-1}) = \frac{(n-1)(n-2)(n-3)}{6(n-1)}$. Assume that v is connected to some vertices in V_j but does not form any triangles with the vertices. Now, if vertex v is added to V_j, the fitness value of the resulting community $V_j \cup \{v\}$ will be $\tau(K_{n-1} \cup \{v\}) = \frac{(n-1)(n-2)(n-3)}{6(n-1+1)}$. Clearly, $\tau(K_{n-1}) > \tau(K_{n-1} \cup \{v\})$, or, $\tau(V_j) > \tau(V_j \cup \{v\})$.

We then prove that there is another vertex $u \in V \setminus V_j$ such that $\tau(V_j) \leq \tau(V_j \cup \{u\})$. Let u be a vertex in clique K_{n-1}. Removing u and its incident edges from K_{n-1} leaves us with a clique K_{n-2} with the fitness value $\tau(K_{n-2}) = \frac{(n-2)(n-3)(n-4)}{6(n-2)}$. Now, if u is added to V_j, then $\tau(K_{n-2} \cup \{u\}) = \frac{(n-1)(n-2)(n-3)}{6(n-2+1)}$. Since $n > 4$, $\tau(K_{n-2}) < \tau(K_{n-2} \cup \{u\})$. Thus, the fitness value increases. □

Lemma 5 implies that devising an efficient algorithm for overlapping community detection in G with an objective to maximize $\tau(\mathcal{V})$ $(= \sum_{V_j \in \mathcal{V}} \tau(V_j))$ is extremely difficult, due to the non-monotonicity of function $\tau(\cdot)$. Instead, we will develop an efficient heuristic algorithm for the overlapping community detection problem.

6.2.3 Problem formulation

Given a network $G = (V, E)$ and the overlapping triangle connectivity fitness metric $\tau(\cdot)$, *the overlapping community detection problem* in G is to find a collection of maximal overlapping communities $\mathcal{V} = \{V_1, ..., V_q\}$ such that for every community $V_i \in \mathcal{V}$, the value of the overlapping triangle connectivity fitness metric $\tau(V_i)$ is maximal, where V_i is a community $(\subseteq V)$, $\cup_{i=1}^{q} V_i = V$, $V_i \cap V_j$ $(i \neq j)$ may or may not be empty with $1 \leq i, j \leq q$, and q is the number of communities of G.

6.3 NP-hardness

We show that the overlapping community detection problem is NP-hard, by a reduction from the *relative density community detection problem* [Šíma and Schaeffer 2006] that has been shown to be NP-hard. Since the non-overlapping community detection problem is a special case of the overlapping community detection problem, the NP-hardness of the non-overlapping community detection problem implies the NP-hardness of the overlapping community detection problem.

Let us formally define the decision versions of the relative density and non-overlapping community detection problems as follows.

Definition 5. *Given a graph $G = (V, E)$ and a positive rational number ρ with $0 < \rho \leq 1$, the decision version of the Relative Density Community Detection (RDCD) problem is to determine whether there is a subset of vertices $V' \subset V$ such that $e(V')/(e(V') + e(V', V \setminus V')) \geq \rho$.*

Definition 6. *Given a graph $G = (V, E)$ and a positive rational number $\epsilon > 0$, the decision version of the Non-overlapping Triangle Community Detection (NTCD) problem is to determine whether there is a subset of vertices $V' \subset V$ such that $|\Delta_G(V')|/(|V'| + |\Delta_G(V', V \setminus V')|) \geq \epsilon$.*

Definition 7. *Given a graph $G = (V, E)$ and a positive rational number $\epsilon' > 0$, the decision version of the Simplified Non-overlapping Triangle Community Detection (SNTCD) problem is to determine whether there is a subset of vertices $V' \subset V$ such that $|\Delta_G(V')|/|\Delta_G(V', V \setminus V')| \geq \epsilon'$.*

Note that the SNTCD problem is similar to the NTCD problem, in a sense that only $|V'|$ is omitted from the denominator of the fitness metric, which makes the SNTCD problem easier than the NTCD problem. The following lemma states that the NTCD problem can be reduced to SNTCD problem in polynomial time.

Lemma 6. *The NTCD problem can be reduced to SNTCD problem in polynomial time.*

Proof. One can transform a polynomial time solution to the decision version of SNTCD into a polynomial time solution for the optimization version of SNTCD by binary search on the bound ϵ, and determine the set V' of vertices in the optimal solution in polynomial time. The algorithm for finding the set V' is as follows.

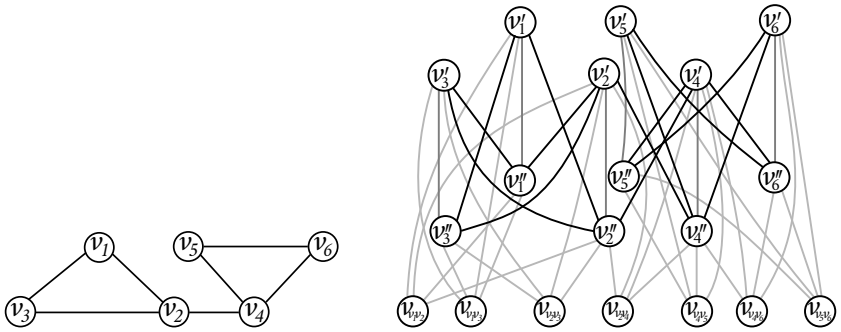
The algorithm proceeds iteratively. Within each iteration, it removes an edge $e \in E$ from G and checks if there is a subset of vertices $C \subset V$ in the resulting graph such that $|\Delta_G(V')|/(|V'| + |\Delta_G(V', V \setminus V')|) \geq \epsilon + 1/n^2$. Having removed edge e from G, if there is still a subset of vertices $V' \subset V$ such that $|\Delta_G(V')|/(|V'| + |\Delta_G(V', V \setminus V')|) \geq \epsilon + 1/n^2$ in the resulting graph, then e is a cut edge (one of its endpoints is in V'); otherwise, e is a community edge (both of its endpoints are in V'). If there is no subset V' such that $|\Delta_G(V')|/(|V'| + |\Delta_G(V', V \setminus V')|) \geq \epsilon + 1/n^2$, but there is a subset V' such that $|\Delta_G(V')|/(|V'| + |\Delta_G(V', V \setminus V')|) \geq \epsilon$, then e is neither a community edge nor a cut edge, it should be removed from G. This procedure continues until all edges in G are examined. Therefore, given a polynomial time algorithm for the NTCD problem, the SNTCD problem can also be solved in polynomial time. That is, given an instance of the SNTCD problem and $\epsilon' = p/q$, the NTCD problem can be solved, using different values of ℓ with $1 \leq \ell \leq n$, i.e., $|\Delta_G(V')|/(|V'| + |\Delta_G(V', V \setminus V')| = p/(\ell + q)$, then determine the set V' and check if $|V'| = \ell$. □

The following theorem shows that the SNTCD problem is NP-complete by a reduction from the RDCD problem.

Theorem 8. *The simplified non-overlapping triangle community detection problem (SNTCD) is NP-complete.*

Proof. We first show that SNTCD belongs to NP. Given a graph $G = (V, E)$, a positive rational number ϵ' and a certificate $C \subset V$, we can count the number of triangles within C, i.e. $|\Delta_G(C)|$, and the ones that have two vertices in C, i.e. $|\Delta_G(C, V \setminus C)|$. We then check if $|\Delta_G(C)|/|\Delta_G(C, V \setminus C)| \geq \epsilon'$. Thus, SNTCD is in NP.

We then show that SNTCD is NP-hard, using a polynomial time reduction from the RDCD problem. Given an instance of the RDCD problem: a graph $G = (V, E)$ and a positive rational number $\epsilon' \leq 1$, we construct an instance of the SNTCD problem containing a graph $G' = (V', E')$ and a positive rational number $\rho = 4\epsilon'/(1 - \epsilon')$ in polynomial time that determines the RDCD problem in polynomial time.



(a) G (b) G'

Figure 6.2: G' is constructed from G in polynomial time. For every vertex $v_i \in V$, there are two vertices v_i', v_i'' in V' and for every edge $(v_i, v_j) \in E$, there is one vertex v_{v_i,v_j} in V'. Every vertex $v_{v_i,v_j} \in V'$ is connected to 4 vertices v_i', v_i'', v_j', v_j'', and every v_i' is connected to v_i''.

Given a graph $G = (V, E)$ and $\epsilon' > 0$, we construct a graph $G' = (V', E')$, where the set V' of vertices contains $2n + m$ vertices that consist of two vertices v' and v'' for every vertex $v \in V$, and a vertex v_{v_i,v_j} for every edge $(v_i, v_j) \in E$ (note that the graph is undirected, therefore (v_i, v_j) and (v_j, v_i) refer to the same edge and we make no distinction between them). The set of edges E' contains $n + 6m$ edges that consist of an edge (v', v'') for every vertex $v \in V$, and six edges (v_i', v_j''), (v_i'', v_j'), (v_i', v_{v_i,v_j}), (v_i'', v_{v_i,v_j}), (v_j', v_{v_i,v_j}) and (v_j'', v_{v_i,v_j}) for every edge $(v_i, v_j) \in E$. From the construction of graph G', it is implied that for every edge $(v_i, v_j) \in E$, there is exactly four triangles in G' ($\Delta_{(v_i', v_j'', v_{v_i,v_j})}$, $\Delta_{(v_i'', v_j', v_{v_i,v_j})}$, $\Delta_{(v_i', v_i'', v_{v_i,v_j})}$, and $\Delta_{(v_j', v_j'', v_{v_i,v_j})}$). Furthermore, the number of triangles in G' is exactly $4m$. Fig. 6.2 illustrates an example of a reduction from graph G to graph G'.

Given a graph $G = (V, E)$ and $\epsilon' > 0$, there is a subset $C \subset V$ of vertices such that $e(C)/(c(C) + e(C, V \setminus C)) \geq \epsilon'$, if and only if, in graph $G' = (V', E')$ (constructed as described), there is a subset $C' \subset V'$ of vertices such that $|\Delta_{G'}(C')|/|\Delta_{G'}(C', V' \setminus C')| \geq 4\epsilon'/(1 - \epsilon')$. Assume that in graph $G = (V, E)$, there exists a subsets of vertices C such that $e(C)/(c(C) + e(C, V \setminus C)) \geq \epsilon'$. Consider a subset of vertices $C' \subset V'$ that consists of $2|C| + e(C)$ vertices, including vertices v', v'' for each vertex $v \in C$, and v_{v_i,v_j} for every edge $(v_i, v_j) \in E(C)$. It can be seen that $|\Delta_{G'}(C')| = 4e(C)$, we thus have $|\Delta_{G'}(C')|/|\Delta_{G'}(C', V \setminus C')| = 4e(C)/|\Delta_{G'}(C', V' \setminus C')|$.

Moreover, for every edge $(v_i, v_j) \in E(C, V \setminus C)$, there is only one triangle $\Delta_{(v_i', v_i'', v_{v_i,v_j})}$ with two endpoints in C', we have

$$
\begin{aligned}
&|\Delta_{G'}(C')|/|\Delta_{G'}(C', V \setminus C')| \\
&= 4(e(C)/e(C, V \setminus C)) \\
&= 1/(e(C, V \setminus C)/e(C) + e(C)/e(C) - 1)
\end{aligned}
$$

Since we assumed that $e(C)/(e(C)+e(C,V\setminus C)) \geq \epsilon'$,

$$|\Delta_{G'}(C')|/|\Delta_{G'}(C',V\setminus C')| \;\; \geq \;\; 4/(1/\epsilon'-1) = \rho.$$

Now, assume that in G' there is a subset $C' \subset V'$ such that $|\Delta_{G'}(C')|/|\Delta_{G'}(C', V \setminus C')| \geq 4\epsilon'/(1-\epsilon')$, we show that there is a subset $C \subset V$ such that $e(C)/(e(C) + e(C, V \setminus C)) \geq \epsilon'$. First, it is noted that if a vertex v' is in C', then its copy v'' is also in C'. For every triangle formed by v' and vertices within C', there is at least one triangle formed by vertices in C' and v''. Therefore, if v'' is excluded from C', the number of cut triangles will be larger than the number of triangles formed by vertex v' and as a result, removing vertex v'' from C' will decrease the fitness value of C'. Therefore, if v' is in C', then v'' is also in C'. Using a similar reasoning, it is also implied that if v_{v_i,v_j} is in C', then vertices v_i', v_i'', v_j', and v_j'' lie in C'.

We finally show that if there is a subset $C' \subset V'$ such that $|\Delta_{G'}(C')|/|\Delta_{G'}(C', V' \setminus C')| \geq 4\epsilon'/(1-\epsilon')$, then there is a subset $C \subset V$ of vertices such that $e(C)/(e(C) + e(C, V \setminus C)) \geq \epsilon'$. For every vertex $v_{v_i,v_j} \in C'$, add both vertices v_i and v_j to C. As a result, the number of edges in $E(C)$ is $|\Delta_{G'}(C')|/4$, and the number of cut edges is equal to the number of cut triangles in C'. Therefore,

$$\begin{aligned} & e(C)/(e(C)+e(C,V\setminus C)) \\ & \quad = \;\; 4|\Delta_{G'}(C')|/(4|\Delta_{G'}(C')| + |\Delta_{G'}(C', V'\setminus C')|) \\ & \quad = \;\; (1+|\Delta_{G'}(C', V'\setminus C')|/4|\Delta_{G'}(C')|)^{-1}, \end{aligned}$$

since $|\Delta_{G'}(C')|/|\Delta_{G'}(C', V' \setminus C')| \geq 4\epsilon'/(1-\epsilon')$,

$$e(C)/(e(C)+e(C,V\setminus C)) \;\; \geq \;\; (1+(1-\epsilon')/\epsilon')^{-1} \geq \epsilon'.$$

Hence, the RDCD problem can be reduced to the SNTCD problem in polynomial time. As the RDCD problem is NP-complete, the SNTCD problem is NP-complete, too. □

6.4 Algorithm

In this section, we first propose an efficient yet scalable algorithm for the overlapping community detection problem, which will deliver a feasible solution. We then show the properties of the overlapping communities delivered by the proposed algorithm and analyze the time complexity of the proposed algorithm.

6.4.1 Algorithm description

To identify high-quality overlapping communities in G, the algorithm consists of two stages. It detects non-overlapping core communities, using the proposed community fitness metric $\tau(\cdot)$. Notice that the core communities are exclusive to each other, they are the bases to form overlapping communities. It then expands the core communities

to form overlapping communities.

In the core community detection, the vertices in $G(V, E)$ is partitioned into core communities so that each core community is a densely connected subgraph, using the fitness metric $\tau(\cdot)$. It is noticed that these core communities are exclusive to each other. Let $\mathcal{V}$ be the set of core communities whose construction proceeds iteratively. Initially, there is only one single community including all the vertices in G, i.e., $\mathcal{V} = \{V\}$. Within iteration k ($k \geq 1$), some of the edges in G will be removed if the support of an edge is no more than k. The edge removal will increase the value of the fitness metric of the resulting connected components. Specifically, for each community $V_i \in \mathcal{V}$ in iteration k, let Φ_k^i be the edges in the induced subgraph $G[V_i]$ with support no greater than k, the set Φ_k^i will be examined to check if its removal can increase the value of the fitness metric of the resulting communities. If yes, the edges in Φ_k^i are removed from G and community V_i is replaced by the number of connected components derived from it. Notice that the support of each remaining edge in the resulting graph will be updated accordingly if the edges in Φ_k^i are removed from G. The value of k is then incremented by one after each iteration. This procedure continues until the support of each edge in the resulting graph is no less than k.

Having found all core communities of G, the core community expansion then follows, by adding vertices from other communities to each core community greedily. Specifically, given the set of core communities $\mathcal{V} = \{V_1, \cdots, V_q\}$, let $\tau(V_i)$ be the overlapping triangle connectivity fitness value of community V_i for all i with $1 \leq i \leq q$. The core community expansion finds a collection $\mathcal{V}$ of overlapping communities, which are local maxima communities according to the fitness metric $\tau(\cdot)$. The core community expansion of each community $V_i \in \mathcal{V}$ proceeds iteratively, by adding a neighbor $v \notin V_i$ of a vertex in V_i such that the value of $\tau(V_i \cup \{v\})$ is the maximum one among all the other neighbors. This iteration is repeated until no such a neighbor can be added to the expanded V_i. The detailed procedure for this is as follows. Let N_{V_i} be the set of neighbors of community V_i that their additions to V_i can increase the fitness value of community V_i. The algorithm finds such a vertex $v \in N_C$ that its addition to C increases the fitness value of $V_i \cup \{v\}$ more than the other vertex $v' \in N_{V_i} \setminus \{v\}$ in N_{V_i}, and v is added to community V_i. Note that only a vertex v is added to community V_i each time and the set N_{V_i} of neighbors then will be updated accordingly. This iterative procedure is repeated for every community $V_i \in \mathcal{V}$ until there is not any neighbor in N_{V_i} that can increase the fitness value of V_i.

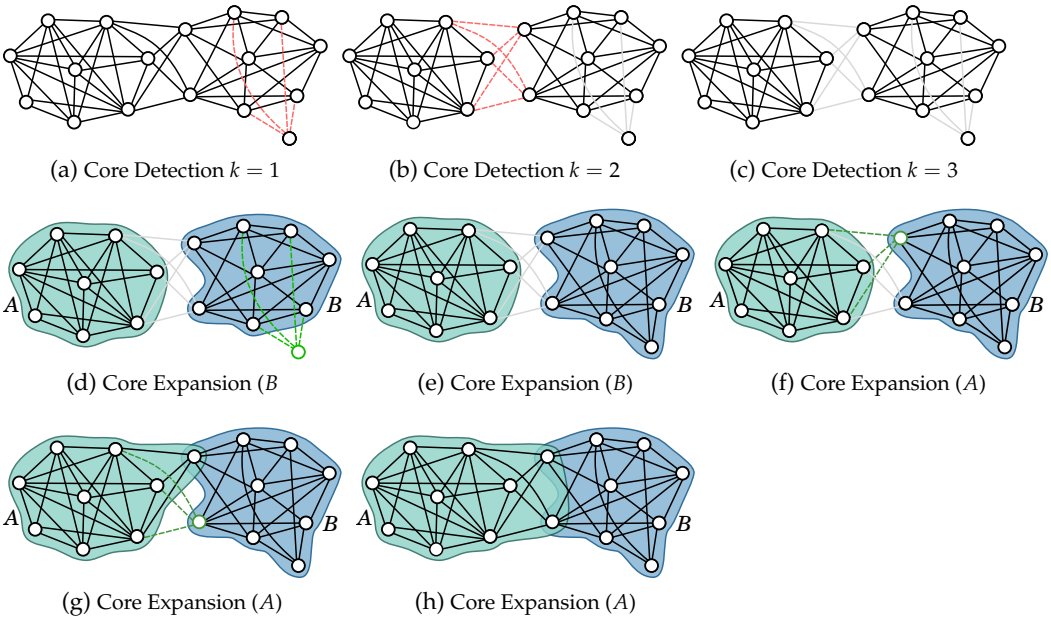
It must be mentioned that although the first stage of the proposed algorithm exhibits some similarities with traditional k-truss detection algorithm [Cohen 2008], they are essentially different. Specifically, the k-truss detection algorithm repeatedly removes edges with support no larger than k for a given k, while the proposed algorithm here starts with $k = 1$ and increments k in each iteration until the sum of the fitness values of all detected communities cannot be further increased. In contrast, traditional k-truss algorithms repeatedly remove edges with support no larger than k, regardless of the sum of the fitness values of the resulting communities [Cohen 2008].

