ON THE PRIVACY AND UTILITY OF SOCIAL NETWORKS

YI SONG

(*B.Com., SOUTHEAST UNIVERSITY*)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2014

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Yi Song

July 31, 2014

ACKNOWLEDGMENTS

My first and foremost thank goes to my supervisor Prof. Stéphane Bressan who has supported me during the past five years and influenced me in many ways. He is a very friendly man with wisdom and humor. His insights in research and rigorous attitude are invaluable for my research. His heuristic guidance in our discussion made me think and work very independently. He provided valuable suggestions for the problems that I encountered not only in my research, but also in the everyday life. I will benefit from what I have learned from him, not only for a Ph.D. degree, but also for the rest of my life. I am deeply grateful to him.

I would like to thank Prof. Tan Kian Lee, Prof. Phan Tuan Quang, and Prof. Talel Abdessalem for serving on my thesis committee and providing many useful comments on the thesis.

I appreciate all the people coauthoring with me, especially Prof. Panagiotis Karras and Dr. Sadegh Nobari. Their contributions further strengthened the technical quality and literary presentation of our papers.

I would like to thank to my lab-mates, my friends, and my colleagues at SAP Research & Innovation during my internship, as well as all the other people with whom I have been working together in the past five years.

I would like to thank to my beloved parents, Song Jianming and Xu Qin. Their unconditional love has brought me into the world, raised me, and supported my decision to pursue a Ph.D. degree.

Last but not least, my deepest love is reserved for my husband, Tan Jin Han, who has been accompanying and taking care of me every day.

Publications

Materials in this thesis are revised from the following list of our previous publications.

- Yi Song, Panagiotis Karras, Sadegh Nobari, Giorgos Cheliotis, Mingqiang Xue, Stéphane Bressan. "Discretionary Social Network Data Revelation with a User-Centric Utility Guarantee", 21st ACM International Conference on Information and Knowledge Management (CIKM 2012) Maui, USA

- Yi Song, Stéphane Bressan. "Fast Community Detection", 24rd International Conference on Database and Expert Systems Applications (DEXA 2013)Prague, Czech Republic

- Yi Song, Stéphane Bressan. "Force-directed Layout Community Detection", 24rd International Conference on Database and Expert Systems Applications (DEXA 2013) Prague, Czech Republic

- Yi Song, Sadegh Nobari, Xuesong Lu, Panagiotis Karras, Stéphane Bressan. "On the Privacy and Utility of Anonymized Social Networks", The 13th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2011) Ho Chi Minh City, Vietnam

- Yi Song, Xuesong Lu, Stéphane Bressan, Panagiotis Karras. "On the Privacy and Utility of Anonymized Social Networks", IJARAS 4(2): 1-34 (2013)

- Xuesong Lu, Yi Song, Stéphane Bressan. "Fast Identity Anonymization on Graphs", 23rd International Conference on Database and Expert Systems Applications (DEXA 2012) Vienna, Austria

The following is a list of other publications I have participated in during my Ph.D study.

- Yi Song, Panagiotis Karras, Stephane Bressan, Qian Xiao. "Sensitive Label Privacy Protection on Social Network Data", 24th International Conference on Scientific and Statistical Database Management (SSDBM 2012) 25-27 June 2012 Chania, Crete, Greece

- Xuesong Lu, Giorgos Cheliotis, Xiyue Cao, Yi Song, Stephane Bressan. "The Configuration of Networked Publics on the Web: Evidence from the Greek Indignados Movement", ACM Web Science 2012 (WebSci 2012) Evanston, Illinois

- Jianhua Qu,Yi Song, Stephane Bressan. "PSOGD: A New Method for Graph Drawing", 4th International Workshop on Graph Data Management: Techniques and Applications (GDM 2013) In Conjunction with IEEE International Conference on Data Engineering (ICDE 2013) Brisbane, Australia

- Yi Song, Daniel Dahlmeier, Stephane Bressan. "Not So Unique in the Crowd: a Simple and Effective Algorithm for Anonymizing Location Data", ACM SIGIR Workshop PIR 2014 Brisbane, Australia

Contents

SUMMARY

The connectedness of social networks inspires the study of social networks for applications in various fields such as sociology, marketing and biology. Online Social Network Sites (SNSs) allow users to discover and share information about themselves and their peers. Data that is produced on a large scale from these networks brings challenges to the exploration of data utility, as well as the protection of data privacy.

We look into both problems from a graph perspective. In particular, we focus on community detection and graph anonymization. Graphs analyses facilitate the study of relationships or social interactions of online social networks modeled as graphs. Community detection constitutes an important tool for the analysis, by exploring the network structure and associated information. It provides insights into the network characteristics and structural properties, and thus, the social phenomena that take place. On the other hand, to protect data from private information disclosure, prevent malicious use and to ease the user concerns, graph anonymization approaches are in demand. Graph anonymization approaches perturb graphs under certain constraints such that the released data has a certain level of guaranteed privacy or data utility.

In this thesis, we discuss the concepts, related works and the problems of graph anonymization and community detection. We then design algorithms to solve the problems, respectively.

First, we study the problem of graph anonymization that protects the privacy of sensitive information in social network data. The target is to perturb the structures of naive anonymized graphs in such a way that, after anonymization, the graphs are capable of resisting attacks and preserving usersṕrivacy. We study the shortcomings of an existing state-of-the-art algorithm and propose a heuristic algorithm that not

only overcomes the shortcomings, but also improves the effectiveness and, most importantly, the efficiency of large graphs. Next, we propose a user-centric utility-driven paradigm, as opposed to the commonly used privacy-driven paradigm, and an algorithm that anonymizes graphs with a utility guarantee, while certain graph properties are preserved. We aim to offer an informative view of the network for users while resisting certain structural attacks.

Second, we study the problem of community detection on a simple graph that explores beneficial knowledge, based on the graph structures underlying social networks. The target is to discover structural communities utilizing various graph properties. We propose three algorithms that detect communities based on a wide range of graph structural properties, including the degree and clustering coefficient, closeness among the vertices, and the physical forces among the vertices when simulating the whole graph as a physical system, respectively. Through a comprehensive experimental study on both the real world network and synthetic data sets, the proposed solutions are shown to be efficient and effective.

List of Tables

List of Figures

Chapter 1

Introduction

Milgram's small world experiment in the United States [110], forerunner in the field of social network analysis, identified the small-world phenomenon. The experiment was designed to count the number of intermediaries via which letters could be sent to the target contact, from individuals who are in different cities and randomly chosen. The result suggests that the distance between two individuals is usually rather small, not exceeding six steps. Similarly, Watts's experiment [154] shows that the average number of intermediaries via which an e-mail message can be delivered to a target was around six. Leskovec and Horvitz [94] found the average distance among users of an instant messaging system to be 6.6. In view of the connectedness of real-world social networks, it is of the same interest or even more to study online social networks that have grown sharply, and are producing huge quantities of data.

Online Social Network Sites (SNSs) allow users to discover and share information about themselves and their peers, while they provide researchers with a valuable tool for social, cultural, and media studies via data analysis and mining [120]. The capacity to exchange information in such networks rests on an assumed underlying trust among users [65]. While trust is thicker among people with strong interpersonal ties, it also affects one's ability to cultivate and mobilize weak social ties for the transfer of valuable information [97]. Trust is thus essential not only for *bonding* social capital, associated with strong ties, but also for *bridging* social capital, associated with weak social ties and information-seeking behavior [75]. SNSs are valuable for the development of both types of social capital, while the positive effects of their use may be stronger for bridging social capital [55]. In short, the technological affordances of SNSs provide leverage in building weak ties and bridging social capital, while the value of these ties for an individual is mediated by interpersonal trust [97].

In order to safeguard such trust, as well as the institutional trust that users place in the owners and administrators of the SNS, the privacy of users has to be guarded from malicious users, as well as from malicious data recipients when data is published to third parties. The data from online social networks raises the interest of marketers, politicians, and sociology researchers, as well as hackers and terrorists. Mining and analyzing the data should only benefit legitimate users while no one, and more critically, no malicious user, should be able to access or infer private information.

We use simple graphs, graphs with only vertices and edges, to model social networks. Each vertex represents a user, and each edge connecting two users represents the strong social ties in the form of relationships or interactions between these two users. Ever since the emergence of graph theory in the 18th century [57], and developments in the following centuries [19], graphs have become useful representations of systems in numerous areas, e.g. biology, sociology and transportation. Graph analysis has gradually become crucial to social network analysis which started in the 1930's. Graph analysis facilitates the study of relationships or social interactions of the underlying networks represented.

However, the publication and analyzing of social network data entails a privacy threat to their users. Researchers, such as the authors of [7], quickly observed that simply hiding the identities of the users in a social network may not suffice to protect privacy. Indeed, the structure of the graph itself may leak sufficient information for an adversary with minimal external knowledge to infer the identity of users, for instance. Consequently, several graph anonymization algorithms have been proposed, that not only remove identity, but also perturb graph content and structure while trying to preserve utility for the sake of mining and analysis.

Therefore, we focus on both aspects in social network analysis. In the utility aspect, we study the detection of structural communities in graphs. In the privacy aspect, we study graph anonymization techniques that anonymize the graph data before data releasing, such that graphs after anonymization are capable of resisting attacks and preserving users' privacy.

1.1 Graph anonymization

Graph anonymization emerges from the privacy concerns in data publication. It is observed that simply hiding the identities of the users in a network may not sufficient to protect privacy [7]. Indeed, the structure of the graph itself may leak sufficient information for an adversary with minimal external knowledge to infer the identity of users, for instance. Consequently, graph anonymization algorithms have been proposed that not only remove identity but also perturb the graph content and structure while trying to preserve utility for the use of analysis.

In some cases, users want to keep their private relationships or personal information to themselves, therefore the goal of graph anonymization has been extend to prevent not only identity disclosure but also link disclosure and attribute disclosure. Ways to perturb the graph include the addition/deletion of vertices/edges, generalization of attributes associated with vertices or edges, generalization of vertices, etc. To measure how much perturbation is induced to the graph and thus how much utility is left, one common method is to compare the measurements on the graphs before and after anonymization.

1.2 Community Detection

Community study constitutes an important part of the graph analysis [157]. It provides insights into the network characteristics and structural properties [153, 135], and thus, the social phenomena that take place either online or offline [16, 145, 148]. Communities can be described as explicit or implicit, where explicit communities are created based on human decisions and member consent, e.g. Facebook Groups, and implicit communities are assumed to exist in the network and waiting to be discovered [122]. The implicit communities are related to network structure and are the targets of the most community detection methods.

Community is a group of vertices that have more connections to each other inside the group, than to the vertices outside the group. As a feature of social networks, it was first called community structure in [69]. For social network, communities suggest quick channels of information flow or better connectedness. Efficiently discovering such structures helps users to identify individuals who are closely related, and facilitates information dissemination, which is instrumental for the study of social behaviors [2], viral marketing [26], politics [56], and etc.

Community detection is sometimes referred to as graph clustering. While finding communities is similar to clustering analysis in the sense that they generate clustering assignments for each object, community detection focuses on the network topology. A variety of methods have been proposed to detect communities. As a user may belong to more than a single community, which is common in social networks, some methods are designed to discover overlapping communities instead of disjoint communities. *Modularity* is one of the popular concepts applied to measure the quality of the communities. Modularity is defined based on this idea that edges between vertices in the same community are dense, but are sparse between different communities. It is defined as the number of edges falling within groups minus the expected number in an equivalent graph with edges placed at random, where the equivalent graph means the graph with same number of edges and the same distribution of degree[115].

1.3 Contributions

Our main contributions in this thesis include the design of algorithms in two important parts of social network analysis: privacy protection of sensitive information in social network data and exploration of beneficial knowledge based on the graph structures underlying social networks.