The detailed algorithm for the overlapping community detection problem is given in Algorithm 7.

Algorithm 7 Overlapping_Community_Detection(G)

Input: $G = (V, E)$
Output: Overlapping communities $\mathcal{V}$ of G

/* $\mathcal{V}$ is the collection of detected communities */
$\mathcal{V} \leftarrow \{V\}$;
$k \leftarrow 0$;
/* Find core communities */
Calculate the support of each edge in E, using triangle counting algorithm in [Latapy 2008];
while $\exists e \in E$ with $sup_G(e) \geq k$ **do**
    for each community $V_i \in \mathcal{V}$ **do**
        $\Phi_k^i \leftarrow \{e \mid edge\ e \in G[V_i]$ with $sup_G(e) \leq k\}$;
        $\mathcal{V}' \leftarrow \{V_i' \mid V_i'$ is a connected component in $G[V_i] \setminus \Phi_k^i\}$;
        if $\tau(V_i) \leq \tau(\mathcal{V}')$ **then**
            Remove the edges in Φ_k^i from network G;
            Replace V_i in $\mathcal{V}$ with communities in $\mathcal{V}'$;
    /* Increment the value of k by 1 */
    $k \leftarrow k + 1$;
/* Expand core communities in $\mathcal{V}$ to form overlapping communities */
for each core $V_i \in \mathcal{V}$ **do**
    $N_{V_i} \leftarrow$ Neighbors of V_i, whose addition
        does not decrease the fitness value of V_i;
    while $N_{V_i} \neq \emptyset$ **do**
        $v \leftarrow \text{argmax}_{u \in N_{V_i}} \{\tau(V_i \cup \{u\})\}$;
        $N_{V_i} \leftarrow N_{V_i} \setminus \{v\}$ /* remove vertex v from N_{V_i} */;
        $V_i \leftarrow V_i \cup \{v\}$ /* add vertex v to C */;
        for each neighbor u of vertex v **do**
            if $\tau(V_i \cup \{u\}) \geq \tau(V_i)$ **then**
                Add the neighbor u to N_{V_i};
return $\mathcal{V}$;



(a) Core Detection $k = 1$ (b) Core Detection $k = 2$ (c) Core Detection $k = 3$

(d) Core Expansion (B) (e) Core Expansion (B) (f) Core Expansion (A)

(g) Core Expansion (A) (h) Core Expansion (A)

Figure 6.3: A running example of the proposed algorithm. In the core detection phase, the network is partitioned into core communities and these communities are expanded in the core expansion phase, where the overlapping between communities is detected.

6.4.2 An example of the algorithm execution

We use an example to illustrate the execution of the proposed algorithm, Algorithm 7. Fig. 6.3 shows the execution results of Algorithm 7 on an input social network at different stages.

Fig. 6.3a-Fig. 6.3c illustrate the results of core community detections in stage one, where the edges with low support (red dashed edges) in Fig. 6.3a and Fig. 6.3b are removed until no edge is left, while Fig. 6.3d-Fig. 6.3h show the results of core community expansion in stage two, where community B in Fig. 6.3d is expanded by adding one extra neighbor (the green vertex) into it, the rest of neighbors of the community will not be added, since they would not increase the fitness value of the expanded community. Similarly, in Fig. 6.3f-6.3h community A is expanded by adding more green neighbors, since adding them to the community will increase its fitness value.

It can be seen in Fig. 6.3a that all edges with support no larger than $k = 1$ are removed, as their removal results in the increase in the fitness value of the resulting communities. Similarly, in Fig. 6.3b, when $k = 2$, those edges with support no larger than two are removed. However, Fig. 6.3c shows that the removal of the edges with support no larger than $k = 3$ does not increase the fitness value of communities, therefore, these edges will not be removed in this iteration. It is also noticed that core communities obtained in the first stage are not k-trusses for $k = 3$. This demonstrates the difference between traditional k-truss detection algorithms [Cohen 2008] and this algorithm for core community detections. Fig. 6.3d-6.3h depict the expansion

of communities. Fig. 6.3d-Fig. 6.3e illustrate that addition of one vertex to community B increases the fitness value of B. Similarly, Fig. 6.3f-6.3h show that addition of two vertices to community A increases the fitness value of A.

6.4.3 Algorithm analysis

The rest is to show the properties of overlapping communities delivered by Algorithm 7 and analyze the time complexity of the proposed algorithm as follows.

We first prove that the overlapping communities delivered by Algorithm 7 do not have the separation effect if certain conditions are met.

Lemma 7. *Given a social network $G = (V, E)$ and two core communities $V_1 \subseteq V$ and $V_2 \subseteq V$ obtained in the first stage of Algorithm 7, if there is a set of vertices $V_2' \subseteq V_2$ in which every vertex forms at least two triangles with the vertices in V_1, V_2' will be assigned to both V_1 and V_2 in the overlapping communities found after the second stage of Algorithm 7.*

Proof. Consider two core communities V_1 and V_2 of G. Let V_2' be a subset of V_2 in which each vertex $v \in V_2'$ forms at least one triangle with two vertices in V_1, as illustrated by Fig. 6.4. The vertices in V_2' then will be assigned to V_1 by the fitness metric

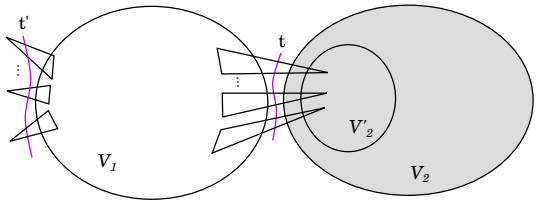


Figure 6.4: Given two communities V_1 and V_2, a subset $V_2' \subset V_2$ forms many triangles with the vertices in V_1.

$\tau(\cdot)$, i.e., $\tau(V_1 \cup V_2') \geq T(V_1)$, where V_2' is a subset of community V_2 such that each vertex in V_2' forms at least two triangles with the vertices in V_1.

Let t be the number of triangles formed by the vertices in V_2' and the vertices in V_1, and let t' be the number of triangles formed by the vertices in V_1 and the vertices in $V \setminus V_1$. We show that the vertices in V_2' will be assigned to V_1 in the second stage of the algorithm by contradiction. Assume that vertices in V_2' will not be assigned to V_1, i.e., $\tau(V_1 \cup V_2') < \tau(V_1)$, then,

$$\tau(V_1 \cup V_2') < \tau(V_1) \Rightarrow \frac{|\Delta_G(V_1 \cup V_2')|}{|V_1 \cup V_2'| + |V_2'| + t'} < \frac{|\Delta_G(V_1)|}{|V_1| + t + t'}. \tag{6.4}$$

Since every vertex in V_2' forms at least one triangle with the vertices in V_1, we have $t \geq 2|V_2'|$,

$$\frac{|\Delta_G(V_1 \cup V_2')|}{|V_1 \cup V_2'| + |V_2'| + t'} < \frac{|\Delta_G(V_1)|}{|V_1| + 2|V_2'| + t'}. \tag{6.5}$$

Since two communities V_1 and V_2 are vertex-disjoint communities, we have

$$V_1 \cap V_2' = \emptyset \Rightarrow |V_1 \cup V_2'| = |V_1| + |V_2'|.$$

Thus,

$$\frac{|\Delta_G(V_1 \cup V_2')|}{|V_1| + 2|V_2'| + t'} < \frac{|\Delta_G(V_1)|}{|V_1| + 2|V_2'| + t'}$$
$$\Rightarrow |\Delta_G(V_1 \cup V_2')| < |\Delta_G(V_1)|. \quad (6.6)$$

Inequality (6.6) does not hold, otherwise this leads to a contradiction that $|\Delta_G(V_1 \cup V_2')| \geq |\Delta_G(V_1)| + t$. Therefore, vertices in V_2' will be assigned to V_1 in the second stage of Algorithm7, since $\tau(V_1 \cup V_2') \geq \tau(V_1)$. □

We then show that the community fitness metric $\tau(\cdot)$ avoids free rider effect if a certain condition is met by the following lemma.

Lemma 8. *Given a social network $G = (V, E)$ and two core communities $V_1 \subseteq V$ and $V_2 \subseteq V$ found in the end of the first stage of Algorithm 7, if there is a set of vertices $V_2' \subseteq V_2$ in which every vertex forms at least one triangle with the vertices in V_1, community V_1 and V_2 will not be merged in the second stage of Algorithm 7 unless $\tau(V_1) \leq \tau(V_1 \cup V_2)$ and $\tau(V_2) \leq \tau(V_1 \cup V_2)$.*

Proof. We show that the community fitness metric $\tau(\cdot)$ avoids free rider effect if a certain condition is met by contradiction. Considering the proof for Lemma 7, we need to show that $\tau(V_2') \leq T(V_1 \cup V_2')$ to avoid the free rider effect on the overlapping communities found in the second stage of Algorithm 7. Without loss of generality, we assume that $\Delta_G(V_1)/|V_1| > \Delta_G(V_2')/|V_2'|$. Assume that $\tau(V_2') > T(V_1 \cup V_2')$,

$$\tau(V_1 \cup V_2') < \tau(V_2') \Rightarrow \frac{|\Delta_G(V_1 \cup V_2')|}{|V_1| + |V_2'| + t'} < \frac{|\Delta_G(V_2')|}{|V_2'| + t' + t''}$$

where t'' is the number of triangles that have two vertices in V_2' and one vertex outside V_2',

$$\frac{|\Delta_G(V_1)| + |\Delta_G(V_2')| + t}{|V_1| + |V_2'| + t'} < \frac{|\Delta_G(V_2')|}{|V_2'| + t' + t''}$$
$$(|\Delta_G(V_1)| + |\Delta_G(V_2')| + t)(|V_2'| + t' + t'') <$$
$$(|\Delta_G(V_2')|)(|V_1| + |V_2'| + t')$$

Since $t'' > t$,

$$\begin{aligned}
(|\Delta_G(V_1)| + t)(|V_2'| + t' + t'') &+ \Delta_G(V_2')t'' < \\
&(|\Delta_G(V_2')|)|V_1|, \\
(|\Delta_G(V_1)| + t)(|V_2'| + t' + t'') &< (|\Delta_G(V_2')|)(|V_1| - t''), \\
\frac{|\Delta_G(V_1)| + t}{|V_1| - t''} &< \frac{|\Delta_G(V_2')|}{|V_2'| + t' + t''},
\end{aligned}$$

which is a contradiction to the initial hypothesis that $\Delta_G(V_1)/|V_1| > \Delta_G(V_2')/|V_2'|$. The lemma thus holds. □

The following theorem studies the time complexity of the algorithm.

Theorem 9. *Given a social network $G = (V, E)$, Algorithm 7 for the overlapping community detection problem will deliver a feasible solution in time $O(|V| \cdot |E|)$, if the support of each edge is constant.*

Proof. Following Algorithm 7, the core community detection in G starts with calculating the support of each edge of G in time $O(|E|^{3/2})$, using the algorithm in [Latapy 2008], and it proceeds iteratively. Within each iteration, the set $\Phi_k = \bigcup_{V_i \in \mathcal{V}} \Phi_k^{V_i}$ of edges with support no larger than k is found, using the values calculated at its step 4. It then calculates the fitness scores of the resulting connected components in $G[V_i] \setminus \Phi_k^{V_i}$ in linear time. Each iteration of the while-loop of Algorithm 7 takes $O(|E|)$ time. Since the maximum support among edges in a real social network usually is constant, the number of iterations k is constant. The time spent on core community detection by Algorithm 7 is $O(|E|^{3/2})$.

In the core community expansion stage of Algorithm 7, each vertex v is added into set V_i at most once. Initially, neighbors of community V_i are added to set N_{V_i}, then a vertex $u \in N_{V_i}$ with maximum $\tau(V_i \cup \{u\})$ is added to V_i. When a vertex v is added to N_C, the value of $\tau(N_{V_i} \cup \{v\})$ is calculated, by finding the number of triangles formed by every edge in $E[V_i]$ connected to v in $O(|E[V_i]|)$ time, assuming that a heap data structure is adopted for keeping track of vertices in N_{V_i}. Then, the insertion and extraction of vertices in N_{V_i} (represented by the heap data structure) takes $O(|V| \cdot \log |V|)$ time. Thus, overlapping community identification derived from the found core communities takes $O(q \cdot |E|)$ time if there are q core communities. While $q \leq |V|$, the time complexity of Algorithm 7 thus is $O(|E|^{3/2} + q \cdot |E|) = O(|E|^{3/2} + |V| \cdot |E|) = O(|V| \cdot |E|)$. □

It must be mentioned that the analytical time complexity of Algorithm 7 is very conservative. Its actual running time on real social networks is much faster, which is almost linear to the problem size $|V| + E|$, due to the sparsity of social networks. This can be witnessed from later empirical evaluation results (see Fig. 6.6).

6.5 Experimental results

In this section, we evaluate the performance of the proposed algorithm against several benchmark algorithms using several real-world datasets. We also study separation and free rider effect on the overlapping communities delivered by different algorithms under different fitness metrics including the one proposed in this paper.

6.5.1 Experimental environment settings

We start with the experimental environment settings and descriptions of different data sets, evaluation metrics, and benchmark algorithms.

Benchmark algorithms. To evaluate the performance of the proposed algorithm, Algorithm 7, denoted by `CoreExp`, for the overlapping community detection problem, the following state-of-the-arts will be adopted for the benchmark purpose.

- `FOCS` [Bandyopadhyay et al. 2015] – A local expansion algorithm, which finds communities starting from neighborhoods of vertices. This algorithm expands the initial communities by adding vertices using the local modularity fitness metric.

- `Demon` [Coscia et al. 2012] – An agent-based algorithm, in which, every vertex v receives a label l, where l is the label appeared in majority of neighbors of v. The labels are propagated iteratively until every vertex has the label of most of its neighbors. Finally, communities that have more than a certain overlap are merged.

- `Bigclam` [Yang and Leskovec 2013] – A matrix factorization-based algorithm, which assigns each vertex-community a value that represents the membership in the community. It then models the probability of an edge as a function of the shared community affiliations and identifies network communities by fitting the model to the given network, and estimating the latent factors.

- `SeedExp` [Whang et al. 2013] – A local expansion algorithm, which consists of four phases: filtering, seeding, seed expansions, and propagations. The filtering phase removes weakly connected subgraphs. The seeding phase finds seed vertices, which are expanded in the seed expansion phase. The propagation attaches the weak components to communities.

- `Bayes` [Gopalan and Blei 2013] – A Bayesian model-based algorithm, which posits a probabilistic model of networks where each vertex can belong to many communities. It finds the conditional distribution of the hidden communities given the observed network. It then approximates the conditional distribution with various methods in combination with stochastic optimization by iteratively subsampling the network and estimation of the hidden communities.

Real datasets. We make use of seven real datasets, which have been widely adopted in literature [Xie et al. 2013] and are publicly available[1]. Specifically, dataset `Amazon` is based on the Amazon products, where there is an edge between products i and j if product i is frequently co-purchased with product j, and each product category provided by Amazon is a ground-truth community. Dataset `DBLP` is a collaboration network, where ground-truth communities are defined as publication venues, e.g., journals or conferences. Dataset `Orkut` is the friendship network of Orkut members, in which communities are the groups that users create and other users join in. Dataset `LiveJournal` is the friendship network of users in LiveJournal blogging web site. Users can define groups and join multiple groups. These groups are considered as the ground-truth communities. Dataset `Facebook` consists of ego networks of Facebook users, which has been collected from survey participants. The groups provided by users are the ground-truth communities. Dataset `Twitter` consists of 'lists' from Twitter. The social communities are the ground-truth communities in Twitter. Dataset

[1] http://snap.stanford.edu/data/index.html

Table 6.1: Details of real datasets, where $\mathcal{C}^*$ represents the set of ground-truth communities and $sup_{\max}$ is the largest support of edges.

Dataset	$\|V\|$	$\|E\|$	$\|\mathcal{C}^*\|$	$sup_{\max}$
Facebook	4,039	88,234	193	293
Twitter	81,306	2,420,766	4,065	9,016
Google Plus	107,614	30,494,866	468	11,488
Amazon	334,863	925,872	253,345	161
DBLP	317,080	1,049,866	13,477	213
Orkut	3,072,441	117,185,083	15,301,901	9,145
LiveJournal	3,997,962	34,681,189	658,401	1,393

`Google Plus` is a social network in Google+. The groups that are defined by users represent ground-truth communities. Notice that each of the datasets `Facebook`, `Twitter` and `Google Plus` in fact is a combined network consisting of ego networks from SNAP. Table 7.1 details the mentioned datasets in our experiments.

Evaluation measures. quantitatively measuring the quality of detected overlapping communities in a social network is challenging, as different measures lead to different quality of overlapping communities. We here employ two widely-adopted measures [Gleich and Seshadhri 2012; Gopalan and Blei 2013; Whang et al. 2013; Xie et al. 2013; Yang and Leskovec 2013] for analyzing the accuracy of the detected communities by different algorithms, i.e., F_1 and F_2-measures [Xie et al. 2013].

Let $\mathcal{C}^*$ be the set of ground-truth communities and $\mathcal{C}$ the detected communities by any mentioned algorithm. The F-measure is based on the precision and recall of each community C compared to C^* defined as follows.

$$p(C, C^*) = \frac{|C \cap C^*|}{|C|}, \quad r(C, C^*) = \frac{|C \cap C^*|}{|C^*|}$$

F_1-measure [Xie et al. 2013] is the harmonic mean of the precision and recall, while F_2-measure [Xie et al. 2013] magnifies the impact of recall in the results, as follows.

$$F_1 = \frac{1}{|\mathcal{C}|} \sum_{C \in \mathcal{C}} \max_{C^* \in \mathcal{C}^*} \left\{ \frac{2 \cdot p(C, C^*) \cdot r(C, C^*)}{p(C, C^*) + r(C, C^*)} \right\}, \tag{6.7}$$

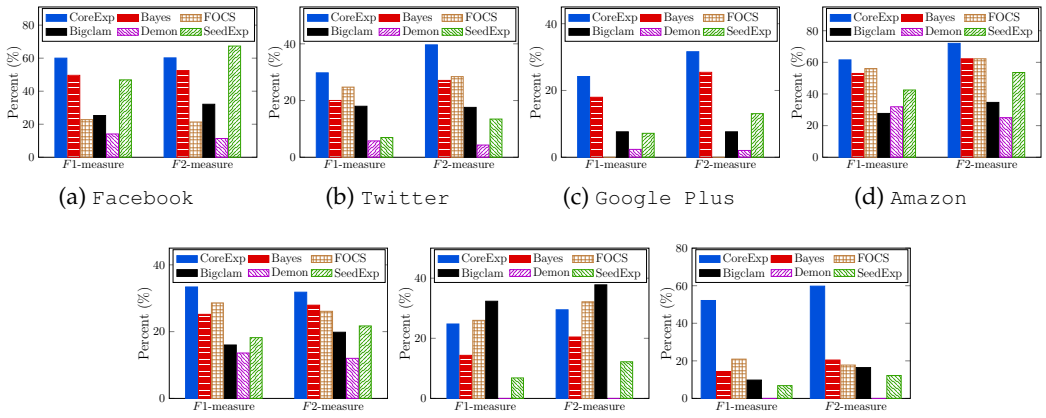
$$F_2 = \frac{1}{|\mathcal{C}|} \sum_{C \in \mathcal{C}} \max_{C^* \in \mathcal{C}^*} \left\{ \frac{5 \cdot p(C, C^*) \cdot r(C, C^*)}{4 \cdot p(C, C^*) + r(C, C^*)} \right\}. \tag{6.8}$$

Note that all the experiments are conducted on a desktop with a 3.06GHz CPU and 16GB memory.