We first design algorithms to anonymize graphs so that social network data can be revealed in a way that restricts information from malicious users while benefiting benevolent users, e.g. expanding social circles and establishing new social ties. Then, we propose techniques to assist the users with the effective analysis of data in a manner of community detection, which provides users with insights into the densely connected groups in the social networks and thus, the diffusion of influence and information and users' social opportunity.

Social network is modeled as simple graph. We explore the structural information rather than semantic information. Edges are of the same type, and so are the vertices. Edges are not associated with weights, and vertices are not associated with attributes. Nevertheless, our methods consider the connectedness between individuals who may or may not connect. We quantify such connectedness based on several graph properties, e.g. degree, clustering coefficient, closeness centrality, geodesic distance, etc. While we conduct experiments to examine the effectiveness and efficiency of our algorithms, the empirical results give a good indication of the effectiveness of the quantifications.

We list the achievements so far as follows.

1.3.1 Fast Identity Anonymization on Graphs

Liu and Terzi proposed the notion of k-degree anonymity to address the problem of identity anonymization in graphs. A graph is k-degree anonymous if, and only if, each of its vertices has the same degree as that of, at least, k-1 other vertices. The anonymization problem is to transform a non-k-degree anonymous graph into a k-degree anonymous graph, by adding or deleting a minimum number of edges. Liu and Terzi proposed an algorithm that remains a reference for k-degree anonymization. The algorithm consists of two phases. The first phase anonymizes the degree sequence of the original graph. The second phase constructs a k-degree anonymous graph with the anonymized degree sequence by adding edges to the original graph. We propose a greedy algorithm that anonymizes the original graph by simultaneously adding edges to the original graph and anonymizing its degree sequence. We thereby avoid testing the realizability of the degree sequence, which is a time consuming operation. We empirically and comparatively evaluated our new algorithm. The experimental results show that our algorithm is indeed more efficient and more effective than the algorithm proposed by Liu and Terzi on large real graphs.

1.3.2 Graph Anonymization with Reachability Constraints

Existing research addresses the graph anonymization problem by following an approach popular in the database community: a model of data privacy is defined, and the data is rendered in a form that satisfies the constraints of that model while aiming to maximize some utility measure. Still, there is no consensus on a clear and quantifiable utility measure over graph data. We take a different approach: we define a *utility guarantee*, in terms of certain graph properties being preserved, that should be respected when releasing data, while otherwise distorting the graph to an extent desired for the sake of confidentiality. We propose a form of data release which builds on current practice in social network platforms: A user may want to see a subgraph of the network graph, in which that user as well as hisconnections and affiliates participate. Such a snapshot should not allow malicious users to gain private information, yet provide useful information for benevolent users. We propose a mechanism to prepare data for user viewing under this setting. In an experimental study with real data, we demonstrate that our method preserves several properties of interest more successfully than methods that randomly distort the graph to an *equal* extent, while withstanding structural attacks proposed in the literature.

1.3.3 Fast Community Detection

We propose algorithms for the detection of disjoint and overlapping communities in networks. The algorithms exploit both degree and clustering coefficient of vertices as these metrics characterize dense connections, which we hypothesize as being indicative of communities. Each vertex independently seeks the community to which it belongs, by visiting its neighbouring vertices and choosing its peers on the basis of their degrees and clustering coefficients. The algorithms are intrinsically data parallel. We devised a version for *Graphics Processing Unit* (GPU). We empirically evaluate the performance of our methods. We measure and compare their efficiency and effectiveness to several state-of-the-art community detection algorithms. Effectiveness is quantified by metrics, namely, modularity, conductance, internal density, cut ratio, weighted community clustering and normalized mutual information. Additionally, average community size and community size distribution are measured. Efficiency is measured by the running time. We show that our methods are both effective and efficient. Meanwhile, the opportunity to parallelize our algorithm yields an efficient solution to the community detection problem.

1.3.4 Force-directed Layout Community Detection

We propose a graph-layout-based method for detecting communities in networks. We first project the graph onto a Euclidean space using the Fruchterman-Reingold algorithm, a force-based graph drawing algorithm. We then cluster the vertices according to Euclidean distance. The idea is a form of dimension reduction. The graph drawing in two or more dimensions provides a heuristic decision as to whether vertices are connected by a short path approximated by their Euclidean distance. We studied community detection for both disjoint and overlapping communities. For the case of disjoint communities, we used k-means clustering. For the case of overlapping communities, we used fuzzy-c means algorithm. We evaluated the performance of our different algorithms for varying parameters and number of iterations. We compared the results to several state of the art community detection algorithms, each of which clusters the graph directly or indirectly according to geodesic distance. We show that, for non-trivially small graphs, our method is both effective and efficient. We measure effectiveness using modularity when the communities are not known in advance and precision when the communities are known in advance. We measure efficiency with running time.

1.3.5 Local Closeness Community Detection

We propose an algorithm for the detection of structural communities in graphs, which leverages a local notion of closeness centrality. The algorithm is able to detect communities in the presence of overlaps. We define this local notion of closeness centrality by adapting the measures used in the local outlier factor algorithm to graphs. The main idea is to restrict the local neighborhood explored for the computation of a local notion of closeness centrality. This is done by computing distance, reachability distance and density of a vertex within its nearest neighbors. It is inspired by the local outlier factor algorithm where local reachability density and local outlier factor are computed with their nearest neighbors. The efficiency of our algorithm arises from the definition and application of this local notion of closeness centrality. We present the notion and the algorithm using this notion. We found that our algorithm is more effective and efficient than the algorithm using closeness centrality. We also compared the performance of our algorithm with that of two state-of-the-art community detection algorithms for overlapping communities: a label propagation algorithm, a game theory algorithm and a probabilistic model algorithm. We empirically evaluated the performance of our algorithm with varying parameters, and assessed effectiveness by calculating the normalized mutual information and omega index between the set of communities found, and the known set of communities. We show that our algorithm displays generally competitive performance on both synthetic graphs and real world graphs. It is more effective and efficient than the three algorithms for large sparse graphs.

1.4 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 presents a detailed review of background knowledge and related work. Chapter 3 and 4 present the main contributions in this thesis, which includes two graph anonymization algorithms (Section 3.1 and 3.2), a force-directed layout-based community detection method (Section 4.1), a vertex-centric community detection method (Section 4.2), and a local closeness community detection method (Section 4.3). Finally we conclude and propose the possible directions in the future work in Chapter 5.

Chapter 2

Background and Related Work

Our framework starts by modeling social networks as graphs. Anonymization techniques are applied to the graphs if they are going to be released, so that user privacy is preserved. On the other hand, we analyze the graphs, either original or anonymized, for practical applications. Out of all the kinds of graph analysis, we focus on the techniques that find dense structural communities according to graph topology. In the following discussions, we first review the general notions about graphs in Section 2.1. Then we review the community detection approaches in different categories in Section 2.2. Section 2.3 discuss the concepts and approaches related to graph anonymization.

2.1 Background

2.1.1 Graph Models

Social networks are modeled by graphs with vertices corresponding to individuals or entities, and edges corresponding to relationships or interactions among individuals. Vertices may have attributes. For example, in social networks, individuals may have profiles containing information about their age, date of birth, occupation and interests. Edges may have attributes such as edge type or weight.

In a simple and usual case, a network is formally modeled as a ***simple graph*** $G = (V, E)$. V is a set of vertices representing entities in the network and E, $\{(u, v)|u, v \in V\}$, is a set of edges representing relations or interactions between entities. G is undirected, un-weighted, and has no self-loop.

Other types of graphs have also been used to model social networks, such as bipartite graphs, graphs with weighted edges, graphs whose vertices have attributes and edges have attributes, graphs whose edges have been distinguished by sensitive or non-sensitive, etc. In our work, we focus on the simple graph where structural information is the main consideration.

2.1.2 Metrics

In this thesis, we are concerned with the topological properties of social networks. We therefore consider the metrics that quantify various structural properties of graphs. These properties or features of graphs such as connectivity and centrality are typically used for studying, for instance, information diffusion in social networks [73].

Degree of a vertex v of a graph is the number of edges incident to the vertex.

Eccentricity of a vertex v means the greatest distance between v and any other vertex.

Diameter is the maximum graph eccentricity of all the vertex in a graph.

Radius is the minimum graph eccentricity of any vertex in a graph.

Density is the ratio of the number of edges to the number of possible edges in an undirected simple graph, defined as: $D = \frac{2e}{n(n-1)}$. n is the number of vertices. e is the number of edges.

Average shortest path length, also known as **mean geodesic distance**, is the average of shortest paths for all possible pairs of vertices. The shortest path length between two vertices in a simple graph is defined as the least number of hops from one vertex to the other.

Degree centrality of vertex v is the number of vertices adjacent to it in graph G. For comparison between different graphs, we use the normalized degree centrality: $\mathcal{C}_D(v) = \frac{d}{n-1}$, d is the degree of vertex v, and n the number of vertices of graph G. A higher degree centrality may mean more connections one individual has, thus indicating a larger social circle in a social network.

Closeness centrality of vertex v_i is the inverse of the mean geodesic distance of v_i to all the other vertices in the graph: $\mathcal{C}_C(v_i) = \frac{n-1}{\sum_{i \neq j}^{n} g(v_i, v_j)}$ where n is the number of vertices of graph G, $g(v_i, v_j)$ is the geodesic distance between v_i and any other vertices in graph G. Closeness centrality measures how close a vertex is to all the other vertices. The higher the value is, the important the vertex is, since the closer a vertex is to the other vertices in a social network or information network, the faster information can be exchanged for example.

Betweenness centrality of vertex v_i is the ratio of the number of shortest paths that pass v_i to the total number of shortest paths in graph G: $\mathcal{C}_B(v_i) = \sum_{v_s \neq v_i \neq v_t \in V} \frac{\sigma_{st}(v_i)}{\sigma_{st}}$. σ_{st} is the total number of shortest paths between vertex v_s and vertex v_t, and $\sigma_{st}(v_i)$ is the number of shortest paths between vertex v_s and v_t that pass through vertex v_i. The higher the value of betweenness centrality, the more important a vertex is.

Eigenvector centrality of vertices is the principle eigenvector of the network's adjacency matrix. It indicates each vertex's importance according to its connections to important vertices (the concept used in Google's Pagerank algorithm [25].)

Algebraic connectivity [35] is the second smallest eigenvalue of the Laplacian matrix of a graph G. It suggests how well connected a graph is. If the graph is disjoint, which is to say the graph has more than one component, then its algebraic connectivity value is 0. Otherwise the upper bound for algebraic connectivity for the graph is the minimum cut set [48] while the lower bound is $\frac{1}{nD}$ where n is the number of vertices of graph G. D is the diameter of graph G.

Earth mover's distance (*EMD*) [131] measures the distance between two probability distributions. It suggests the minimum amount of work that must be performed to transform one distribution to another. To further analyze degrees of vertices, we apply it to the degree distributions of graphs.

Edit distance is used as the total number of edges deleted inserted: $\text{Cost}(G, G^*) = (E(G) \bigcup E(G^*)) - (E(G) \bigcap E(G^*))$, where $E(G)$ is the set of edges in graph G, and $E(G^*)$ is the set of edges in the anonymized graph G^*.

Clustering coefficient indicates the extent to which vertices in a graph tend to cluster together. It can be measured globally and locally.

Global clustering coefficient is the count of triangles and triples in the whole graph. It is defined as: $\frac{3*\triangle}{\Lambda}$. $\triangle$ is the number of triangles, and Λ is the number of connected triples.