6.5.2 Performance evaluation of different algorithms

We first study the performance of the proposed algorithm `CoreExp` against the benchmark algorithms, by evaluating the quality of the found communities under two measures: F_1-measure and F_2-measure. We also compare the running times of different algorithms and the number of communities delivered by each of them. Fig. 6.5 plots the quality bars of the communities delivered by different algorithms, using different

datasets and community fitness metrics. Notice that algorithm `Demon` did not terminate after 48 hours for datasets `LiveJournal` and `Orkut`. The results on those datasets are not available, and thus cannot be shown in the bar chart.



(a) `Facebook` (b) `Twitter` (c) `Google Plus` (d) `Amazon`

(e) `DBLP` (f) `LiveJournal` (g) `Orkut`

Figure 6.5: The quality of overlapping communities found by various algorithms compared with the quality of the ground-truth communities under different community fitness metrics. Note that algorithm `Demon` did not terminate for datasets `LiveJournal` and `Orkut` after 48 hours, thus its results thus are not available in the plots.

Fig. 6.5 shows that algorithm `CoreExp` delivers the most accurate communities for most datasets (6 out of the 7 datasets). Specifically, it can be seen from Fig. 6.5a that for dataset `Facebook`, algorithm `CoreExp` outperforms all other algorithms at least by 10% in both F_1 and F_2-measures. Similarly, Fig. 6.5b demonstrates that algorithm `CoreExp` outperforms all other algorithms by at least 12% in the ego network of dataset `Twitter`. Fig. 6.5c indicates that algorithm `CoreExp` outperforms all other algorithms by at least 15% for dataset `Google Plus`. For the dataset `Amazon`, Fig. 6.5d shows that algorithm `CoreExp` is superior to other mentioned algorithms by at least 10% under both F_1-measure and F_2-measure. It can also be seen from Fig. 6.5e that for the dataset `DBLP`, algorithm `CoreExp` outperforms the other algorithms by at least 11% based on both F_1 and F_2 measures. It is noticed that for dataset `LiveJournal` Fig. 6.5f indicates that both algorithm `Bigclam` and algorithm `FOCS` beat algorithm `CoreExp`. This may partially contribute to the specific topological structure of the LiveJournal network. However, the better quality solution delivered by algorithm `Bigclam` is at the expense of prohibitive running time that is several orders of magnitudes greater than that of algorithm `CoreExp`. On the other hand, despite that algorithm `FOCS` has a less running time compared with algorithm `CoreExp`, the communities delivered by algorithms `Bigclam` and `FOCS` will cause separation and free rider effects that can be seen from Table 6.2, while the communities delivered by algorithm `CoreExp` can avoid such effects. Fig. 6.5g demonstrates that algorithm `CoreExp` is the best one among all comparison algorithms which has the highest accuracy of com-

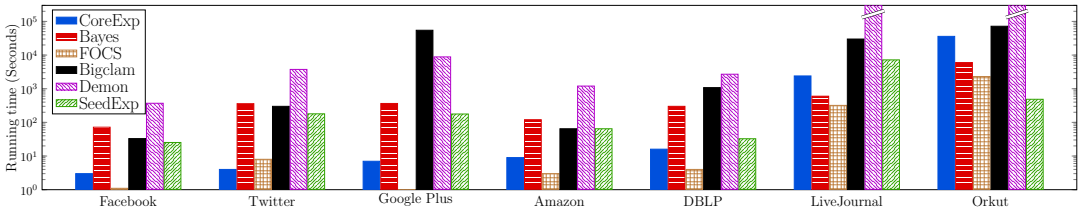


Figure 6.6: The running times of various algorithms. The bars with parallel lines represent algorithms that did not terminate.

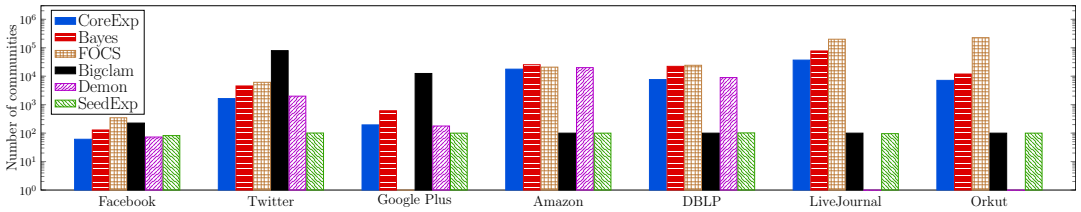


Figure 6.7: The number of detected communities by various algorithms. Algorithm `Demon` did not terminate for datasets `LiveJournal` and `Orkut`, thus their results on those datasets thus will not be available from the bar chart.

munities found for dataset `Orkut`. The reason is that dataset `Orkut` contains more than 100 million edges and identifying communities in such a large-scale network is challenging. Algorithm `CoreExp` demonstrates that even for such a massive dataset, it outperforms the other algorithms significantly in terms of both the quality of the solution and the running time (see Fig. 6.6).

Fig. 6.6 plots the running times of different algorithms. It is seen that algorithm `CoreExp` takes a few hours for the massive dataset such as `Orkut` with more than 100 million edges and less than one hour for dataset `LiveJournal` with more than 34 million edges. It can be observed from Fig. 6.6 that the running time of algorithm `CoreExp` is at least 10% less than that of all other algorithms except that of algorithms `Bayes` and `FOCS`. However, the quality of the solutions delivered by algorithms `Bayes` and `FOCS` is not as good as that by algorithm `CoreExp`. Similarly, for dataset `Orkut`, the running time of algorithm `SeedExp` is the smallest, while the communities obtained by `SeedExp` have the poor quality for this dataset.

Fig. 6.7 depicts the number of communities delivered by each mentioned algorithm, from which, it can be seen that algorithm `CoreExp` delivers more communities than that of any other mentioned algorithm except algorithms `Bayes` and `FOCS`. However, the quality of the solution delivered by algorithm `CoreExp` is better than either by algorithms `Bayes` and `FOCS` in most cases.

6.5.3 Separation and free rider effects on the communities found by different algorithms

We then evaluate separation and free rider effects on the communities found by different algorithms, using a synthetic dataset, as there is not any available information

Table 6.2: Separation and free rider effects caused by different algorithms, where "–" means that algorithm does not terminate after 48 hours.

Algorithm	Percentage of separation effect	Percentage of free rider effect
CoreExp	0%	0%
Bayes	7.60%	2.036%
FOCS	65.320%	0%
Bigclam	64.24%	18.55%
Demon	4.150%	2.330%
SeedExp	–	–

of these two effects on real datasets. To examine whether the communities delivered by different algorithms cause separation and free rider effects, we generate a synthetic dataset based on SSCA[2] synthetic networks. We first generate SSCA networks with network size from 2^{10} to 2^{20}, where each SSCA network consists of a set of cliques. We then place a vertex v_{ij}, if there is an edge between two cliques C_i and C_j, and connect vertex v_{ij} to each vertex in cliques C_i and C_j with a probability 0.5. Vertex v_{ij} is then treated as an overlapping vertex between communities C_i and C_j, i.e, it will belong to both overlapping communities derived from C_i and C_j, respectively. Note that the synthetic networks randomly generated may not be connected, and algorithm `SeedExp` thus is inapplicable in this case.

Table 6.2 lists the experimental results of free rider and separation effects of the communities delivered by different algorithms, using the constructed synthetic datasets. For each benchmark algorithm, the percentage of vertices in the overlap that were assigned to only one community represents the separation effect, and the percentage of detected communities spanning more than one community in the ground-truth communities represents the free rider effect. The values reported in Table 6.2 are averaged across all synthetic networks.

Table 6.2 indicates that algorithm `CoreExp` can avoid both free rider and separation effects in these synthetic datasets. It can be observed from Table 6.2 that the percent for separation effect by algorithm `FOCS` is 65.320%, which means that a large number of vertices in the overlap were assigned to only one community. The reason is that algorithm `FOCS` uses the modularity fitness metric, which is prone to separation effect. Similarly, the percent of separation effect in algorithm `Bigclam` is 64.24%, which means that algorithm `Bigclam` also causes separation effect. Table 6.2 also shows that the percent of free rider effect caused by algorithm `Bigclam` is 18.55%, which is considerably larger than the other community detection algorithms. This number is followed by the percent of free rider effect by algorithms `Demon` and `Bayes`. While algorithm `CoreExp` can avoid both free rider and separation effects on these datasets, none of the benchmark algorithms can avoid both the effects.

[2]http://www.cse.psu.edu/∼kxm85/software/GTgraph/

6.6 Summary

We investigated the problem of finding high quality overlapping communities from large-scale complex networks by taking both separation and free rider effects on found overlapping communities into consideration. To this end, we first proposed a novel community fitness metric - the overlapping triangle connectivity fitness metric for overlapping community detection. We then devised an efficient yet scalable algorithm for the problem. We finally validated the effectiveness of the proposed fitness metric and evaluated the performance of the proposed algorithm, by conducting extensive experiments on seven real-world datasets. Experimental results show that the proposed algorithm is very promising and outperforms state-of-the-arts.

Chapter 7

Community Search in Complex Networks

7.1 Overview

The aim of this paper is to search all communities that are relevant to a given set of query vertices without assuming that the given query vertices necessarily belong to the same community. While removing this assumption makes the problem general, it poses new challenges. Particularly, one key challenge is that the number of communities that may contain a given set of query vertices is *unknown*. This raises two questions: (1) how to efficiently group query vertices into different communities, as there is an exponential number of ways in which query vertices can be grouped together, and (2) how to effectively measure the quality o fa community that covers the given query vertices, which would require an effective way to expand communities for a given group of query vertices that are related to each other.

To the best of our knowledge, the only research that has assumed that query vertices may not belong to the same community is that of Ruchansky et al. [Ruchansky et al. 2017], where the community search problem is defined as the problem of finding a subgraph with minimum *inefficiency* that is not necessarily connected and contains the query vertices. However, it can be shown that the inefficiency of a connected subgraph is always smaller than its disconnected subgraph. Thus, the optimal solution to the minimum inefficiency problem is always a connected subgraph.

We formally define the problem of community search, which is compatible with the assumption that query vertices may belong to different communities. We first propose the notion of *propinquity* to measure the closeness between two vertices. We show how to utilise the propinquity measure to determine whether query vertices are in the same community or not, which enables us to find more than one community per search. We ultimately exploit the propinquity measure to examine the community membership of the given query vertices with other vertices in the network and expand the communities. Thus, we formally define the problem of community search in a network using a given propinquity measure, as the problem of finding a maximal subgraph where two vertices are connected if their propinquity is no less than a given ℓ.

We propose a novel instance of propinquity measure and develop two efficient algorithms for the community search problem. It is observed that a small length of shortest path between two query vertices implies the closeness between them. However, only one shortest path may be insufficient to capture the propinquity between two vertices, which denotes the likelihood that two vertices belong to the same community. Nevertheless, the top-k shortest paths provide more information about the closeness between query vertices. We thus propose a novel propinquity measure in terms of the top-k shortest paths, whose sum is no larger than a given ℓ. We develop our community expansion algorithms based on our second observation that the shortest path between vertices in a community is unlikely to leave the community, due to the highly connectivity among vertices in a community. In other words, given two vertices that lie in the same community, the vertices on the shortest path between them are likely to be in the same community. Since the shortest paths are least expected to leave the community, the vertices on the shortest paths are expected to be highly relevant to the query vertices, making the community resilient to the so-called free-rider effect. Our experimental results show that the proposed algorithm is capable of producing results that are significantly more accurate than the existing algorithms.

Our contributions in this chapter are summarized as follows.

- We propose a novel notion, called propinquity, to measure the closeness between vertices, which represents the likelihood that two vertices belong to the same community.
- We formally define the generic community search problem, under which various existing community search methods can be formulated as special cases.
- We devise two efficient algorithms for the community search problem in large-scale networks, which are capable of handling queries with more than one vertex and finding more than one community per search.
- We conduct experiments on real-world datasets. Our experimental results show that the proposed algorithms deliver communities that highly match with ground-truth communities, outperforming the benchmark algorithms.

The remainder of this chapter is structured as follows. In Section 7.2 we formally define the community search problem and propose a novel definition for propinquity. We devise two algorithms for community search in Section 7.3 and analyze their time complexities. We then present the results of our extensive experiments on real networks in Section 7.4. We summarise the chapter in Section 7.5.

7.2 Community search problem

In this section, we formally define the generic community search problem based on the notion of propinquity between vertices.

Given two vertices u and v in a network G, let $\Delta(u, v)$ denote the *propinquity* value between vertices u and v, which represents the closeness of two vertices u and v to each other, or how conveniently they can communicate with each other across a network. Generally, a large $\Delta(u, v)$ means that u and v are likely to be in the same community, while a small $\Delta(u, v)$ means that they are less acquainted with each other, thus less likely to be in the same community. Since the strength of a chain is measured in terms of the strength of its weakest link, we measure the propinquity of a set of vertices V' as the smallest propinquity between pairs of its vertices, i.e. $\Delta(V') = \min_{u,v \in V'} \{\Delta(u, v)\}$.

We now define the community search problem in terms of a given propinquity measure.

Definition 8 (Community Search Problem)**.** *Given a network $G = (V, E)$, a query $Q \subseteq V$ and a positive number k, the community search problem is to detect a minimum cover $\mathcal{C} = \{V_1, \cdots, V_t\}$ ($t \leq |Q|$) such that:*

1. *the propinquity between every two vertices u and v in V_i ($V_i \in \mathcal{C}$) is no less than k, i.e. $\Delta(V_i) \geq k$,*

2. *every community $V_i \in \mathcal{C}$ is maximal, and*

3. *for every pair of query vertices $q_i, q_j \in Q$ the following holds,*

$$\Delta(q_i, q_j) \geq k \Rightarrow \exists\, V_p \in \mathcal{C} \;\; (\{q_i, q_j\} \subseteq V_p).$$

We study several properties for propinquity measure such as reflexiveness, symmetry and transitivity to narrow down special cases of the community search problem. We say a propinquity measure Δ is *reflexive* if for every vertex u in V we have $\Delta(u, u) = \infty$. A propinquity Δ is *symmetric* if $\Delta(u, v) = \Delta(v, u)$ for any two vertices u and v. To characterize a special case of the community search problem, we here define the transitivity property for the propinquity measure as follows.

Definition 9 (Transitive propinquity)**.** *Given a network $G = (V, E)$ and a positive number k, a propinquity measure Δ is transitive if $\Delta(u, v) \geq k$ and $\Delta(v, w) \geq k$ implies $\Delta(u, w) \geq k$ for any three vertices u, v and w in V.*

Note that the definition of transitive propinquity is similar to the definition of transitive binary relation. It is possible to convert the propinquity measure to a binary relation R_Δ with respect to a given k, where $uR_\Delta v$ if and only if $\Delta(u, v) \geq k$ and $\forall u, v, w \in V (uR_\Delta v \wedge vR_\Delta w \Rightarrow uR_\Delta w)$.

Using different definitions for propinquity may lead to different community structures. In the following, we show how the proposed problem can serve as a unified form to explain the existing works on community search. We then propose a novel definition for propinquity that addresses issues with the existing ones.

7.2.1 NP-hardness

In the following, we show the community search problem is NP-hard, using a reduction from the maximum clique problem. We first introduce the decision version of a special case of the community search problem that deals with queries of size 2.

Definition 10 (CSD). *Given a network $G = (V, E)$, two vertices $q_1, q_2 \in V$, a positive number k and a positive integer r, the community search decision (CSD) problem is to determine if there is a community V_1 of size r that*

1. *if the propinquity between q_1 and q_2 is no less than k, then they both belong to V_1, i.e. $q_1, q_2 \in V_1$,*
2. *for every pair of vertices $u, v \in V_1$ the propinquity between u and v is no less than k, i.e. $\Delta(u, v) \geq k$.*

We now show that the maximum clique problem can be reduced to the CSD problem.

Theorem 10. *The CSD problem is NP-complete.*

Proof. We first show that the CSD problem is verifiable in polynomial time. Given a certificate for the CSD problem, one can check $\Delta(u, v) \geq k$ for all pairs (u, v) in the solution (V_1) in polynomial time. Thus, the CSD problem is in NP.

We reduce the maximum clique problem to the CSD problem. Let $I = \langle G, r \rangle$ be an instance of the maximum clique decision problem, in which we are interested in determining if there is a clique of size r in G. We construct an instance $I' = \langle G', Q, k, r' \rangle$ of the CSD problem, where the solution of I' determines the solution of I. That is, there is a clique of size r in G if and only if there is a community of size r' with propinquity k in G' for query Q.

We now show how to construct $G' = (V', E')$ from $G = (V, E)$. In order to construct G', for every vertex in G we add a vertex in G', and for every edge in G we add an edge to G'. For every edge $(u, v) \in E$, let $\Delta(u, v) = 2$, and if u and v are not connected with an edge let $\Delta(u, v) = 0$. We then add two vertices q_1 and q_2, that are connected to all vertices and $\Delta(q_1, q_2) = 2$. Note that for every vertex $v \in V'$, we let $\Delta(q_1, v) = \Delta(q_2, v) = 2$. Let the query consists of vertices q_1 and q_2, i.e. $Q = \{q_1, q_2\}$, and let $r' = r + 2$ and $k = 1$. Since two query vertices q_1 and q_2 have a propinquity 2, they must lie in the same community. We show that network G' consists of a community V_1 with size r' if and only if the network G contains a clique with size r. Let V_1 be the community detected for the aforementioned instance of the CSD problem, i.e. $I' = \langle G', Q, k, r' \rangle$. If the cardinality of V_1 is r', this means that there are r vertices in G' where the minimum propinquity is 1, and there is one edge between every pair of those vertices in G; therefore, there is a clique with size r in G. Similarly, if there is no community with size r' in G' for the CSD problem, this means that there is no set of vertices with size r that form a clique. Therefore, the reduction is complete. □

The following theorem can now be obtained.

Theorem 11. *The community search problem is NP-hard.*

Proof. Since CSD is a special case of the community search problem, the theorem is implied. □

Although the general form of the community search problem is NP-hard, there are several cases of this problem with special propinquity measures that can be solved in polynomial time. The following lemma shows that a special case of the community search problem can be solved in polynomial time.

Theorem 12. *Given a network $G = (V, E)$, a query $Q \subseteq V$ ($|Q| \leq 2$), and a positive number k, if the propinquity measure Δ has the transitive property, then the community search problem can be solved in polynomial time.*

Proof. Since the propinquity measure Δ is transitive and the size of query is at most 2, we can devise a polynomial time algorithm for the problem. The algorithm considers three cases: (1) there is only one vertex in the query, (2) two query vertices are not in the same community, (3) two query vertices are in the same community.