Local clustering coefficient is defined for each vertex. Local clustering coefficient of vertex v_i is $\mathcal{C}_i = \frac{|e_{jk}|}{k_i*(k_i-1)}$. v_j and v_k are the neighbors of v_i. $|e_{jk}|$ is the number of edges between the neighbors of v_i while k_i is the number of v_i's neighbors.

We introduce community related metrics as follows.

Modularity [114] is defined as

$$modularity = \frac{1}{2m}\Sigma_{i,j \in V}(A_{ij} - \frac{k_i k_j}{2m})\delta(c_i, c_j) \tag{2.1}$$

where $A_{ij} = 1$ if i and j are connected, otherwise $A_{ij} = 0$, and $\delta(c_i, c_j) = 1$ if i

and j belong to the same cluster, otherwise $\delta(c_i, c_j) = 0$. Modularity is defined based on this idea that edges between vertices in the same community are dense, and are sparse between different communities. To find communities with natural division, modularity is defined as the number of edges falling within groups minus the expected number in an equivalent (the same number of edges and the same degree distribution) graph with edges placed at random [115].

The revised **modularity** for overlapping communities is defined as:

$$Q_{ov}^{E} = \frac{1}{2m}\Sigma_c\Sigma_{i,j\in c}[A_{ij} - \frac{k_i k_j}{2m}]\frac{1}{O_i O_j} \tag{2.2}$$

where O_i is the number of communities to which vertex i belongs, and O_j is the number of communities to which vertex j belongs.

Conductance for a set of vertices S is defined as

$$conductance(S) = \frac{c_s}{2m_s + c_s} \tag{2.3}$$

where $c_s = |(u, v) \in E : u \in S, v \notin S|$. It is the number of edges with one end in the set and the other end outside the set. $m_s = |(u, v) \in E : u \in S, v \in S|$. It is the number of edges in S.

Internal density for a set of vertices S is defined as

$$InternalDensity(S) = \frac{m_s}{n_s(n_s - 1)/2} \tag{2.4}$$

where m_s is the same as above. n_s is the number of vertices in S. Internal Density is the internal edge density of S.

Cut ratio for a set of vertices S is defined as

$$CutRatio(S) = \frac{c_s}{n_s(n - n_s)} \tag{2.5}$$

Cut Ratio is the fraction of existing edges out of all possible edges having one end outside the cluster.

Weighted community clustering [124] for a set of vertices S is defined as

$$WCC(S) = \frac{1}{|S|} \sum_{x \in S} f(x, S) \tag{2.6}$$

where $f(x, S) = \frac{t(x,S)}{t(x,V)} * \frac{vt(x,V)}{|S \backslash x| + vt(x, V \backslash s)}$ if $t(x, V) \neq 0$; $f(x, S) = 0$ if $t(x, V) = 0$. $t(x, S)$ is the number of triangles that vertex x closes with vertices in S and $vt(x, S)$ is the number of vertices in S that form at least one triangle with x.

High modularity suggests dense connections between the vertices within communities, but sparse connections between vertices in different communities, while a modularity value of zero suggests the connections within communities are no better than those in random graphs which have no community structures. Conductance, internal density, and cut ratio measure the quality of communities in terms of internal and external connectivity. WCC measures the community quality based on the closed triangles. High WCC suggests a higher probability of closed triangles among the vertices within communities, compared to those between communities.

Additionally, we can use a widely adopted metric called normalized mutual information (*NMI*) [91] to measure the similarity between detected disjoint or overlapping communities and ground truth.

Normalized Mutual Information (*NMI*) of two sets of communities $\{C_1\}$ and $\{C_2\}$ is defined as:

$$NMI(X|Y) = 1 - [H(X|Y) + H(Y|X)]/2, \tag{2.7}$$

where $H(X)(H(Y))$ is the entropy of the random variable $X(Y)$ associated with the set of community $\{C_1\}(\{C_2\})$, and $H(X|Y)$ is the conditional entropy of X

with respect to Y. *NMI* indicates the similarity between two sets of communities.

Omega Index is also a metric based on two sets of communities. It measures how many pairs of vertices belong together in the same number of communities. Both the *NMI* and *Omega Index* yield values between 0 and 1, with value 1 corresponding to a perfect match between the two sets of communities.

2.2 Community Detection Related Work

Community detection dates back to Weiss and Jacobson's work [156] in 1955. They sought work groups within a government agency by studying the matrix of working relationships. A large number of approaches were proposed after Girvan and Newman's work [69, 114] in 2002. Their method identifies and removes the between-cluster edges iteratively, according to the betweenness centrality. The graph is disjointed, and consequently results in communities.

2.2.1 Traditional methods

Community detection is sometimes referred to as graph clustering. Basically, it can be categorized into partition clustering, hierarchical clustering, and spectrum clustering [61]. Hierarchical clustering can be further divided into divisive clustering, and agglomerative clustering. Partition clustering is dividing the vertices into groups with a possible predefined size or predefined number of groups, so as to maximize/minimize a given metric. Hierarchical clustering computes the pairwise vertex similarities, and then iteratively merges the vertices with sufficient high similarity or divides clusters by removing the edges between vertices with sufficiently low similarities. Spectrum clustering projects vertices to some spaces and partitions them, based on the eigenvectors of matrices.

In the following sections (2.2.3-2.2.7), we review the specific methods according to a different classification scheme that is based on the features of the methods. We emphasize that many methods may fall into more than one category. In this case, we only classified them according to what we consider their main features.

2.2.2 Random-walk based methods

Several methods [123, 84, 149, 107, 129] are based on the idea of random walks. Pons and Latapy [123] calculated the similarities which they call distance, between pairs of adjacent vertices and between communities, by using short random walks, Then, they adopted Ward's [152] agglomerative hierarchical clustering approach to find communities. Community structures are obtained and represented as a tree called *dendrogram*. Jin et al. [84] proposed an algorithm based on the Markov random walk to unfold the communities, and extracted them with a cut-off criterion based on conductance. Dongen [149] utilized the *Markov Chain* to simulate the random walks. This work is based on the idea that if starting at a vertex and then traveling to other vertices randomly, the possibility of staying within the clusters is higher compared to travelling between the clusters, because there are more edges within the cluster than between clusters. The method is based on the transition probability matrix with adjustments. In the long run, this effect will disappear, so Dongen designed an algorithm that stops half way in the *Markov Chain*. [107] proposed using random walks with restarts to find significant clusters in large-scale protein networks. The idea is to expand a given cluster to include the protein with the highest proximity to the current cluster. Rosvall and Bergstrom [129] combined an information theoretic approach and random walk to detect communities in weighted and directed network.

2.2.3 Modularity-based methods

Several authors [114, 36, 117, 76, 77] focused on *modularity*, which was first proposed by Girvan and Newman [69]. Girvan and Newman in [114] proposed a divisive method to identify communities. The edges with the highest betweenness are removed iteratively, thus disconnecting the graph and creating communities. The best partition has the highest *modularity*. Clauset [36] proposed a method based on the assumption of a lack of global knowledge about the graph, and thus communities are detected by exploring one vertex at a time. Clauset defined a local measurement of community structure called *locally modularity*, and proposed an agglomerative algorithm to maximize the *local modularity* of the communities detected. Nicosia et al. [117] optimized modularity for overlapping communities on directed graphs. Gregory's method [76] finds overlapping communities by extending Girvan and Newman's method. Instead of removing an edge after identifying the edge with high betweenness, vertices are split. They defined *vertex betweenness*, which is calculated according to the edge betweenness, for the vertices to decide which, how and when to split. Gregory [77] improved the complexity of the algorithm by computing local betweenness instead of global betweenness. Still the problem with these methods is the scalability of the algorithms due to betweenness computation.

2.2.4 Clique-based methods

Some methods are based on cliques. Du et al. [50] used maximal cliques for community detecting instead of star-shaped subgraphs. The problem of star clustering [6], as they pointed out, is that the verteice with high degrees do not necessarily mean involvement in a community. Therefore, they proposed an algorithm called *ComTector*. It enumerates all maximal cliques, finds the clustering kernel in each group of the overlapping maximal cliques, assigns the rest vertices to the closest kernels, and then merges fractional communities. Palla et al. [67] designed the clique percolation method which finds all cliques of size k and thus the communities. They defined a community as the union of all k-cliques that can be reached from each other through a series of adjacent k-cliques, where two k-cliques are adjacent if they share k-1 vertices. Communities are connected union of k-cliques. Cui et al. [43] focused on finding overlapping communities given an online query vertex. They pointed out that defining a community as a k-clique component, an example of which can be seen in [67], is too restrictive. They relaxed the community requirement by defining a γ-*quasi*-k-*clique* and define a community as a γ-*quasi*-k-*clique* component. The components are obtained by a depth-first search on the clique graph.

2.2.5 Agglomerative algorithms

The methods [37, 3] detect community in an agglomerative way. Clauset et al. [37] proposed a greedy hierarchical agglomerative algorithm. It starts from each vertex being a community, and joins two communities at each iteration. Two communities are selected, based on the requirement of maximizing modularity increment. They used a dendrogram to represent the whole process. Ahn et al. Ahn et al. [3] defined clusters as sets of edges. They found hierarchical community structures through single-linkage hierarchical clustering, which groups edges pairwise according to their edge similarity until all edges are in the same cluster. Edge similarity is defined by the Jaccard index, where the sample sets are the immediate neighboring non-common vertices of the two edges.

2.2.6 Local algorithms

Some methods [11, 12, 72, 91, 41, 36] detect community locally. Baumes et al. [11, 12] proposed two heuristics to detect locally dense subgraphs as communities. Two subgraphs with significant overlap can be locally optimal, and are overlapping communities. The first heuristic finds disjoined clusters by deleting high-ranking vertices, and then adds the deleted vertices to one or more clusters. The second heuristic starts from randomly chosen seeds, and then adds or deletes one vertex at a time until the density metric cannot be further improved. Goldberg et al. [72] proposed an additional requirement to Baumes et al.'s algorithms, that requires the community to be a connected sub-graph. According to a density metric, the density of a community cannot be improved with the removal or addition of a single vertex. Lancichinetti et al. [91] proposed an algorithm that is capable of finding both hierarchical and overlapping communities. A fitness function of clusters is designed. Each cluster starts from a single vertex. Neighbors of vertices in the clusters are added and deleted one by one, as long as the fitness increases. The cluster is formed when the fitness reaches a local maximum. Then another unvisited vertex is chosen randomly to start forming a cluster. The algorithm proceeds until all vertices have been assigned to clusters. The vertices may be assigned to different clusters, which forms overlapping communities. Michele et al. [41] proposed an algorithm that democratically allows each vertex to choose their communities in their local views. Specifically, the method extracts the ego network of each vertex, applies a *label propagation* algorithm to the subgraph and then combines the results of all vertices. The algorithm is parallelizable and incremental.

2.2.7 Alternative algorithms

As time evolves, a sequence of networks is generated. Some methods [44, 17] consider dynamic networks. Cuzzocrea et al. [44] proposed a match-based community detection algorithm for dynamic network. Algorithms were proposed to capture and model various kinds of community transitions, by matching network snapshots of adjacent time steps. Boden et al. [17] detected communities at each time step using the *DB-CSC* approach [78], and then match the communities by projecting the vertices of a graph onto a dimensional vector space and check whether the two given samples were generated from the same underlying distribution.