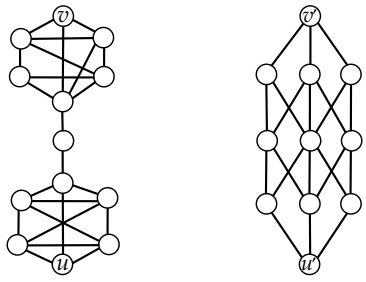
Case (1): If the query Q consists of only one vertex q, then we first find a set V' of all vertices $v \in V$ such that $\Delta(q, v) \geq k$. Then we create a graph $G' = (V', E')$, where there is an edge between two vertices u and v in E' if and only if $\Delta(u, v) \geq k$. It can be shown that the set of vertices in the connected component of G' with the largest number of vertices is the community which is associated with q. In order to prove that the set of vertices in the largest connected component of G' is the community for q, we use contradiction. Let us assume that $C \subseteq V'$ is the set of vertices in the largest connected component of G'. We assume that there is one vertex v in the solution of the community search problem C^* ($|C^*| > |C|$) that is not included in C, i.e. $v \notin C$. Since v is in C^*, then for every vertex u in C^* we have $\Delta(u, v) \geq k$, which means that v is connected to all vertices in C^* in graph G'. Thus, $G'[C^*]$ is a connected component of G', which is a contradiction to the fact that C with $|C| < |C^*|$ is the largest connected component of G'.

Case (2): If the value of the propinquity between query vertices is strictly smaller than k, then for every vertex q_i in the query Q, we find the appropriate community similar to case (1). The correctness of this case can be proved with a similar proof of Case (1).

Case (3): If the propinquity between query vertices q_1 and q_2 is no smaller than k, then we find a set V' of all vertices $v \in V$ such that $\Delta(q_1, v) \geq k$ and $\Delta(q_2, v) \geq k$. We then create a graph $G' = (V', E')$, where there is an edge between two vertices u and v, if and only if $\Delta(u, v) \geq k$. It can be shown that the set of vertices in the largest connected component of G' is the community which is associated with vertices in Q. This case can also be proved in a similar way as Case (1). □

7.2.2 A novel propinquity measure

We now propose a new propinquity measure based on the top-k shortest paths between vertices in a network. We show that the proposed propinquity measure avoids the free rider effect.



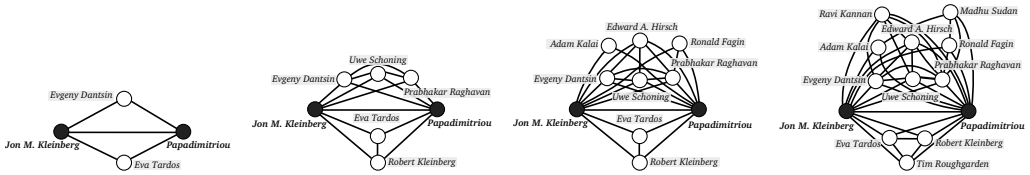
(a) (b)

Figure 7.1: An example that illustrates the distances between two different pairs of vertices are equal, while the structure of network around these pairs is completely different.

Given a network $G = (V, E)$, one may naturally consider the inverse of the length of a shortest path between two vertices can be considered as a propinquity measure between the two vertices. However, existence of hubs [Watts and Strogatz 1998] and structural hole spanners [Rezvani et al. 2015] implies that complex networks exhibit a small-world characteristic, by which the diameter of such networks is small. As a result, the length of a shortest path between any pair of vertices is usually small, and it may obscure the propinquity of vertices with each other. This may lead to inaccurate communities found for a query Q.

Nonetheless, the top-k shortest paths between vertices can provide sufficient information about the propinquity between vertices. Although the length of a shortest path between a pair of vertices is usually small in complex networks, the top-k shortest paths between them may vary, which helps expose the propinquity between pairs of vertices. Fig. 7.1 shows two examples, where the distance between u and v is equal to the distance between u' and v'; however u' and v' are densely connected, while u and v belong to different communities. Fig. 7.1a illustrates how a structural hole spanner bridges vertices in two communities to shorten the shortest paths between vertices in those communities. However, although the first shortest path between u and v is small, i.e. 4, the second, third and fourth shortest paths between them are 5, 6 and 6, respectively. On the other hand, the first, second, third and fourth shortest paths between u' and v' are all 4. Therefore, we here suggest using top-k shortest paths in the propinquity, instead of only one single shortest path. Specifically, we consider two vertices are close enough if the sum of lengths of the top-k shortest paths between them is no larger than a given ℓ.

Let $N_1^{(1)}(v)$ denote the set of vertices whose shortest paths to vertex v are no larger than 1, i.e. the 1-neighborhood of vertex v, and let $N_2^{(1)}(v)$ denote the vertices whose shortest paths to vertex v are no larger than 2, i.e. the 2-neighborhood of vertex v, and let $N_\ell^{(1)}(v)$ denote that set of vertices whose shortest paths to v are no larger than ℓ, i.e. the ℓ-neighborhood of vertex v. Similarly, let $N_\ell^{(2)}(v)$ be the set of vertices that the sum of top-2 shortest paths between these vertices and v is no larger than ℓ. Let $N_\ell^{(k)}(v)$ be the set of vertices that the sum of the top-k shortest paths between these vertices and vertex v is no larger than ℓ. The propinquity between two vertices u and



(a) $\ell = 5$ (b) $\ell = 11$ (c) $\ell = 17$ (d) $\ell = 23$

Figure 7.2: Communities that were detected for a query {Kleingerg, Papadimitriou} using dataset DBLP.

v with respect to a certain ℓ is then defined as:

$$\Delta(u, v; \ell) = \max\{k : u \in N_\ell^{(k)}(v)\}. \tag{7.1}$$

Notice that the propinquity measure defined in Eq. 7.1 represents how close two vertices are in a network and whether they are likely to be in the same community. Given two vertices u and v, a small value of propinquity between them (i.e. $\Delta(u, v; \ell)$ is small) means that u and v are far from each other, because the sum of the lengths of the top-k shortest paths between them is larger than ℓ, therefore they are less likely to be in the same community. In contrast, a large value of propinquity between u and v means that there are many paths with short lengths between u and v, implying that they are likely to be in the same community.

Fig. 7.2 illustrates the novel propinquity in action when searching communities for two researchers in computer science, i.e. Jon Kleinberg and Christos Papadimitriou, using different values of ℓ. Fig. 7.2 shows that vertices on the shortest paths between query vertices represent other researchers who work in the same area and reveal the community structure around query vertices. Furthermore, it can be seen in Fig. 7.2 that vertices within the communities are densely connected and the distance between them is quite short. Specifically, the diameter in each community is 2. Also, with the increase of the value of parameter ℓ from Fig. 7.2a to Fig. 7.2d, the number of vertices being identified in the community increases. This shows the flexibility of the proposed propinquity in identifying communities.

7.2.3 Discussion on free rider effect

Free rider effect is a common issue with many community search methods [Wu et al. 2015; Huang et al. 2015]. Several existing works [Wu et al. 2015] have used a fitness metric that evaluates the quality of a given community. Wu et al. [Wu et al. 2015] discussed the free rider effect in such community search methods . We here explain why existing works suffer from the free rider effect based on a certain property of propinquity. We use the formal definition of free rider effect used by Huang et al. [Huang et al. 2015].

Definition 11 (Free Rider Effect [Huang et al. 2015])**.** *Given a network $G = (V, E)$ and a query $Q \subseteq V$, let H be a community found by a community search method based on a fitness metric $f(\cdot)$. Let H^* ($H^* \not\subseteq H$) be a community with $f(H^*) > f(H)$. If $f(H \cup H^*) \geq f(H)$,*

we say that $f(\cdot)$ suffers from the free rider effect and H^* is called a free rider community for query Q.

The following lemma shows that if a propinquity measure is transitive then it is prone to the free rider effect.

Lemma 9. *Given a network $G = (V, E)$, a query $Q \subseteq V$ and parameter k, let H be a community found for Q, where $\Delta'(H) = k$. If there exists a subgraph H^*, with $\Delta'(H^*) \geq k$ and $\Delta'(u, v) \geq k$ for any $u \in H$ and $v \in H^*$, then $\Delta'(H \cup H^*) \geq \Delta'(H)$, i.e. Δ' causes the free rider effect.*

Proof. Based on the definition of transitive propinquity $\Delta'(u, v) \geq k$ for any two vertices $u \in H$ and $v \in H^*$ implies that $\Delta'(x, y) \geq k$ for every two vertices $x, y \in H \cup H^*$. Therefore, $\Delta'(H \cup H^*) \geq k$. Thus, the transitive propinquity Δ' may cause the free rider effect and the lemma is proved. □

In the following we demonstrate how our proposed propinquity measure can avoid such free rider effect.

Lemma 10. *Given a network $G = (V, E)$, a query $Q \subseteq V$ and parameter k, let H be a community found using the community search problem with the propinquity measure Δ. If there exist a subgraph H^* with $\Delta(H^*) > \Delta(H)$ and $\Delta(H \cup H^*) \geq \Delta(H)$ then $H^* \subseteq H$.*

Proof. We prove this lemma using contradiction. Assume that H^* is not a subset of H, yet $\Delta(H \cup H^*) \geq \Delta(H)$. Therefore, there exists at least one vertex $u \in H^* - H$. Since $\Delta(H \cup H^*) \geq \Delta(H)$, for every vertex $v \in H$, $\Delta(u, v) \geq \Delta(H)$. Thus, for every vertex $v \in H$, we have $\Delta(u, v) \geq k$, which is a contradiction to the fact that H is maximal as stipulated in Definition 8. □

7.3 Community search algorithm

In this section, we propose two efficient algorithms for the community search problem. One is called clique-based algorithm for community search (CAC) and the other is called fast algorithm for community search (FAC). Each algorithm consists of two main stages: (1) identifying the community profile of query vertices, which determines community membership of query vertices, and (2) expanding communities of query vertices. In the first stage, each algorithm determines which query vertices are in the same community using the pairwise propinquity between query vertices, while in the second stage, each algorithm expands the communities using top-k shortest paths to find other vertices that are in the same community as query vertices. Procedure 3 describes the overall process of each algorithm.

It is noted that the main difference between algorithms CAC and FAC is in the first stage, i.e. identifying the community profile. More specifically, this stage finds a profile which consists assignment of query vertices to communities. Such community profile of query vertices have a significant impact on the accuracy of the communities found by the algorithms.

Procedure 3 CommunitySearch(G, Q, ℓ, k)

Input: $G = (V, E)$, Query Q, ℓ, k
Output: Communities of vertices in Q

```
1: /* Stage 1: Determine community membership */
2: G_Q ← ConstructQueryRelevanceGraph(G, Q, ℓ, k);
3: 𝒞 ← Find communities of query vertices in G_Q;
4: /* Stage 2: Expand each community of 𝒞 */;
5: i ← 0;
6: for each community C in 𝒞 do
7:     i ← i + 1;
8:     V_i ← ExpandCommunity(G, Q, C, ℓ, k);
9:     𝒞 ← 𝒞 \ {C} ∪ {V_i};
10: output 𝒞;
```

Procedure 4 ConstructQueryRelevanceGraph(G, Q, ℓ, k)

Input: $G = (V, E)$, Query Q, ℓ, k
Output: Query relevance graph $G_Q = (Q, E_Q)$

```
1: E_Q ← ∅;
2: G_Q ← (Q, E_Q);
3: for each q_i and q_j ∈ Q do
4:     if Δ(q_i, q_j; ℓ) ≥ k then
5:         E_Q ← E_Q ∪ {(q_i, q_j)};
6: return G_Q = (Q, E_Q);
```

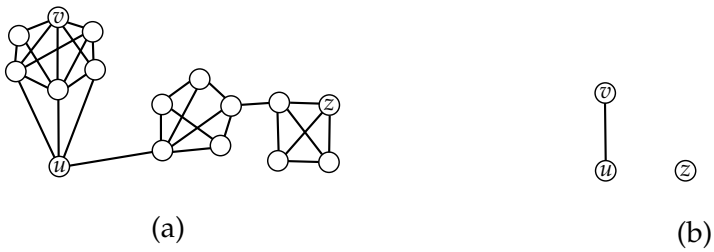
7.3.1 Identifying the community profile

Given a set of query vertices, some vertices may belong to different communities. Therefore, it is crucial to group these query vertices into communities. A community profile for query vertices is an assignment of query vertices to one or more communities, where each vertex can belong to more than one community. Let us define the query relevance graph, which enables us to determine the community membership for query vertices.

Definition 12 (Query Relevance Graph)**.** *Given a network $G = (V, E)$, a query Q, two positive integers ℓ and k, the query relevance graph is a graph $G_Q = (Q, E_Q)$ such that its vertices are the query vertices, and there is an edge between two vertices q_1 and q_2 in G_Q if $\Delta(q_1, q_2) \geq k$.*

Procedure 4 details the construction of the query relevance graph.

Example 4. *Fig. 7.3(b) illustrates the query relevance graph of query vertices $Q = \{v, u, z\}$ in the graph of Fig. 7.3(a), when parameters are set as suggested in the experiments section, i.e. $k = 4$ and $\ell = 11$. It is noticed that the set of vertices in this query relevance graph matches the set of vertices in the given query. Since, the propinquity value between only vertices u and*



(a) (b)

Figure 7.3: Given a graph shown in (a), the query relevance graph of query vertices $Q = \{u, v, z\}$ is shown in (b), with the suggested parameters $k = 4$ and $\ell = 11$.

v is no smaller than 4, there is an edge only between these two vertices. The query relevance graph can show the community membership of query vertices more clearly.

In the following, we discuss how query relevance graphs can be used to determine the community membership among query vertices.

7.3.1.1 Clique-based algorithm

In CAC, the community membership among query vertices is determined using maximal cliques of the query relevance graph. The proposed community search problem in this chapter implies that if the propinquity between two query vertices q_i and q_j is at least k, then there must be a community V_p which includes both query vertices. Simultaneously, two vertices may belong to the same community only if the propinquity between them is at least k. Therefore, maximal cliques of the query relevance graph must satisfy the first condition of the community search problem, which dictates that two query vertices may belong to the same community if and only if their propinquity is no smaller than k. Notice that these maximal cliques may overlap with each other.

Please note that the set of maximal cliques of the query relevance graph does not necessarily provide the *minimum cover* of query vertices. Instead it finds a feasible cover. In order to find a minimum cover of query vertices, one may need to solve the *edge clique cover* problem [Orlin 1977] in the query relevance graph.

Here we propose using a maximal clique detection algorithm, such as the one proposed by Bron and Kerbosch [Bron and Kerbosch 1973], on the query relevance subgraph to identify groups of query vertices that belong to the same community.

7.3.1.2 Fast algorithm

Since the running time of existing algorithms for finding maximal cliques in a network is exponential [Bron and Kerbosch 1973], in FAC we propose a fast approach for determining the community membership among query vertices based on their connectivity in the corresponding query relevant graph.

This approach determines whether the query vertices are in the same community or not using the query relevance graph in a slightly different way. Two vertices $q_i \in Q$

and $q_j \in Q$ are considered to be in the same community if there is a path between them in the query relevance graph. In other words, two query vertices lie in the same community if they belong to the same connected component of the query relevance graph.

7.3.2 Expanding communities

In order to find communities for query vertices, we use the communities of query vertices identified from the query relevance graph by either algorithms CAC or FAC. Given a current community C of the query relevance graph G_Q, the algorithm initializes a new community C' using the vertices in C. The algorithm processes two different cases: (1) C consists of only one query vertex, and (2) C consists of more than one query vertex. If C contains only one query vertex, the algorithm first finds a neighbour u of $q \in C$ with the largest propinquity value $\Delta(u, q; \ell)$ and then expands the community C' by adding the top-k shortest paths between u and q to community C'. However, if C contains more than one query vertex, the algorithm then repeatedly adds to C' the vertices of the top-k shortest paths between every two pair of vertices that have an edge between them in the query relevance graph G_Q. In other words, let $q_i \in Q$ and $q_j \in Q$ be two vertices that have an edge in the query relevance graph G_Q. The algorithm first finds the top-k shortest paths between q_i and q_j in G and then adds to the community all vertices of the top-k shortest paths. To find the top-k shortest paths in G between a pair of vertices q_i and q_j, the algorithm starts with $k' = 1$ and repeatedly finds the k'-th shortest path between q_i and q_j in G. Procedure 5 details the community expansions.

Procedure 5 ExpandCommunity(G, Q, C, ℓ, k)

Input: $G = (V, E)$, Q, C, ℓ, k
Output: Expanded community C'

$C' \leftarrow C$;
if $|C| > 1$ **then**
  for each q_i and q_j in C **do**
    if $\Delta(q_i, q_j; \ell) \geq k$ **then**
      $KP \leftarrow$ top-k shortest paths between q_i & q_j;
      $C' \leftarrow C' \cup KP$;
else if $|C| = 1$ **then**
  for $q \in C$, let $u \leftarrow \text{argmax}_{v \in N(q)}\{\Delta(v, q; \ell)\}$;
  $C' \leftarrow C' \cup \{\text{top-}k \text{ shortest paths between } u \text{ \& } q\}$;
return C';

7.3.3 Algorithm analysis

The rest is to analyze the time complexity of algorithms CAC and FAC.

Theorem 13. *Given a network $G = (V, E)$, a query $Q \subseteq V$ and two positive integers ℓ and k, algorithm CAC delivers a feasible solution to the community search problem in G in time $O(3^{|Q|}|Q| + |Q|^2 k(|V| + |E|))$, where ℓ and k are the input arguments of Procedure 3.*

Proof. Procedure 3 consists of two stages: (1) clique based algorithm for determining community membership, and (2) expanding communities (Procedure 5). We study the time complexity of each procedure separately.

In Steps 3-6 of Procedure 4, pairwise propinquity of query vertices is calculated. Calculating the propinquity for every pair of vertices can be done using a top-k shortest path algorithm. The best known algorithm for the top-k shortest paths runs in time $O(k(|V| + |E|))$ for unweighted graphs [Katoh et al. 1982]. Therefore, the time complexity of Procedure 4 is $O(|Q|^2 k(|V| + |E|))$. Next, the algorithm needs to find the maximal cliques of the query relevance subgraph, where the time complexity of the Bron and Kerbosch algorithm [Bron and Kerbosch 1973] for maximal clique detection [Bron and Kerbosch 1973] is $O(3^{|Q|}|Q|)$.

In Steps 2-4 of Procedure 5, the set of vertices in the top-k shortest paths can be found using the same top-k shortest paths algorithm in time $O(k(|V| + |E|))$. As this procedure can be done for all pairs of query vertices in Q, the time required for Steps 2-4 is $O(|Q|^2 k(|V| + |E|))$. Thus, the overall time complexity of Algorithm 3 is $O(|Q|^2 k(|V| + |E|))$. □

Theorem 14. *Given a network $G = (V, E)$, a query $Q \subseteq V$ and two positive integers ℓ and k, algorithm FAC delivers a feasible solution to the community search problem in G in time $O(|Q|^2 k(|V| + |E|))$, where ℓ and k are the input arguments of Procedure 3.*

Proof. Procedure 3 consists of two stages: (1) determining community membership (Procedure 4), and (2) expanding communities (Procedure 5). We study the time complexity of each procedure separately.

In Steps 3-6 of Procedure 4, pairwise propinquity of query vertices is calculated. Calculating the propinquity for every pair of vertices can be done using a top-k shortest path algorithm. The best known algorithm for the top-k shortest paths runs in time $O(k(|V| + |E|))$ for unweighted graphs [Katoh et al. 1982]. Therefore, the time complexity of Procedure 4 is $O(|Q|^2 k(|V| + |E|))$.