Some methods detect communities based on structural and extra information, e.g. vertex attributes, edge content, and event information [99, 130, 126, 17]. Li et al. [99] proposed an agglomerative clustering method for detecting communities based on event information. They designed a special type of edge called *virtual links* connecting a pair of vertices representing individuals from different events who do not have direct interactions but who work on some similar topics. The algorithm starts by having the members from each event form an initial community. Then it detects similar events in terms of overlapping vertices and virtual links, and then merges them to form bigger communities in an agglomerative way until the quality of the detected communities in the merging process have become maximal. Ruan et al. [130] designed a method for community detection by combining content and link information to strengthen the community signal. The similarity of content is measured and is used to sharpen the link strength, where link strength is an estimation of probability for an edge to reside within a community. Content edges are constructed based on the content similarities. The algorithm samples the union of content edges and normal graph edges with bias to retain edges that are relevant in local neighborhoods for each vertex. Then graph partition algorithms can be applied to the sampled graph and output clusters. Qi et al. [126] proposed an edge-induced matrix factorization model to detect community incorporating edge content, which they believe provides a number of distinguishing characteristics of the communities which cannot be modeled by vertex content. They designed a latent representation which can effectively expose the community factors with the use of both structure and edge content, and then well-known clustering methods can be applied to the latent vectors to find communities of edge. Finally, vertices can be assigned to those communities correspondingly.

Jierui and Boleslaw [161] improved the speaker-listener label propagation algorithm to make it possible for disjoint community detection [127] to be capable of detecting overlapping communities. Each vertex holds one or multiple labels and iteratively updates the labels according to the popular labels among its neighbors. The algorithm scales linearly with the number of edges. Similarly, [93] used label propagation, but their method detects community in an active semi-supervised way.

Zhang et al. [173] proposed a method that combines spectral mapping, fuzzy clustering and the optimization of a quality function. Graphs are projected to a low dimensionality Euclidean space, and then the vertices are clustered by *fuzzy c-mean* algorithm. Yan and Gregory [164] proposed an optimization for existing community detection algorithms. Pairwise vertex similarities are measured beforehand, and existing algorithms are applied on the graph with the vertex similarities as edge weights.

2.3 Graph Anonymization Related Work

The need for more involved graph anonymization stems from one shortcoming of naive anonymization [7]. Naive anonymization replaces the identities of vertices with synthetic identifiers before publishing the graph. With minimal external knowledge, adversaries may be able to recover these identities from the graph structure. In this section, we discuss the attacks, anonymity, and the anonymization approaches proposed previously.

2.3.1 Attack Taxonomy

Backstrom et al. [7] proposed ***active attack*** and ***passive attack*** on social networks. An ***active attack*** uses a strategy that plants a unique small graph into the network by creating fake accounts before releasing, and then tries to find this planted graph in the anonymized graph after release, so as to get information about the targets. A ***passive attack***, as opposed to an active attack, aims to identify targets in anonymized graphs according to graph patterns that are formed by existing accounts in the social networks, instead of accounts purposely created by attacks. For these two attacks, adversaries have the structural information about subgraphs in the released graphs.

Adversaries can have various types of background knowledge to breach the privacy of data. The ability to attack depends on background knowledge of adversaries. For example, an adversary may use knowledge of quasi-identifiers to detect the identities in tabular data. Because of the versatile information contained, adversaries' background knowledge may be any of the following:

- ***vertex knowledge:*** An individual vertex has information such as degree, attributes and labels. Degree is one of the most important topology characteristics about one vertex in graph. It represents the number of relationships or interactions an individual has with other individuals. Accurate degree information is not hard to get as long as it has not been perturbed. Vertex attributes (identity excluded) can be sensitive or non-sensitive and may function as identifiers, similarly to tabular data. One example is *NodeInfo* [33], which is defined as information that is attached to a vertex, and any identifying information such as name or personal identify number is excluded.

- ***link knowledge:*** We define any link-related but not structural information as link knowledge, such as edge weight and type.
- ***structural knowledge:*** Structural knowledge includes the paths between pairs of vertices or neighborhood structural information of a certain vertex, such as a subgraph around the vertex. This kind of information is the main background knowledge for adversaries and makes the biggest difference between graph data and tabular data. The authors of [33, 81, 175, 177, 169] focus on attacks based on this kind of background knowledge.
- ***graph metrics knowledge:*** Adversaries may achieve attacks with knowledge about metrics, such as graph eccentricity, diameter, average path length, clustering coefficient, betweenness centrality, subgraph centrality and transitivity. Hay et al. [81] mentioned hub fingerprint queries utilizing information about degree and betweenness centrality.
- ***auxiliary knowledge:*** Besides information that an adversary may obtain within the network graph, there is "outside" information, called aggregate auxiliary information [4]. It is defined as large-scale information from other data sources and social networks whose memberships overlap with the target network, as opposed to individual auxiliary information [4] which is identifi-

able details about a small number of individuals from the target network and possibly relationships between them.

The background knowledge may be accurate but not complete. Hay et al. [81] discussed the possible effects arising out of whether the background knowledge is complete or not. They view an adversary with absent facts as a closed-world adversary while assuming that absent facts are not true. On the contrary, if the absent facts are just unknown, then the adversary is an open-world adversary.

Information Disclosure can be categorized into three main types: identity disclosure, attributes disclosure, and edge(link) disclosure. We view information disclosure and attacks as being the same.

- ***Identity disclosure*** : The identity of the individual (vertex) is disclosed.
- ***Attributes disclosure***: The privacy of information associated with each vertex is disclosed. It can also be called content disclosure.
- ***Link disclosure***: Sensitive relationships between two individuals are disclosed.

Several authors, e.g. [14, 102, 174] describe attacks according to this classification. Attacks have also been classified by other standards too. Cheng et al. [33] divided attacks into two types: attacks on NodeInfo, and attacks on edge information which they call *LinkInfo*. NodeInfo attack includes both *identity disclosure* and *attributes disclosures*. LinkInfo is the information about the relationships among the individuals, which may be sensitive or non-sensitive.

Cormode et al. [40] partitioned link disclosure into static attacks and learned link attacks in terms of whether adversaries have prior knowledge about relationships between individuals.

- ***Static attack*** : Adversaries analyze solely the information which is published, and try to deduce explicit relationships.

- ***Learned link attack*** : Adversaries use the published graph and a few relationships that are already known, to deduce other explicit relationships.