In Steps 2-4 of Procedure 5, the set of vertices in the top-k shortest paths can be found using the same top-k shortest paths algorithm in time $O(k(|V| + |E|))$. As this procedure can be done for all pairs of query vertices in Q, the time required for Steps 2-4 is $O(|Q|^2 k(|V| + |E|))$. Thus, the overall time complexity of Algorithm 3 is $O(|Q|^2 k(|V| + |E|))$. □

Note that the analysis on the time complexities $O(3^{|Q|}|Q| + |Q|^2 k(|V| + |E|))$ and $O(|Q|^2 k(|V| + |E|))$ for Procedure 3 are very conservative. In our experiments, the real running times of both algorithms CAC and FAC are nearly linear-time in terms of the number of edges.

Parameter setting. Although communities in different networks share general characteristics, such as the density of connections among vertices and sparsity of connec-

tions between communities, their exact density of connections and size is inherited from the network, thus may differ due to different network topologies. Communities in some networks are small, while relatively large in other networks, and connections among vertices in a community depends on the nature of the relationship between vertices. Thus, one of the vital features of a good community search algorithm is to provide a flexible way to deal with differences in network topologies. Two parameters ℓ and k are the keys that can be tuned to reveal communities in different networks.

It is possible to probe the proper values for parameters ℓ and k using a binary search approach, since the values of parameters k and ℓ are bounded. Specifically, we start by setting the values of these parameters as small as possible, i.e. $\ell = k = 2$. We then continuously double the values of ℓ and k, until the quality of the obtained communities starts to decrease. We then reduce the value of parameters, until the quality of the obtained communities starts to decrease. This process continues until the proper values of ℓ and k are found. We later demonstrate this process in the experimental results section.

7.4 Experimental results

We have evaluated the performance of our proposed community search algorithms on real datasets. In this section, we start with presenting the experimental settings (Section 7.4.1). We then discuss the performance of the proposed algorithms against several benchmark algorithms.

7.4.1 Experimental settings

We present the settings of the experiments in this subsection, including datasets, benchmark algorithms, evaluation metrics and the computing environment.

Datasets. We adopted four real datasets that are publicly available[1], and have been widely used in the literature [Barbieri et al. 2015; Huang et al. 2014; Huang et al. 2015; Shan et al. 2015b; Wu et al. 2015]. Dataset Amazon is a network of Amazon products, where if a product i is frequently co-purchased with another product j, there is an edge between i and j. Products are categorized by Amazon, and each category provided by Amazon is a ground-truth community. Dataset DBLP is a collaboration network, where ground-truth communities are defined as major journals or conferences. Dataset Youtube is a social network formed by users of the video sharing website Youtube. Users in this network can create groups, and each user-defined group is considered as a ground-truth community. Dataset LiveJournal is a friendship network of users in the LiveJournal blogging web site. Users can create groups and join different groups. These groups are considered as the ground-truth communities. Table 7.1 summarizes the details of datasets.

In order to evaluate the accuracy of different algorithms in finding the ground-truth communities (Sections 7.4.2, 7.4.3, and 7.4.4), for each dataset in Table 7.1, we use

[1] `http://snap.stanford.edu/data/index.html`

250 queries of sizes 2, 4, 6, 8 and 10 that are chosen uniformly at random (50 samples per size). In order to ensure that each random query lies in at least one community, we first choose a community randomly, and then randomly select two vertices from that community. We repeat this operation $|Q|/2$ times to generate a random query of size $|Q|$.

For calculating the performance of the proposed algorithm in determining the membership of query vertices in a community, we randomly select 150 pairs of query vertices in the following manner: 50 pairs of vertices are selected completely randomly, regardless of their community membership; 50 pairs are selected such that vertices in each pair are in the same community; and 50 pairs are selected such that vertices in each pair are in two different communities. We then calculate the accuracy of the proposed algorithms in determining the community membership for these queries.

Benchmark algorithms. We compare the results of our proposed algorithms, denoted by `CAC` and `FAC`, with the following benchmark algorithms,

- `MIS` [Ruchansky et al. 2017] – In `MIS`, a community is defined as a subgraph with minimum inefficiency that includes all vertices of a query. Each connected component in a minimum inefficiency subgraph is treated as a separate community. We test the algorithm with the set of parameters suggested in the original paper [Ruchansky et al. 2017].

- `CSM` [Barbieri et al. 2015] – In `CSM`, a community is defined as a subgraph that includes all vertices of the query, and the smallest degree of its vertices is maximized. We use the greedy algorithm with ShellStruct as descrbed in the original paper [Barbieri et al. 2015].

- `CTC` [Huang et al. 2015] – In `CTC`, a community is defined as a k-truss subgraph that includes all vertices of the query, and the diameter of this subgraph is minimized.

Evaluation metrics. Measuring the quality of detected communities in a network is challenging, as different metrics lead to different quality of communities. We here employ two widely-adopted measures in the literature [Huang et al. 2015] for quantitatively analyzing the accuracy of the detected communities by different algorithms. Given a query Q, let $\mathcal{C}^*$ be the set of ground-truth communities that contain at least one vertex from Q, and $\mathcal{C}$ be the set of communities detected by one of the mentioned algorithms. The F-measure is based on the precision and recall. The precision and recall of a detected community $C \in \mathcal{C}$ compared to a ground-truth community $C^* \in \mathcal{C}^*$, are defined as follows.

$$p(C, C^*) = \frac{|C \cap C^*|}{|C|}, \quad r(C, C^*) = \frac{|C \cap C^*|}{|C^*|}. \tag{7.2}$$

Table 7.1: Details of Datasets

Dataset	# vertices	# edges	# communities
Amazon	334,863	925,872	253,345
DBLP	317,080	1,049,866	13,477
Youtube	1,134,890	2,987,624	8,385
LiveJournal	3,997,962	34,681,189	658,401

$F1$ and $F2$-measures [Huang et al. 2015] are then defined as follows.

$$F1 = \frac{1}{|\mathcal{C}|} \sum_{C \in \mathcal{C}} \max_{C^* \in \mathcal{C}^*} \left\{ \frac{2 \cdot p(C, C^*) \cdot r(C, C^*)}{p(C, C^*) + r(C, C^*)} \right\}, \tag{7.3}$$

$$F2 = \frac{1}{|\mathcal{C}|} \sum_{C \in \mathcal{C}} \max_{C^* \in \mathcal{C}^*} \left\{ \frac{5 \cdot p(C, C^*) \cdot r(C, C^*)}{4 \cdot p(C, C^*) + r(C, C^*)} \right\}. \tag{7.4}$$

To evaluate the performance of the proposed algorithm in determining the community membership, we also use the traditional definitions of precision and recall, as follows.

$$p(C, C^*) = \frac{tp}{tp + fp}, \quad r(C, C^*) = \frac{tp}{tp + fn}, \tag{7.5}$$

where *tp* stands for the number of true-positives, *fp* for the number of false-positives and *fn* stands for the number of false-negatives.

All the experiments were conducted on a desktop with an Intel(R) Core(TM) i7 CPU (3.40GHz) with 32GB memory.

7.4.2 Quality evaluation

We evaluate the proposed algorithms, i.e. `CAC` and `FAC`, against the benchmark algorithms for random queries by setting parameters $\ell = 11$ and $k = 4$. Based on our experimental results in Section 7.4.5, these values of parameters provide the best solutions in the majority of datasets. Fig. 7.4 shows the performance of different algorithms on four different datasets in terms of the quality of detected communities. It can be seen in Fig. 7.4a that for dataset Amazon, the accuracy of results delivered by `CAC` and `FAC` in terms of F1-measure is 100% higher than those of algorithms `CTC`, `CSM` and `MIS`, while for larger queries ($|Q| > 2$), the performances of algorithms `CAC` and `FAC` are at least three times better than those of algorithms `CTC`, and `CSM` and `MIS`. A similar trend is observed in Fig. 7.4e that presents the accuracy of different algorithms in terms of F2-measure, where the performance of algorithms `CAC` and `FAC` is 60% better than that of algorithms `CTC`, `CSM` and `MIS` for the smallest query size, and the accuracy is nearly 100% higher than other benchmark algorithms for larger queries. It can be seen that the performance of algorithm `CTC` has a decreasing trend in Fig. 7.4a and Fig. 7.4e. The reason is that algorithm `CTC` finds only one community per search, while when the size of a query becomes larger, more communities are involved.

Fig. 7.4b plots the accuracy of the results delivered by different algorithms for dataset DBLP, in terms of F1-measure. It can be seen in Fig. 7.4b that algorithm `CAC`

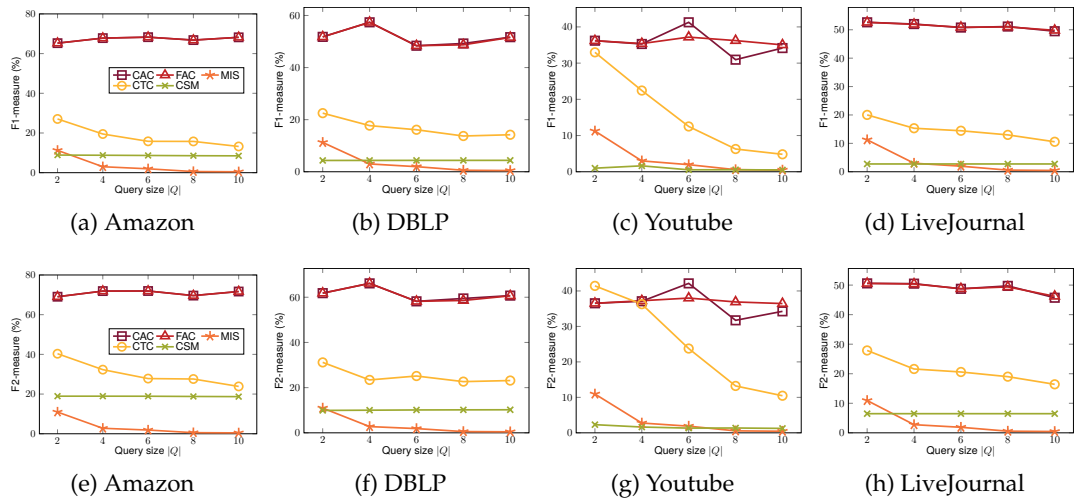
Table 7.2: Comparison of running times (seconds), where IT stands for Indexing Time and ST stands for Searching Time.

Algorithm	Amazon		DBLP		Yoube		LiveJournal	
	IT	**ST**	**IT**	**ST**	**IT**	**ST**	**IT**	**ST**
CAC	0	0.041	0	0.365	0	15.779	0	16.066
FAC	0	0.038	0	0.205	0	15.598	0	14.432
CTC	8.575	46.645	13.6	57.938	58.814	752	1,364	2,762
CSM	156	577	122	381	4,990	13,969	10,519	14,317

outperforms algorithms CTC, CSM and MIS by at least 100% across all query sizes in terms of F1-measure. In a similar manner, algorithm FAC outperforms algorithms CTC, CSM and MIS by at least 90% across different query sizes. Fig. 7.4b also shows that algorithms CAC and FAC have some minor fluctuations in the accuracy, which is due to the randomness of the queries. Similarly, in Fig. 7.4f, the accuracy of the results provided by CAC is at least 110% higher than those of CTC, CSM and MIS across all query sizes. Fig. 7.4f also shows that algorithm FAC outperforms algorithms CTC, CSM and MIS by at least 10% in small queries ($|Q| = 2$) and by at least 110% for larger queries ($|Q| > 2$). One of the interesting trends in Fig. 7.4b and 7.4f is that the accuracy of algorithm CTC has a significant decreasing trend, as the size of query increases, which is due to finding only one community per search.

Fig. 7.4c illustrates the accuracy of the communities detected by different algorithms for dataset Youtube, in terms of F1-measure. It can be observed in Fig. 7.4c that algorithm CAC outperforms algorithms CTC, CSM and MIS by at least 25% in terms of F1-measure for query size $|Q| = 2$ and by at least 100% for larger queries ($|Q| > 2$). Similarly, the accuracy of communities found by algorithm FAC is almost the same as algorithm CTC for small queries ($|Q| = 2$), while the gap between these accuracies increases significantly as the size of queries become larger. It is noted that the accuracy gain of algorithms CAC and FAC over the benchmark algorithms for larger query sizes increases rapidly, because the other algorithms fail to deliver more than one community per search. Similarly, in Fig. 7.4g, the accuracy of the results provided by CAC is slightly higher than that of algorithm CTC for small queries ($|Q| = 2$), while the performance gap between algorithms CAC, CTC and MIS increases with larger queries. On the other hand, the accuracy of communities found by algorithm FAC for small queries ($|Q| = 2$) is lower than that of algorithm CTC, while for larger queries, algorithm FAC consistently outperforms algorithms CTC. Fig. 7.4b and 7.4f illustrate that the accuracy of algorithm CTC has a significant decreasing trend, as the size of query increases. The reason behind is that algorithm CTC assumes that all query vertices belong to the same community, which makes the accuracy of the algorithm drop by increasing the number of query vertices that are likely to be in different communities.

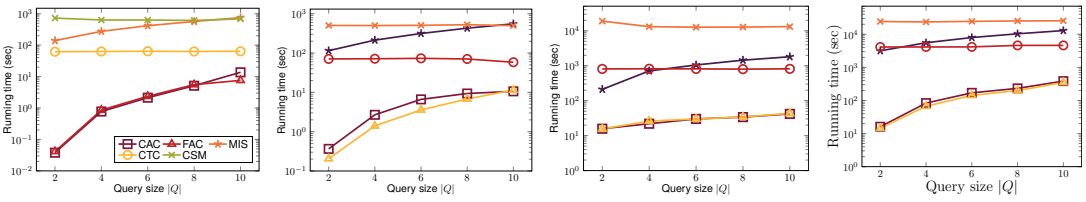
In Fig. 7.4d, the accuracy of the results of each algorithm is presented in terms of F1-measure for dataset LiveJournal, which shows that algorithms CAC and FAC outperform all other algorithms by at least 100% across different query sizes. For queries with more vertices, the gap between accuracy of the results of FAC and CTC is becom-



(a) Amazon (b) DBLP (c) Youtube (d) LiveJournal

(e) Amazon (f) DBLP (g) Youtube (h) LiveJournal

Figure 7.4: F1-measure and F2-measure of different algorithms on random queries of various sizes.

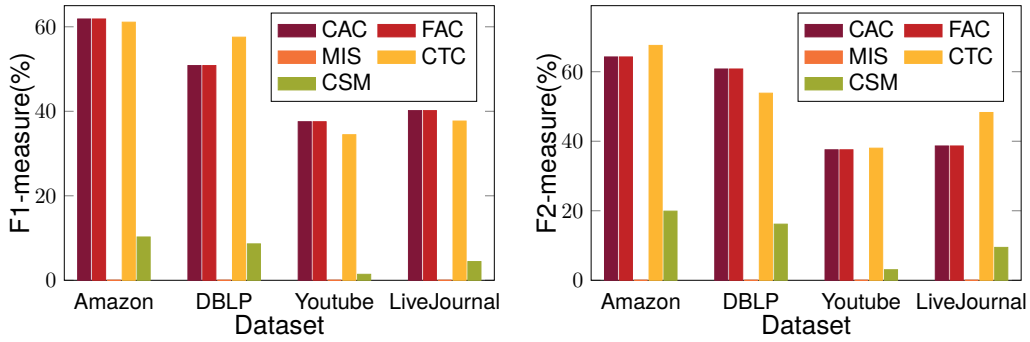


(a) Amazon (b) DBLP (c) Youtube (d) LiveJournal

Figure 7.5: Running time of different algorithms on random queries of various sizes.

ing larger, which means that algorithm FAC significantly outperforms algorithm CTC even in queries with more vertices. Fig. 7.4h illustrates the accuracy of solutions in terms of F2-measure. In Fig. 7.4h, it can be seen that the performance of algorithms CAC and FAC is at least 100% higher than those of algorithms CTC and CSM.

Quality evaluations for queries of size 1. We now evaluate the quality of the communities detected by the proposed algorithms using queries that consist of only one vertex. Notice that queries are selected uniformly at random and the performance is reported in terms of F1-measure and F2-measure. Fig. 7.6 presents the quality of communities detected when the query size is 1. It is also noteworthy that the performance of both algorithms CAC and FAC is identical when queries consist of only one vertex, because the difference between these two algorithms is in determining community membership for queries that consist of more than one vertex. It can be seen in Fig. 7.6a that in terms of F1-measure, the quality of communities detected by algorithms CAC and FAC is higher than other algorithms, i.e. MIS, CTC and CSM for datasets Amazon and Youtube and LiveJournal. However, for dataset DBLP, the performance of algorithm CTC is around 10% higher than that of algorithms CAC and FAC. The reason is that algorithm CTC identifies a large community that contains the single query vertex, which is a better match with the community structure of DBLP. Fig. 7.6b shows that



(a) F1-measure (b) F2-measure

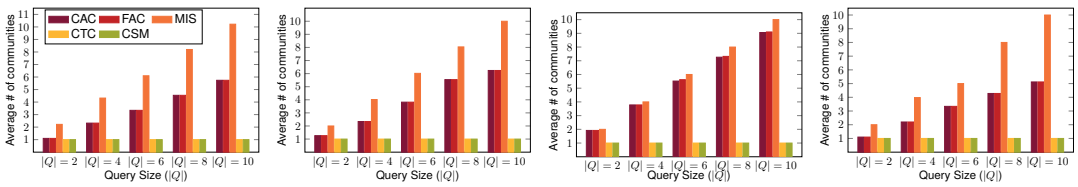
Figure 7.6: The quality of communities detected for queries of size 1 using different datasets.

for dataset DBLP, the quality of communities detected by algorithms CAC and FAC is at least 10% higher than that of other algorithms. However, for dataset Youtube, the performance of algorithms CAC, FAC and CTC is almost identical. Fig. 7.6b illustrates that for dataset Amazon and LiveJournal, the quality of communities detected by algorithm CTC is larger than that of algorithms CAC and FAC. It is noteworthy that the performance gap between F1-measure and F2-measure accuracy of communities identified by CTC clearly highlights that the main reason that the algorithm CTC achieved a higher accuracy is because its identified communities are larger, leading to a better recall value and consequently a better F2-measure accuracy, but not better F1-measure accuracy. In general, the performance of algorithm CSM is consistently inferior to the other algorithms, while algorithms CAC and FAC have similar performances on these datasets for queries of size 1. It must be noted that the F-measure values obtained by algorithm MIS are very small and barely noticeable in Fig. 7.6.

7.4.3 Time efficiency

Table 7.2 illustrates the running times of all different algorithms for different datasets. Since algorithms CTC and CSM have an indexing phase, we separate the times required for indexing and community searching, and present the running times for all queries of size 2, to make a fair comparison.

In Table 7.2, it is noticed that for all datasets, the running time of algorithm CAC is no larger than a fraction 1/50 of the running time algorithm CTC and no larger than 1/10 of the running time of algorithm MIS, while algorithm CAC has a more significant improvement over algorithm CSM, which is nearly 1/900 of the running time of algorithm CSM. This performance is in the light of the accuracy of the solution delivered by algorithm CAC is noticeably better than those of algorithms CTC, CSM and MIS, such running time improvement is still astonishing. It is also noticeable that algorithm FAC has up to 30% improvement over the running time of algorithm CAC. The reason for such significant performance of algorithms CAC and FAC is the efficiency of these algorithms in identifying the top-k shortest paths, while other algorithms spend a large



(a) Amazon (b) DBLP (c) Youtube (d) LiveJournal

Figure 7.7: Average number of communities detected for each queries of various sizes.

amount of time on searching for communities, despite having stored search indexes for communities.

Table 7.2 also shows the running time of different algorithms for dataset DBLP, from which it is observed that the running times of algorithms `CAC` and `FAC` are still significantly better than those of algorithms `CTC`, `CSM` and `MIS`. It is observed in Table 7.2 that the running time of algorithms `CAC` and `FAC` is less than 1/100 of the running time of algorithm `CTC`, while the running time of algorithm `FAC` is less than 1/1000 of the running time of algorithm `CTC` and 1/300 of the running time of algorithm `MIS`.

Table 7.2 shows that the running times of algorithms `CAC` and `FAC` are much smaller that of algorithms `CTC` and `CSM` for dataset LiveJournal, too. While the running time of algorithm `FAC` is less than 1/190 of the running time of algorithm `CTC` and less than 1/900 of the running time of algorithm `CSM`, it is less than 1/13 of the running time of algorithm `MIS`. In Table 7.2, it is noticed that the indexing time for algorithms `CAC` and `FAC` is zero, while other algorithms spend a huge amount of time on indexing. Nonetheless, in the presence of indexes for algorithms `CTC` and `CSM`, the running times of algorithms `CAC` and `FAC` are still better than those of algorithms `CTC` and `CSM`.

Fig. 7.5 also presents the running time of different algorithms using queries of different sizes. It is noticed that the running time of algorithm `CAC` increases with the growth of the query size. A similar trend is observed in Fig. 7.5 for algorithm `FAC`. The reason behind is that algorithms `CAC` and `FAC` spend time in finding community membership for different pairs of query vertices. By comparing plots in Fig. 7.5 and in Fig. 7.4, it can be seen that algorithms `CAC` and `FAC` have increasing trends in running time, while the plots of accuracy seem to be stable with minor fluctuations. It is noted that minor fluctuations are due to randomness of queries. Unlike algorithms `CAC` and `FAC`, the accuracy of the solution delivered by algorithm `CTC` reduces with the increase of the query size, while its running time is almost flat across query sizes.

7.4.4 Number of detected communities

Fig. 7.7 reports the average number of detected communities by algorithms `CAC` and `FAC` for all queries of different sizes, compared to the number of ground-truth communities for each query size. It can be seen from Fig. 7.7a that for dataset Amazon, the average number of detected communities by both algorithms `CAC` and `FAC` is

slightly higher than the number of ground-truth communities. This means that the number of detected communities by algorithms CAC and FAC can be more than one. The gap between the number of communities found by CAC and FAC and the number of ground-truth communities suggests that the propinquity measure may have some false-negative predictions in community membership. Similarly, for dataset DBLP, the average number of detected communities has almost 75% similarity to the average number of communities in the ground-truth. However, for dataset Youtube, the similarity between number of detected communities and number of ground-truth communities is slightly less obvious, but the overall number of communities is more than one in most cases. The reason behind is that the parameters selected for the experiments should be tuned for dataset Youtube separately. For dataset LiveJournal, the average number of communities detected for each query is again almost identical to the average number of communities in the ground-truth. Note that algorithms CTC and CSM find only one community, while the quality of the communities detected by CAC and FAC are significantly higher than those of algorithms CTC and CSM. It is noticed that algorithm MIS consistently finds a large number of communities per search, which is far more than the number of communities in the ground-truth. This indicates that algorithm MIS tends to find approximately one community per vertex in the query.

7.4.5 Impact of parameters ℓ and k

We discuss the accuracy of the proposed propinquity measure in determining if query vertices belong to the same community and the impact of parameters ℓ and k on the accuracy and quality of communities detected by the algorithms. We randomly select pairs of vertices and check if the value of propinquity between them is no smaller than a given number k, and vary the values of these parameters ℓ and k for performance evaluations. Fig. 7.8 plots the accuracy of propinquity for random query vertices with varying values of parameters ℓ and k, where each column is associated with one value for ℓ in different datasets.

Fig. 7.8 shows the trend of precision and recall, when varying the value of ℓ ($5 \leq \ell \leq 14$), and the value of k ($1 \leq k \leq 4$) for different datasets. For the dataset Amazon, It can be seen in the first row of Fig. 7.8 that as we increase the value of k, the precision is increased. The reason behind is that when we increase the value of parameter k, the algorithms become stricter for adding vertices to the same community. Therefore, the precision increases as the value of k increases. However, with the increase on the value of k, the recall decays. The reason is that when the community membership becomes stricter, the number of false-negative results increases and we observe a decreasing trend of recall. Similarly in the second row of Fig. 7.8 for dataset DBLP, the precision value increases, with the increase of the required number of top-k shortest paths for membership, i.e. k, while the recall gets decreased. The third row of Fig. 7.8 illustrates the results for dataset Youtube, where the precision value gets increased, when the required number of paths for membership, i.e. k, is increased, while the recall gets decreased. The last row in Fig. 7.8 shows the results for dataset LiveJournal, where

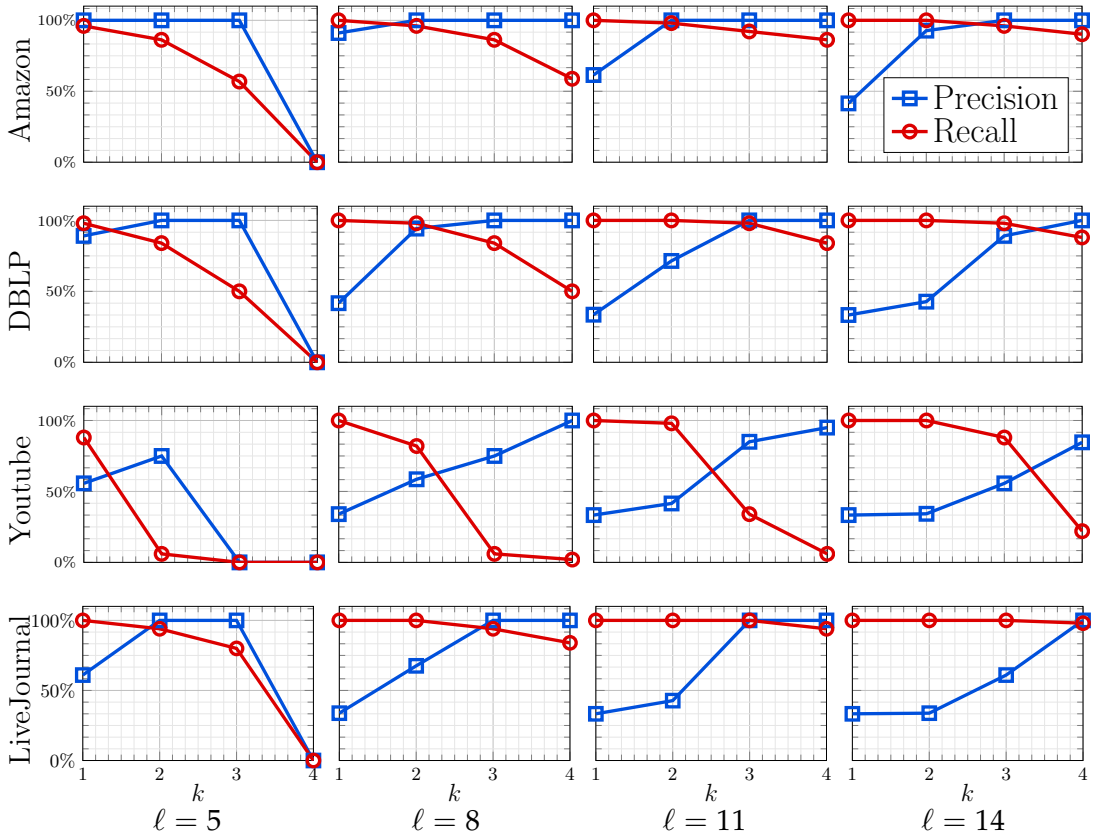


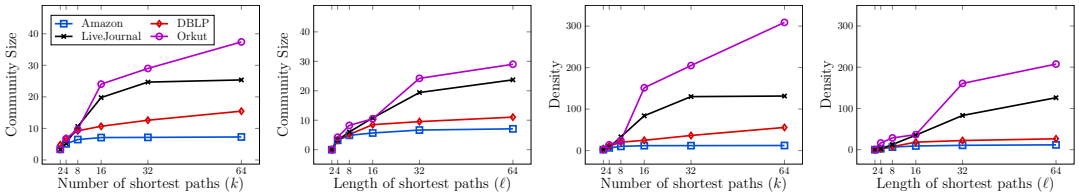
Figure 7.8: Impact of parameters on community membership detection. In each row, precision and recall of the community membership for each dataset is shown, using different values of ℓ and k.

the trend is similar to the one for Amazon, DBLP and Youtube.

The ideal point for choosing proper values for k and ℓ is the position where the line of precision and recall meet each other at their highest value. For example, for dataset Amazon, the perfect point is $\{\ell = 11, k = 2\}$, for datasets DBLP and LiveJournal, the perfect point is $\{\ell = 11, k = 3\}$ and for dataset Youtube, the perfect point is $\{\ell = 14, k = 3\}$. It is also noted that in each dataset, as we increase the value of ℓ, the value of k for which precision and recall meet is also increased. The reason is that when we increase the value of ℓ, the threshold for considering vertices in the same community needs to be stricter, i.e. the value of k needs to be increased.

We now evaluate the impact of parameters on the quality of the communities detected by the proposed algorithms in terms of size and density. We randomly generate 50 sample queries each consisting of 2 vertices and report the average size of communities ($|C|$) and density of edges in the communities ($E[C]/|C|$) by varying parameters k and ℓ in Fig. 7.9. Since the query consists of only two vertices, both algorithms `CAC` and `FAC` have an identical performance. Therefore, in this section, we discuss the performance of both algorithms under the name of `FAC`.

Fig. 7.9a illustrates the average size of communities detected for different datasets



(a) (b) (c) (d)

Figure 7.9: Average size and density of communities detected based on parameters k and ℓ.

by varying parameter k. It can be seen in Fig. 7.9a that as we increase the value of parameter k, the size of communities increases, for all four datasets. The reason is that by increasing the parameter k, more shortest paths between query vertices will be discovered and more vertices will be added to communities. Similarly, Fig. 7.9b shows that parameter ℓ has a similar effect, that is the average size of communities increases as we increase the value of ℓ. The reason is that as we increase the value of parameter ℓ, the length of the shortest paths will be increased. Consequently more vertices will be added to each community and the size of communities increases.

Fig. 7.9c shows the average density of communities detected for different datasets by varying parameter k. It can be seen that by increasing the value of parameter k, the size of communities increases and as a result, the number of edge between them increases. Fig. 7.9d illustrates that parameter ℓ exhibits a similar characteristic, by which the density of edges inside communities increases. The reason behind is that the size of communities increases, as we increase the value of parameter ℓ, thereby more edges between communities get revealed.

7.5 Summary

We studied the community search problem for a given query of vertices, where the vertices in the query may belong to different communities. We first defined the notion of propinquity in a network that represents the closeness between a pair of vertices and the likelihood that they lie in the same community. We then proposed a generic problem definition for community search based on the propinquity measure and utilized the propinquity measure to determine which query vertices belong to the same community. We made an interesting observation about the community structure in networks, that is vertices on the shortest path between two query vertices are most likely to be in the same community. Using this intuition, we devised an instance of the propinquity measure that captures the cohesiveness of a pair of vertices. We further devised two efficient algorithms for the community search problem that are capable of finding more than one communities for a given query. We finally conducted experiments on real-world datasets and compared our results with several benchmark algorithms for community search. Our experimental results showed that the proposed algorithm delivers communities that accurately match with ground-truth communities, while its running time is only a fraction of the running time of the benchmark algorithms.

Chapter 8

Conclusion and future works

In this thesis, we studied some of the problems that are related to community structure in complex networks, such as hierarchical and overlapping community detection, community search and structural hole spanners. While the existing algorithms for these problems mainly suffer from scalability and accuracy, we showed that the shortest path in a network can be used to devise accurate, yet scalable algorithms for a variety of problems related to community structure. In particular, we solved several problems including structural hole spanners, hierarchical community detection, community search and overlapping community detection in large-scale complex networks. We utilised the shortest path between communities as a measure to determine community membership and importance of a vertex in the communication between communities. In the following we summarise our conclusions for each of these problems.

8.1 Hierarchical community detection

In Chapter 4, we studied the hierarchical community detection problem in large-scale complex networks. We proposed the notion of a cohesive hierarchy of communities, as a rooted tree of communities where each community is a subset of its parent in the tree, and the information centralities of communities is no less than that of their parent in the hierarchical tree. We then formally defined the problem of cohesive hierarchy detection, as the problem of identifying a cohesive hierarchy of communities with maximum information centrality, and we showed that the problem of finding hierarchical communities is NP-hard. We devised two efficient and scalable heuristic algorithms for this problem, which use a sparsification method to reduce the network size for finding global cuts. We also proposed a fast randomized algorithm to estimate the value of information centrality in large-scale networks. We finally validated the effectiveness of our proposed algorithms using extensive experiments using five large-scale real-world datasets.

The research conducted on hierarchical communities can be used to explore the relationships among users in real-world social networks and biological networks. One of the key problems to solve in social network analysis is the study of hierarchies of relationships among users, where each relationship in lower levels of the hierarchy

can be given a higher weight, compared to the edges in higher levels of the hierarchy. Other instances where our findings on hierarchical community detection can contribute to include hierarchical clustering in data mining and finding the connectivity patterns in the network of organisms.

8.2 Structural hole spanners

In Chapter 5, we studied the top-k structural hole spanner problem in a large-scale complex network and proposed a novel model to measure the quality of structural hole spanners. We then formulated a novel top-k structural hole spanner problem and showed its NP-hardness. We thirdly devised two fast yet scalable linear-time algorithms for the problem by exploring the bounded inverse closeness centrality of vertices and articulation points in the network. We finally validated the effectiveness of the proposed model and evaluated the performance of the proposed algorithms through extensive experiments on real and synthetic datasets. Our experimental results demonstrated that the proposed model can capture the characteristics of structural hole spanners accurately, and the proposed algorithms are promising.

The result of the research on structural hole spanners can contribute to several applications and research areas. One of the interesting areas to explore and apply our findings of structural hole spanners is the disease propagation control, where identifying the top-k structural hole spanners in a network can disconnect communities and stop the spread of disease to a wider span. Our findings of structural hole spanners can be applied to identify multi-disciplinary researchers, who are active in bridging ideas from theorists to practitioners in different research areas to solve a wide range of problems. Similarly, influential papers in the citation network of publications can be found by structural hole spanners that make impact in several research areas.

8.3 Overlapping community detection

In Chapter 6, we designed an efficient algorithm for the problem of overlapping communities detection from a complex network. We first proposed a novel community fitness metric that can model the quality of detected communities more accurately. We then devised an efficient yet scalable algorithm for detecting overlapping communities, using the proposed community fitness metric. We finally validated the effectiveness of the fitness metric, and evaluated the efficiency of the proposed algorithm by conducting extensive experiments on real datasets with considerable network sizes. The experimental results show that the proposed algorithm outperforms the state-of-the-arts and runs faster, while it finds higher quality overlapping communities.

The proposed overlapping community detection algorithm in this thesis can be used to study the overlap between communities. These communities can be then used in several applications including marketing, influence maximization and finding webpages with high content commonality. It is noted that communities in real networks usually overlap with each other. Our approach can also be used as an ad-

dition to the hierarchical community detection algorithm and the community search algorithm to enable those works to detect the overlap between communities.

8.4 Community search

In Chapter 7, we investigated the problem of community search for a given query of vertices. We first discussed one of the main properties of complex networks that is vertices that lie on the shortest path between two query vertices are most likely to be in the same community as them. We further analysed the community structure and formulated the problem of community search, as the problem of finding top-k shortest paths between vertices that are relevant, i.e. have many paths with short length between them. We then devised an efficient algorithm for the problem and conducted several experiments on real-world datasets and compared our results with the state-of-the-art algorithms for community search problem. Our experimental results showed that the proposed algorithm delivers communities that accurately match with ground-truth communities, while the running time of the proposed algorithm is reasonably fast compared to the existing ones.

The community search algorithm described in this thesis can be used to solve several real-world problems. One of the instances where our community search algorithm can contribute to is the expansion of communities around seed vertices, which is useful in finding global communities of the network. Furthermore, our community search algorithm can be useful in detection of key players in the relationship between huge organisations and groups. Finding the affiliation between individuals in social networks and revealing hidden communication patterns among users is another major contributors of our community search algorithm.

8.5 Concluding remarks

In this thesis, we showed that shortest path can be used in different ways to solve a large span of problems that deal with large-scale networks. We showed that the top-k structural hole spanners can be seen as the top-k vertices that their removal can incur the largest increase in the information centrality of a network. Furthermore, we shows that the information centrality, i.e. mean length of shortest paths, in communities gets increased, as we move towards the leaves of a hierarchical structure of communities. Similarly, the top-k shortest paths between query vertices can be used to determine whether two query vertices are in the same community. Furthermore, the top-k shortest paths provide an approach to expand communities of a given set of query vertices in a network. Last but not least, when the length of the second shortest path between two adjacent vertices is two, the top-k shortest paths (same as the number of triangles formed by the edge between them) can be used to find overlapping communities. Our findings in this thesis suggest several directions of research for future researchers to pursue.

One of the important directions for future research is studying weighted networks.

The main approach presented in this thesis is specific to undirected and unweighted networks, though it is possible to extend this approach to work on weighted networks using a simple extension of the unweighted shortest path algorithm (such as BFS) to the weighted shortest path algorithms (such as Dijkstra and Floyd-Warshall). In order to find edge-cuts in a weighted network to find the hierarchy of communities, it is possible to adopt the weighted version of the maximum adjacency algorithm [Stoer and Wagner 1994]. Therefore, one might be interested in extending the proposed approaches in this thesis and examining the ideas on weighted networks.

Another direction to explore in future research is the parallel computing approach in community detection. The algorithms presented in this thesis are sequential algorithms, while the presented algorithms including structural hole spanners detection algorithm, hierarchical and overlapping community detection and community search could also be parallelized. Conducting extensive experiments to determine the effectiveness of parallelization on the efficiency of the proposed algorithms in this thesis is another interesting direction for future research.

The hierarchical community detection algorithm that was presented in this thesis can be extended to find overlapping hierarchical communities. The cut-based approach that has been adopted in hierarchical community detection partitions a network into disjoint communities. A further study could assess the impact of community expansion based on the overlapping community detection fitness metric to detect the overlapping communities in large-scale networks.

Throughout this thesis, we studied several problems related to community detection, including structural hole spanners, hierarchical and overlapping community detection and community search, and proposed efficient algorithms that were scalable to networks with hundreds of millions of vertices using a single desktop computer. We showed that the shortest paths between vertices can provide a useful, yet efficient way to identify the underlying structure of communities in large-scale networks.

Appendix

Lemma 11. *Given a graph $G = (V \cup \{s,t\}, E)$, a positive integer k, and the constructed graph $G' = (V \cup S \cup T, E')$ from G, assume that the $s-t$ vertex connectivity in G is at least $k+1$ and graph G does not contain edge (s,t), where $|S| = |T| = l = 4n^6$ and $n = |V \cup \{s,t\}|$. For each subset V_S of $V \cup S \cup T$ with $|V_S| = k$, if $V_S \cap S \neq \emptyset$ or $V_S \cap T \neq \emptyset$, then there is a subset V_S^* of V with $|V_S^*| = k$ such that $D(G' \setminus V_S^*) > D(G' \setminus V_S)$.*

Proof. We only consider the case that $V_S \cap S \neq \emptyset$, since the proofs for the cases of $V_S \cap T \neq \emptyset$ is similar.

In the following, we show that we can construct a subset V_S' of $V \cup S \cup T$ from V_S with $|V_S'| = k$, by replacing any vertex $s_i \in V_S \cap S$ with a special vertex $w \in V \setminus V_S$, i.e., $V_S' = (V_S \setminus \{s_i\}) \cup \{w\}$, so that $D(G' \setminus V_S') > D(G' \setminus V_S)$. Then, we can find a subset V_S^* of V with $|V_S^*| = k$ so that $D(G' \setminus V_S^*) > D(G' \setminus V_S)$, by repeatedly performing such a construction until there is a set V_S^* containing no vertices in both S and T. The rest is to show how to find vertex w.

Denote by $\kappa^G(u,v)$ the $u-v$ vertex connectivity of any two vertices u and v in graph G. Given any pair of vertices $s_j \in S \setminus V_S$ and $t_j \in T \setminus V_S$, following the construction of G', the $s_j - t_j$ vertex connectivity in graph G' is at least $k+1$ too, i.e., $\kappa^{G'}(s_j, t_j) \geq k+1$, since the $s-t$ vertex connectivity in G is no less than $k+1$. Thus, the $s_j - t_j$ vertex connectivity in $G' \setminus V_S$ is no less than $\kappa^{G'}(s_j, t_j) - |V_S| \geq k+1-k = 1$.

As the $s_j - t_j$ vertex connectivity in graph $G' \setminus V_S$ is at least 1, there is a shortest path $P_{s_j t_j}$ between s_j and t_j in graph $G' \setminus V_S$. Assume that path $P_{s_j t_j}$ starts from vertex s_j and ends at vertex t_j. Let w be the second vertex on path $P_{s_j t_j}$ from left to right. We can see that w is in the vertex set $V \setminus V_S$. Otherwise, w is in $(S \setminus V_S) \setminus \{s_j\}$ or $T \setminus V_S$. This implies that there is an edge between two different vertices in S, or one vertex in S and another vertex in T, which contradicts the construction of graph G' or the assumption that there is no edge connecting vertices s and t in G. Following the construction of graph G', we know that vertex w is adjacent to every vertex $s_i \in S$ in graph G', and $d_{s_i w}^{G'} = 1$. Furthermore, the shortest path between any two different vertices s_i and s_j in S is $s_i - w - s_j$. Then, $d_{s_i, s_j}^{G'} = 2$. Similarly, $d_{t_i t_j}^{G'} = 2$, where $t_i, t_j \in T$ and $i \neq j$.

We then show that $D(G' \setminus V_S') > D(G' \setminus V_S)$ as follows. Let $n_S = |V_S \cap S|$ and

$n_T = |V_S \cap T|$. Also, let $V' = V \cup S \cup T$ be the set of vertices in graph G', and $V_c = V' \setminus (V_s \cup \{w\})$. Then,

$$V' \setminus V_S = V_c \cup \{w\}, \tag{A.1}$$

and

$$\begin{aligned} V' \setminus V'_S &= V' \setminus ((V_S \setminus \{s_i\}) \cup \{w\}) \\ &= (V' \setminus (V_S \cup \{w\})) \cup \{s_i\} \\ &= V_c \cup \{s_i\}. \end{aligned}$$

We now calculate $D(G' \setminus V_S)$.

$$D(G' \setminus V_S) = \sum_{u,v \in V' \setminus V_S} d_{uv}^{G' \setminus V_S},$$

following Eq. (A.1),

$$\begin{aligned} D(G' \setminus V_S) &= \sum_{u,v \in V_c \cup \{w\}} d_{uv}^{G' \setminus V_S} \\ &= \sum_{u,v \in V_c} d_{uv}^{G' \setminus V_S} + 2 \sum_{v \in V_c} d_{wv}^{G' \setminus V_S} \\ &= \sum_{u,v \in V_c} d_{uv}^{G' \setminus V_S} + 2 \left(\sum_{s_j \in V_c \cap S} d_{ws_j}^{G' \setminus V_S} + \sum_{v \in V_c \cap V} d_{wv}^{G' \setminus V_S} + \sum_{t_j \in V_c \cap T} d_{wt_j}^{G' \setminus V_S} \right) \end{aligned}$$

since $d_{ws_j}^{G' \setminus V_S} = 1$ and $|V_c \cap S| = l - n_S$,

$$D(G' \setminus V_S) = \sum_{u,v \in V_c} d_{uv}^{G' \setminus V_S} + 2 \left((l - n_S) + \sum_{v \in V_c \cap V} d_{wv}^{G' \setminus V_S} + \sum_{t_j \in V_c \cap T} d_{wt_j}^{G' \setminus V_S} \right).$$

The value of $D(G' \setminus V'_S)$ can be calculated similarly, which is described as follows.

$$D(G' \setminus V'_S) = \sum_{u,v \in V' \setminus V'_S} d_{uv}^{G' \setminus V'_S},$$

following Eq. (A.1),

$$\begin{aligned} D(G' \setminus V'_S) &= \sum_{u,v \in V_c \cup \{s_i\}} d_{uv}^{G' \setminus V'_S} \\ &= \sum_{u,v \in V_c} d_{uv}^{G' \setminus V'_S} + 2 \left(\sum_{s_j \in V_c \cap S} d_{s_i s_j}^{G' \setminus V'_S} + \sum_{v \in V_c \cap V} d_{s_i v}^{G' \setminus V'_S} + \sum_{t_j \in V_c \cap T} d_{s_i t_j}^{G' \setminus V'_S} \right) \end{aligned}$$

since $d_{s_i s_j}^{G' \setminus V'_S} = 2$ and $|V_c \cap S| = l - n_S$,

$$D(G' \setminus V'_S) = \sum_{u,v \in V_c} d_{uv}^{G' \setminus V'_S} + 2 \left(2(l - n_S) + \sum_{v \in V_c \cap V} d_{s_i v}^{G' \setminus V'_S} + \sum_{t_j \in V_c \cap T} d_{s_i t_j}^{G' \setminus V'_S} \right).$$

In the following we calculate the difference between $D(G' \setminus V'_S)$ and $D(G' \setminus V_S)$,

$$\begin{aligned} D(G' \setminus V'_S) - D(G' \setminus V_S) & \\ &= \sum_{u,v \in V_c} (d_{uv}^{G' \setminus V'_S} - d_{uv}^{G' \setminus V_S}) + 2(l - n_S) \\ &+ 2 \sum_{v \in V_c \cap V} \left(d_{s_i v}^{G' \setminus V'_S} - d_{wv}^{G' \setminus V_S} \right) + 2 \sum_{t_j \in V_c \cap T} \left(d_{s_i t_j}^{G' \setminus V'_S} - d_{w t_j}^{G' \setminus V_S} \right), \end{aligned}$$

we note that $G' \setminus V_S = G'[V' \setminus V_S] = G'[V_c \cup \{w\}]$ and $G' \setminus V'_S = G'[V' \setminus V'_S] = G'[V_c \cup \{s_i\}]$. Consider graph $G'[V_c \cup \{w, s_i\}]$. On one hand, the removal of vertex s_i from graph $G'[V_c \cup \{w, s_i\}]$ does not increase the distance between vertices u and v in the resulting graph, i.e., $d_{uv}^{G'[V_c \cup \{w\}]} = d_{uv}^{G'[V_c \cup \{w, s_i\}]}$, since if s_i is contained in a $u - v$ shortest path in graph $G'[V_c \cup \{w, s_i\}]$, we can construct another $u - v$ shortest path in graph $G'[V_c \cup \{w, s_i\}]$, which does not contain s_i, by replacing s_i with any vertex $s_j \in S \setminus V_S$. On the other hand, if w is included in a $u - v$ shortest path in graph $G'[V_c \cup \{w, s_i\}]$, then $d_{uv}^{G'[V_c \cup \{s_i\}]} \geq d_{uv}^{G'[V_c \cup \{w, s_i\}]}$. Therefore, $d_{uv}^{G' \setminus V'_S} \geq d_{uv}^{G'[V_c \cup \{w, s_i\}]} = d_{uv}^{G' \setminus V_S}$,

$$\begin{aligned} D(G' \setminus V'_S) - D(G' \setminus V_S) \geq 0 + 2(l - n_S) + 2 \sum_{v \in V_c \cap V} \left(d_{s_i v}^{G' \setminus V'_S} - d_{wv}^{G' \setminus V_S} \right) \\ + 2 \sum_{t_j \in V_c \cap T} \left(d_{s_i t_j}^{G' \setminus V'_S} - d_{w t_j}^{G' \setminus V_S} \right), \end{aligned}$$

we now observe that for each vertex $s_j \in V_c \cap S$, we have $d_{s_j v}^{G' \setminus V'_S} = d_{s_i v}^{G' \setminus V'_S}$. Since $s_j, v \in V_c$, following $d_{uv}^{G' \setminus V'_S} \geq d_{uv}^{G' \setminus V_S}$, we have $d_{s_j v}^{G' \setminus V'_S} \geq d_{s_j v}^{G' \setminus V_S}$. On the other hand, following the definition of vertex w, graph $G' \setminus V_S$ contains edge (w, s_j). Then, given a shortest path $P_{s_j v}$ between s_j and v in graph $G' \setminus V_S$, we can construct another path P_{wv} between w and v in graph $G' \setminus V_S$, by concatenating edge (w, s_j) and path $P_{s_j v}$, i.e., $P_{wv} = (w, s_j) P_{s_j v}$. We thus have $d_{wv}^{G' \setminus V_S} \leq d_{s_j v}^{G' \setminus V_S} + 1$. Therefore, $d_{s_j v}^{G' \setminus V'_S} \geq d_{s_j v}^{G' \setminus V_S} \geq d_{wv}^{G' \setminus V_S} - 1$,

$$D(G' \setminus V'_S) - D(G' \setminus V_S) \geq 2(l - n_S) + 2|V_c \cap V| \cdot (-1) + 2 \sum_{t_j \in V_c \cap T} \left(d_{s_i t_j}^{G' \setminus V'_S} - d_{w t_j}^{G' \setminus V_S} \right)$$

we see that $d_{s_i t_j}^{G' \setminus V'_S} \geq d_{w t_j}^{G' \setminus V_S} + 1$ for any vertex $t_j \in V_c \cap T$. Following the definition of vertex w, there is a shortest path $P_{s_j t_j}$ between s_j and t_j in graph $G' \setminus V_S$, which starts from s_j and ends at t_j, so that vertex w is the second vertex in path $P_{s_j t_j}$. Then, we can

construct another path P_{w,t_j} between w and t_j, by removing vertex s_j and edge (s_j, w) from path $P_{s_jt_j}$. We can see that path P_{w,t_j} is a shortest path between w and t_j. Then, $d_{s_jt_j}^{G'\setminus V_S} = d_{wt_j}^{G'\setminus V_S} + 1$. Since $d_{s_jt_j}^{G'\setminus V'_S} \geq d_{s_jt_j}^{G'\setminus V_S}$, therefore $d_{s_jt_j}^{G'\setminus V'_S} \geq d_{wt_j}^{G'\setminus V_S} + 1$,

$$\begin{aligned} D(G'\setminus V'_S) - D(G'\setminus V_S) &\geq 2(l-n_S) - 2|V_c \cap V| + 2(l-n_T) \\ &> 2(2l - n_S - n_T - n), \text{ as } |V_c \cap V| \leq |V| = n-2 < n \\ &> 2(2l-2n), \text{ as } n_S + n_T \leq k < n \\ &\geq 0, \text{ as } l = 4n^6 \geq n, \end{aligned}$$

i.e., $D(G'\setminus V'_S) > D(G'\setminus V_S)$. The lemma then follows. □

Lemma 12. *Given a graph $G = (V \cup \{s,t\}, E)$, a positive integer k, and the constructed graph $G' = (V \cup S \cup T, E')$ from G, assume that the $s-t$ vertex connectivity in graph G is at least $k+1$, where $|S| = |T| = l = 4n^6$ and $n = |V \cup \{s,t\}|$. Let $n_V = |V| = n-2$. For each subset V_S of V with $|V_S| = k$, we have*

$$D(G'\setminus V_S) \geq 4l(n_V - k) + 4l(l-1) + (n_V - k)(n_V - k - 1) + 2l^2 d_{s_jt_j}^{G'\setminus V_S}, \quad \text{(A.2)}$$

and

$$D(G'\setminus V_S) \leq 4l(n_V - k)\zeta + 4l(l-1) + (n_V - k)(n_V - k - 1)\zeta + 2l^2 d_{s_jt_j}^{G'\setminus V_S}, \quad \text{(A.3)}$$

where $s_j \in S$ and $t_j \in T$.

Proof. Since $V' = S \cup V \cup T$ and $V_S \subset V$, we calculate $D(G'\setminus V_S)$ as follows.

$$\begin{aligned} D(G'\setminus V_S) &= \sum_{u,v\in(S\cup V\cup T)\setminus V_S} d_{uv}^{G'\setminus V_S} \\ &= \sum_{s_i,s_j\in S} d_{s_is_j}^{G'\setminus V_S} + 2\sum_{s_i\in S, v\in V\setminus V_S} d_{s_iv}^{G'\setminus V_S} + 2\sum_{s_i\in S, t_j\in T} d_{s_it_j}^{G'\setminus V_S} \\ &+ \sum_{u,v\in V\setminus V_S} d_{uv}^{G'\setminus V_S} + 2\sum_{v\in V\setminus V_S, t_j\in T} d_{vt_j}^{G'\setminus V_S} + \sum_{t_i,t_j\in T} d_{t_it_j}^{G'\setminus V_S} \end{aligned}$$

since $d_{s_is_j}^{G'\setminus V_S} = d_{t_it_j}^{G'\setminus V_S} = 2$ and $|S| = |T| = l$,

$$\begin{aligned} D(G'\setminus V_S) &= 2l(l-1) + 2\sum_{s_i\in S, v\in V\setminus V_S} d_{s_iv}^{G'\setminus V_S} + 2\sum_{s_i\in S, t_j\in T} d_{s_it_j}^{G'\setminus V_S} \\ &+ \sum_{u,v\in V\setminus V_S} d_{uv}^{G'\setminus V_S} + 2\sum_{v\in V\setminus V_S, t_j\in T} d_{vt_j}^{G'\setminus V_S} + 2l(l-1) \end{aligned}$$

and since $d_{s_iv}^{G'\setminus V_S} = d_{s_jv}^{G'\setminus V_S}$ holds for any two vertices $s_i, s_j \in S$ and a vertex $v \in (V \setminus$

$V_S) \cup T$, $d_{vt_i}^{G'\setminus V_S} = d_{vt_j}^{G'\setminus V_S}$ for any two vertices $t_i, t_j \in T$ and a vertex $v \in (V \setminus V_S) \cup S$,

$$\begin{aligned} D(G' \setminus V_S) &= 2l(l-1) + 2l \sum_{v \in V \setminus V_S} d_{s_i v}^{G'\setminus V_S} + 2l^2 d_{s_i t_j}^{G'\setminus V_S} \\ &+ \sum_{u,v \in V \setminus V_S} d_{uv}^{G'\setminus V_S} + 2l \sum_{v \in V \setminus V_S} d_{vt_j}^{G'\setminus V_S} + 2l(l-1), \end{aligned} \tag{A.4}$$

We can see that $1 \leq d_{s_i v}^{G'\setminus V_S}, d_{vt_i}^{G'\setminus V_S}, d_{uv}^{G'\setminus V_S} \leq \zeta$. Then, both inequality (A.2) andinequality (A.3) hold when we substitute 1 and ζ for $d_{s_i v}^{G'\setminus V_S}$, $d_{vt_i}^{G'\setminus V_S}$, and $d_{uv}^{G'\setminus V_S}$, respectively. □

Bibliography

AGGARWAL, C. C., WOLF, J. L., AND YU, P. S.-L. 2004. Method for targeted advertising on the web based on accumulated self-learning data, clustering users and semantic node graph techniques. US Patent 6,714,975. (p. 8)

AHN, Y.-Y., BAGROW, J. P., AND LEHMANN, S. 2010. Link communities reveal multiscale complexity in networks. *Nature 466*, 7307, 761–764. (pp. 23, 38)

AHUJA, G. 2000. Collaboration networks, structural holes, and innovation: A longitudinal study. *Administrative science quarterly 45*, 3, 425–455. (p. 22)

AKBAS, E. AND ZHAO, P. 2017. Truss-based community search: a truss-equivalence based indexing approach. *Proc. of VLDB'17 10*, 11, 1298–1309. (p. 10)

ANDERSEN, R. AND LANG, K. J. 2006. Communities from seed sets. In *WWW'06* (New York, NY, USA, 2006), pp. 223–232. ACM. (p. 7)

ARENAS, A., FERNANDEZ, A., AND GOMEZ, S. 2008. Analysis of the structure of complex networks at different resolution levels. *New journal of physics 10*, 5, 053039. (p. 3)

BANDYOPADHYAY, S., CHOWDHARY, G., AND SENGUPTA, D. 2015. Focs: Fast overlapped community search. *TKDE'15 27*, 11, 2974–2985. (pp. 8, 65, 80)

BAR-NOY, A., KHULLER, S., AND SCHIEBER, B. 1998. The complexity of finding most vital arcs and nodes. Technical report. (p. 45)

BARABÁSI, A.-L. AND ALBERT, R. 1999. Emergence of scaling in random networks. *science 286*, 5439, 509–512. (p. 49)

BARBIERI, N., BONCHI, F., GALIMBERTI, E., AND GULLO, F. 2015. Efficient and effective community search. *Data Mining and Knowledge Discovery 29*, 5, 1406–1433. (pp. 9, 10, 24, 25, 99, 100)

BEAUCHAMP, M. A. 1965. An improved index of centrality. *Systems Research and Behavioral Science 10*, 2, 161–163. (pp. 15, 43)

BENSON, A. R., GLEICH, D. F., AND LESKOVEC, J. 2016. Higher-order organization of complex networks. *Science 353*, 6295, 163–166. (p. 66)

BOCCALETTI, S., LATORA, V., MORENO, Y., CHAVEZ, M., AND HWANG, D.-U. 2006. Complex networks: Structure and dynamics. *Physics reports 424*, 4-5, 175–308. (p. 1)

BONDY, J. A., MURTY, U. S. R., ET AL. 1976. *Graph theory with applications*, Volume 290. Citeseer. (p. 1)

BOZORGI, A., HAGHIGHI, H., ZAHEDI, M. S., AND REZVANI, M. 2016. Incim: A community-based algorithm for influence maximization problem under the linear threshold model. *Information Processing & Management 52*, 6, 1188–1199. (p. 4)

BRON, C. AND KERBOSCH, J. 1973. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM 16*, 9, 575–577. (pp. 30, 96, 98)

BUDAK, C., AGRAWAL, D., AND EL ABBADI, A. 2011. Limiting the spread of misinformation in social networks. In *WWW'11* (New York, NY, USA, 2011), pp. 665–674. (p. 7)

BURT, R. S. 1992. *Structural holes: The social structure of competition*. Harvard university press. (pp. 4, 6, 22, 45, 57)

BURT, R. S. 2004. Structural holes and good ideas. *American journal of sociology 110*, 2, 349–399. (p. 22)

BURT, R. S. 2007. Secondhand brokerage: Evidence on the importance of local structure for managers, bankers, and analysts. *Academy of Management Journal 50*, 1, 119–148. (p. 22)

CAI, L., MENG, T., HE, T., CHEN, L., AND DENG, Z. 2017. K-hop community search based on local distance dynamics. In *International Conference on Neural Information Processing* (2017), pp. 24–34. Springer.

CHANG, L., YU, J. X., QIN, L., LIN, X., LIU, C., AND LIANG, W. 2013. Efficiently computing k-edge connected components via graph decomposition. In *SIGMOD'13* (2013), pp. 205–216. (p. 31)

CHEN, J., ZAÏANE, O., AND GOEBEL, R. 2009. Local community identification in social networks. In *Proc. of ASONAM'09* (2009), pp. 237–242. IEEE. (p. 25)

CHENG, J., KE, Y., CHU, S., AND ÖZSU, M. T. 2011. Efficient core decomposition in massive networks. In *Proc. of ICDE'11* (2011), pp. 51–62. IEEE. (p. 6)

CLAUSET, A., NEWMAN, M. E., AND MOORE, C. 2004. Finding community structure in very large networks. *Physical review E 70*, 6, 066111. (p. 38)

COHEN, J. 2008. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 16. (pp. 6, 21, 65, 68, 74, 76)

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., AND STEIN, C. 2001. *Introduction to algorithms*, Volume 6. MIT press Cambridge. (p. 30)

COSCIA, M., ROSSETTI, G., GIANNOTTI, F., AND PEDRESCHI, D. 2012. Demon: a local-first discovery method for overlapping communities. In *Proc. of SIGMOD'12* (2012), pp. 615–623. ACM. (pp. 23, 24, 80)

CUI, W., XIAO, Y., WANG, H., AND WANG, W. 2014. Local search of communities in large graphs. In *Proc. of SIGMOD'14* (2014), pp. 991–1002. ACM. (pp. 9, 10, 24, 25)

DOURISBOURE, Y., GERACI, F., AND PELLEGRINI, M. 2007. Extraction and classification of dense communities in the web. In *Proc. of WWW'07* (2007), pp. 461–470. ACM. (p. 8)

DU, N., WANG, B., AND WU, B. 2008. Overlapping community structure detection in networks. In *Proc. of CIKM'08* (2008), pp. 1371–1372. ACM. (pp. 21, 24)

EPPSTEIN, D. AND WANG, J. 2001. Fast approximation of centrality. In *Proceedings of SODA'01* (2001), pp. 228–229. Society for Industrial and Applied Mathematics. (p. 35)

EUSTACE, J., WANG, X., AND CUI, Y. 2015. Overlapping community detection using neighborhood ratio matrix. *Physica A: Statistical Mechanics and its Applications 421*, 510–521. (p. 23)

FELL, D. A. AND WAGNER, A. 2000. The small world of metabolism. *Nature biotechnology 18*, 11, 1121. (p. 1)

FORTUNATO, S. 2010. Community detection in graphs. *Physics reports 486*, 3, 75–174. (pp. 1, 2, 4, 5, 18, 19, 20, 24, 28)

FORTUNATO, S. AND BARTHELEMY, M. 2007. Resolution limit in community detection. *PNAS'07 104*, 1, 36–41. (pp. 8, 65, 66)

FORTUNATO, S., LATORA, V., AND MARCHIORI, M. 2004. Method to find community structures based on information centrality. *Physical review E 70*, 5, 056104. (pp. 5, 16, 21, 29)

GIRVAN, M. AND NEWMAN, M. E. 2002. Community structure in social and biological networks. *PNAS'02 99*, 12, 7821–7826. (pp. 3, 5, 20, 25, 44)

GLEICH, D. F. AND SESHADHRI, C. 2012. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *Proc. of KDD'12* (2012), pp. 597–605. ACM. (pp. 25, 81)

GOPALAN, P. K. AND BLEI, D. M. 2013. Efficient discovery of overlapping communities in massive networks. *PNAS'13 110*, 36, 14534–14539. (pp. 38, 39, 80, 81)

GOYAL, S. AND VEGA-REDONDO, F. 2007. Structural holes in social networks. *Journal of Economic Theory 137*, 1, 460–492. (pp. 7, 22, 57)

GUILLE, A., HACID, H., FAVRE, C., AND ZIGHED, D. A. 2013. Information diffusion in online social networks: A survey. *SIGMOD'13 42*, 2 (July), 17–28. (p. 7)

HOEFFDING, W. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association 58*, 301, 13–30. (p. 36)

HOPCROFT, J. AND TARJAN, R. 1973. Algorithm 447: efficient algorithms for graph manipulation. *Communications of the ACM 16*, 6, 372–378. (p. 52)

HUANG, X., CHENG, H., LI, R.-H., QIN, L., AND YU, J. X. 2013. Top-k structural diversity search in large networks. *Proceedings of the VLDB Endowment 6*, 13, 1618–1629. (p. 22)

HUANG, X., CHENG, H., QIN, L., TIAN, W., AND YU, J. X. 2014. Querying k-truss community in large and dynamic graphs. In *Proc. of SIGMOD'14* (2014), pp. 1311–1322. ACM. (pp. 9, 10, 24, 99)

HUANG, X., LAKSHMANAN, L. V., YU, J. X., AND CHENG, H. 2015. Approximate closest community search in networks. *VLDB'15 9*, 4, 276–287. (pp. 10, 24, 25, 68, 93, 99, 100, 101)

JONSSON, P. F., CAVANNA, T., ZICHA, D., AND BATES, P. A. 2006. Cluster analysis of networks generated through homology: automatic identification of important protein communities involved in cancer metastasis. *BMC bioinformatics 7*, 1, 2. (p. 3)

KANG, U. AND FALOUTSOS, C. 2011. Beyond 'caveman communities': Hubs and spokes for graph compression and mining. In *ICDM'11* (Washington, DC, USA, 2011), pp. 300–309. (p. 7)

KANNAN, R., VEMPALA, S., AND VETTA, A. 2004. On clusterings: Good, bad and spectral. *J. ACM 51*, 3 (May), 497–515. (pp. 8, 18)

KATOH, N., IBARAKI, T., AND MINE, H. 1982. An efficient algorithm for k shortest simple paths. *Networks 12*, 4, 411–427. (p. 98)

KATZ, L. 1953. A new status index derived from sociometric analysis. *Psychometrika 18*, 1, 39–43. (p. 26)

KEMPE, D., KLEINBERG, J., AND TARDOS, É. 2003. Maximizing the spread of influence through a social network. In *SIGKDD'03* (2003), pp. 137–146. ACM. (p. 7)

KLEINBERG, J., SURI, S., TARDOS, É., AND WEXLER, T. 2008. Strategic network formation with structural holes. In *EC'08* (2008), pp. 284–293. (pp. 7, 22)

KLEINBERG, J. M. 2000. Navigation in a small world. *Nature 406*, 6798, 845. (pp. 2, 49)

KUMAR, R., NOVAK, J., RAGHAVAN, P., AND TOMKINS, A. 2004. Structure and evolution of blogspace. *Communications of the ACM 47*, 12, 35–39. (p. 3)

LANCICHINETTI, A., FORTUNATO, S., AND KERTÉSZ, J. 2009. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics 11*, 3, 033015. (pp. 20, 21, 24)

LANCICHINETTI, A., RADICCHI, F., RAMASCO, J. J., AND FORTUNATO, S. 2011. Finding statistically significant communities in networks. *PloS one 6*, 4, e18961. (p. 38)

LATAPY, M. 2008. Main-memory triangle computations for very large (sparse (power-law)) graphs. *TCS 407*, 1, 458–473. (pp. 75, 79)

LEE, C., REID, F., MCDAID, A., AND HURLEY, N. 2010. Detecting highly overlapping community structure by greedy clique expansion. *SNA/KDD'10*, 33–42. (pp. 21, 24)

LESKOVEC, J. AND KREVL, A. 2014. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data. (p. 56)

LIM, K. H. AND DATTA, A. 2013. A seed-centric community detection algorithm based on an expanding ring search. In *Proceedings of the First Australasian Web Conference-Volume 144* (2013), pp. 21–25. Australian Computer Society, Inc.

LIU, J., WANG, D., FENG, S., ZHANG, Y., AND ZHAO, W. 2016. A novel approach of discovering local community using node vector model. In *International Conference on Web Information Systems Engineering* (2016), pp. 513–521. Springer.

LOU, T. AND TANG, J. 2013. Mining structural hole spanners through information diffusion in social networks. In *Proc. of WWW'13* (2013), pp. 825–836. International World Wide Web Conferences Steering Committee. (pp. 7, 23, 56, 57, 58)

LUO, F., WANG, J. Z., AND PROMISLOW, E. 2008. Exploring local community structures in large networks. *Web Intelligence and Agent Systems 6*, 4, 387–400. (pp. 8, 18)

LUSSEAU, D. 2003. The emergent properties of a dolphin social network. *Proceedings of the Royal Society of London B: Biological Sciences 270*, Suppl 2, S186–S188. (p. 3)

MARATHE, M. AND VULLIKANTI, A. K. S. 2013. Computational epidemiology. *Communications of the ACM 56*, 7, 88–96. (p. 7)

MIHAIL, M., GKANTSIDIS, C., SABERI, A., AND ZEGURA, E. 2002. On the semantics of internet topologies. Technical Report GIT-CC-02-07, College of Computing, Georgia Institute of Technology, Atlanta, GA. (pp. 8, 18)

NAGAMOCHI, H. AND IBARAKI, T. 1992. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics 5*, 1, 54–66. (p. 33)

NEMHAUSER, G. L., WOLSEY, L. A., AND FISHER, M. L. 1978. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming 14*, 1, 265–294. (p. 69)

NEPUSZ, T., PETRÓCZI, A., NÉGYESSY, L., AND BAZSÓ, F. 2008. Fuzzy communities and the concept of bridgeness in complex networks. *Physical Review E 77*, 1, 016107. (p. 23)

NEWMAN, M. E. 2004. Fast algorithm for detecting community structure in networks. *Physical review E 69*, 6, 066133. (pp. 5, 8)

NEWMAN, M. E. 2006. Modularity and community structure in networks. *PNAS'06 103*, 23, 8577–8582. (pp. 8, 18)

NEWMAN, M. E. AND GIRVAN, M. 2003. Mixing patterns and community structure in networks. In *Statistical mechanics of complex networks*, pp. 66–87. Springer.

ORLIN, J. 1977. Contentment in graph theory: covering graphs with cliques. In *Indagationes Mathematicae (Proceedings)*, Volume 80 (1977), pp. 406–424. Elsevier. (p. 96)

PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. 1999. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66 (November), Stanford InfoLab. (p. 57)

PALLA, G., DERÉNYI, I., FARKAS, I., AND VICSEK, T. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *Nature 435*, 7043, 814–818. (pp. 21, 24)

PLESNÍK, J. 1984. On the sum of all distances in a graph or digraph. *Journal of Graph Theory 8*, 1, 1–21. (pp. 15, 51)

RAVASZ, E., SOMERA, A. L., MONGRU, D. A., OLTVAI, Z. N., AND BARABÁSI, A.-L. 2002. Hierarchical organization of modularity in metabolic networks. *Science 297*, 5586, 1551–1555. (pp. 3, 5)

REZVANI, M., LIANG, W., LIU, C., AND YU, J. X. 2018. Efficient detection of overlapping communities using asymmetric triangle cuts. *IEEE Transactions on Knowledge and Data Engineering*. (p. iii)

REZVANI, M., LIANG, W., XU, W., AND LIU, C. 2015. Identifying top-k structural hole spanners in large-scale social networks. In *Proc. of CIKM'15* (2015), pp. 263–272. ACM. (pp. iii, 7, 8, 92)

REZVANI, M., WANG, Q., AND LIANG, W. 2018. Fach: Fast algorithm for detecting cohesive hierarchies of communities in large networks. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining* (2018), pp. 486–494. ACM. (p. iii)

RINIA, E., VAN LEEUWEN, T., BRUINS, E., VAN VUREN, H., AND VAN RAAN, A. 2001. Citation delay in interdisciplinary knowledge exchange. *Scientometrics 51*, 1, 293–309. (p. 6)

ROSVALL, M. AND BERGSTROM, C. T. 2011. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PloS one 6*, 4, e18209. (p. 38)

RUCHANSKY, N., BONCHI, F., GARCIA-SORIANO, D., GULLO, F., AND KOURTELLIS, N. 2017. To be connected, or not to be connected: That is the minimum inefficiency subgraph problem. In *Proc. of CIKM'17* (2017), pp. 879–888. ACM. (pp. 87, 100)

SAHA, B., HOCH, A., KHULLER, S., RASCHID, L., AND ZHANG, X.-N. 2010. Dense subgraphs with restrictions and applications to gene annotation graphs. In *Annual International Conference on Research in Computational Molecular Biology* (2010), pp. 456–472. Springer. (pp. 8, 18)

SALATHÉ, M. AND JONES, J. H. 2010. Dynamics and control of diseases in networks with community structure. *PLoS computational biology 6*, 4, e1000736. (p. 8)

SALES-PARDO, M., GUIMERA, R., MOREIRA, A. A., AND AMARAL, L. A. N. 2007. Extracting the hierarchical organization of complex systems. *PNAS'07 104*, 39, 15224–15229. (p. 5)

SARIYUCE, A. E., SESHADHRI, C., PINAR, A., AND CATALYUREK, U. V. 2015. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *Proc. of WWW'15* (2015), pp. 927–937. ACM.

SCHAEFFER, S. E. 2007. Graph clustering. *Computer science review 1*, 1, 27–64. (pp. 5, 20)

SHAN, J., SHEN, D., NIE, T., KOU, Y., AND YU, G. 2015a. An efficient approach of overlapping communities search. In *DASFAA'15* (2015), pp. 374–388. Springer.

SHAN, J., SHEN, D., NIE, T., KOU, Y., AND YU, G. 2015b. Searching overlapping communities for group query. *World Wide Web*, 1–24. (pp. 10, 25, 99)

SHEN, H., CHENG, X., CAI, K., AND HU, M.-B. 2009. Detect overlapping and hierarchical community structure in networks. *Physica A: Statistical Mechanics and its Applications 388*, 8, 1706–1712. (pp. 21, 24)

ŠÍMA, J. AND SCHAEFFER, S. E. 2006. On the NP-completeness of some graph cluster measures. In *SOFSEM 2006: Theory and Practice of Computer Science*, pp. 530–537. Springer. (p. 70)

SOZIO, M. AND GIONIS, A. 2010. 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* (2010), pp. 939–948. ACM. (p. 10)

STOER, M. AND WAGNER, F. 1994. A simple min cut algorithm. In *Proc. of European Symposium on Algorithms* (1994), pp. 141–147. Springer. (pp. 27, 112)

STROGATZ, S. H. 2001. Exploring complex networks. *nature 410*, 6825, 268.

TANG, J., LOU, T., AND KLEINBERG, J. 2012. Inferring social ties across heterogenous networks. In *WSDM'12* (2012), pp. 743–752. (pp. 22, 57)

TANG, J., WU, S., AND SUN, J. 2013. Confluence: Conformity influence in large social networks. In *KDD'13* (2013), pp. 347–355. ACM. (p. 7)

TANG, Y., XIAO, X., AND SHI, Y. 2014. Influence maximization: Near-optimal time complexity meets practical efficiency. In *SIGMOD '14* (New York, NY, USA, 2014), pp. 75–86. (p. 7)

TONG, H., PAPADIMITRIOU, S., FALOUTSOS, C., PHILIP, S. Y., AND ELIASSI-RAD, T. 2012. Gateway finder in large graphs: problem definitions and fast solutions. *Information retrieval 15*, 3-4, 391–411. (pp. 1, 22, 23)

TSOURAKAKIS, C., PACHOCKI, J., AND MITZENMACHER, M. 2016. Scalable motif-aware graph clustering. *arXiv preprint arXiv:1606.06235*. (p. 66)

UGANDER, J., BACKSTROM, L., MARLOW, C., AND KLEINBERG, J. 2012. Structural diversity in social contagion. *Proceedings of the National Academy of Sciences 109*, 16, 5962–5966. (p. 22)

WANG, L., LOU, T., TANG, J., AND HOPCROFT, J. E. 2011. Detecting community kernels in large social networks. In *ICDM'11* (Washington, DC, USA, 2011), pp. 784–793. (p. 7)

WANG, N., ZHANG, J., TAN, K.-L., AND TUNG, A. K. 2010. On triangulation-based dense neighborhood graph discovery. *Proc. of VLDB'10 4*, 2, 58–68.

WATTS, D. J. AND STROGATZ, S. H. 1998. Collective dynamics of small-worldnetworks. *nature 393*, 6684, 440–442. (pp. 2, 36, 49, 92)

WHANG, J. J., GLEICH, D. F., AND DHILLON, I. S. 2013. Overlapping community detection using seed set expansion. In *Proc. of CIKM'13* (2013), pp. 2099–2108. ACM. (pp. 20, 21, 24, 38, 39, 80, 81)

WU, Y., JIN, R., LI, J., AND ZHANG, X. 2015. Robust local community detection: on free rider effect and its elimination. *VLDB'15 8*, 7, 798–809. (pp. 8, 10, 24, 25, 65, 67, 69, 93, 99)

XIE, J., KELLEY, S., AND SZYMANSKI, B. K. 2013. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Computing Surveys (csur) 45*, 4, 43. (pp. 4, 7, 20, 23, 24, 38, 39, 80, 81)

XU, W., REZVANI, M., LIANG, W., YU, J. X., AND LIU, C. 2017. Efficient algorithms for the identification of top-*k* strutural hole spanners in large social networks. *IEEE Transactions on Knowledge and Data Engineering 29*, 5, 1–17–1030.

YANG, J. AND LESKOVEC, J. 2013. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proc. of WSDM'13* (2013), pp. 587–596. ACM. (pp. 4, 7, 23, 38, 39, 80, 81)

YANG, J., MCAULEY, J., AND LESKOVEC, J. 2014. Detecting cohesive and 2-mode communities indirected and undirected networks. In *Proc. of WSDM'14* (2014), pp. 323–332. ACM. (p. 25)

ZHANG, Y. AND PARTHASARATHY, S. 2012. Extracting analyzing and visualizing triangle k-core motifs within networks. In *ICDE'12* (2012), pp. 1049–1060. IEEE.

ZHAO, Y., KARYPIS, G., AND FAYYAD, U. 2005. Hierarchical clustering algorithms for document datasets. *Data mining and knowledge discovery 10*, 2, 141–168. (pp. 38, 39)

ZHENG, D., LIU, J., LI, R.-H., ASLAY, C., CHEN, Y.-C., AND HUANG, X. 2017. Querying intimate-core groups in weighted graphs. In *Semantic Computing (ICSC), 2017 IEEE 11th International Conference on* (2017), pp. 156–163. IEEE.

ZHOU, R., LIU, C., YU, J. X., LIANG, W., CHEN, B., AND LI, J. 2012. Finding maximal k-edge-connected subgraphs from a large graph. In *Proc. of EDBT'12* (2012), pp. 480–491. ACM. (pp. 21, 25, 26)