Some authors concentrate on ***structural attacks***, particularly since structural information specializes in social networks, compared to tabular data, and its versatility makes it the most difficult part to analyze. This kind of attacks may involve all the disclosures we mentioned early. Attacks such as the *degree attack*, *sub-graph attack*, *1-neighbor-graph attack*, and *hub fingerprint attack* all belong to the category of structural attacks. Similarly, [109] proposed three structural queries: *vertex refinement query*, *subgraph query*, and *hub fingerprint query*, corresponding to degree attack, sub-graph attack, and hub-fingerprint attack separately.

2.3.2 Anonymity

Anonymity usually refers to an individual's identity being publicly unknown. For privacy preservation in social networks, the concept becomes more diverse. Researchers have used this term to indicate anonymous sensitive attributes, anonymous sensitive relationships, or anonymous sensitive weights of edges. A huge amount of work related to anonymity has been done with tabular data, e.g. k-anonymity [142], l-diversity [106], t-closeness [98]. However, these concepts cannot be applied to graph data directly, since each record is independent in tabular data. But still concepts such as k-anonymity have been introduced and developed in the domain of graph anonymization.

k-anonymity [175] Let G be a social network and G' an anonymization of G. If G' is k-anonymous, then with neighborhood background knowledge, any vertex in G cannot be re-identified in G' with confidence larger than $1/k$.

k-degree anonymity [102] A vector of integers $\mathbf{v}$ is *k-anonymous* if every distinct value in $\mathbf{v}$ appears at least k times. A graph is *k-degree anonymous* if the degree sequence of G is *k-anonymous*.

k-candidate anonymity [109] Let Q be a structural query. An anonymized graph satisfies *k-candidate Anonymity* given Q if for any x in V, the probability, given Q, of candidate y for x is less than $1/k$. It implies that the target vertex cannot be distinguished with other at least $k-1$ vertices.

k-security [33] Let $G = (V, E)$ be a given graph with unique vertex information $I(v)$ for each vertex $v \in V$. Each vertex $v \in V$ is linked to a unique individual $U(v)$. Let G_k is k-secure, with respect to G if for any two target individuals A and B with corresponding *neighborhood attack graphs* G_A and G_B that are known by the adversary, the following two conditions hold: 1)(NodeInfo Security) the adversary cannot determine from G_k and $G_A(G_B)$ that A(B) is linked to I(v) for any vertex v with a probability of more than 1/k; 2) (LinkInfo Security) the adversary cannot determine from G_k, G_A and G_B that A and B are linked by a path of a certain length with a probability of more than 1/k. In other words, the adversary cannot disclose a certain vertex's NodeInfo with a probability of more than 1/k with the published graph and the query results, and cannot determine path of a certain length with probability of more than 1/k.

τ-confidence [172] Given a vertex description type, the confidence of a graph G is defined as $conf_d(G) = 1 - maxP_{G,D}$, where $P_{G,D} = \{p_{ij} | C_i, C_j \in P_D(G), i \leq j\}$ is the set of linking probabilities calculated based on the type-D partition of G. A graph G is τ-confidence w.r.t. D if $conf_d(G) \geq \tau$. Here,linking probability is the probability that an edge in edge equivalence class E_{ij} links a target individual in vertex equivalence class C_i and another target individual in vertex equivalence class

C_j.

k^2***-degree anonymity*** [143] A graph G is k^2-degree anonymous if, for every vertex with an incident edge of degree pair (d_1, d_2) in G, there exit at least k-1 other vertices, such that each of the k-1 vertices also has an incident edge of the same degree pair.

k***-symmetry anonymity*** [158] Given a graph G and an integer k, if $\forall \triangle \in$ automorphism partition of G, $|\triangle| \geq k$, then G is k-symmetric.

2.3.3 Anonymization Approaches

Hay et al [81] focused on the risk of re-identifying entities in an anonymized network using primarily structural information. Attacks are classified and expressed through three variants of knowledge query: vertex refinement query, subgraph query and hub fingerprint query. They studied the re-identification risks caused by these attacks on three real network data sets and found that the impacts of attacks differed significantly across different data sets. For vertex refinement, the biggest change is between the query for the degree of target vertex and the neighbors' degrees of the target vertex. Compared with the subgraph query and the hub fingerprint query, vertex refinement is more efficient, which suggests that privacy undergoes more risks under such queries at the same time. Conducting tests using random graphs and power law graphs, they also integrated the attributes of vertices into adversary knowledge and evaluated the result with regard to the number of distinct attributes values and attribute-structure correlations. They proposed the *k-candidate anonymity* model and advised anonymizing graphs by grouping vertices into partitions and publishing the number of vertices and the edge density in each partition, and also the edge density across partitions. Liu and Terzi [102] specifically focused on attacks leveraging an adversary's background knowledge of degree and proposed an anonymization framework focusing on identity disclosure. They divided attacks which they mention as privacy breaches into three categories: identity disclosure, link disclosure, and content disclosure. They proposed *k-degree anonymity* as privacy guarantee and the approach to achieve *k-degree anonymity* is completed in two steps: constructing a new degree sequence according to the original degree sequence aimed at maximum degree similarity and constructing the graph, which is the supergraph of the original graph in general based on the new degree sequence. We study this work in detail in Section 3.1 and Appendix A. Stronger privacy guarantees than those adaptions of *k-degree anonymity* are provided by models such as k^2*-degree anonymity* by Tai et al. [143]. A k^2*-degree anonymous* graph prevents re-identification by adversaries with background knowledge of the degrees of two vertices connected by an edge. Even stronger privacy guarantees than *k-degree anonymity* and k^2*-degree anonymity* are provided by *k-automorphism* [177]. Zou et al. focused on identity disclosure and structural attacks, and proposed to modify the graph to be *k-automorphic* before releasing. Any vertex in such a graph cannot be distinguished from other at least $k-1$ vertices via the graph structure, so all kinds of structure attacks are prevented. The modifications are achieved by the addition and deletion of edges and, occasionally, the addition of vertices. A method for dynamic publishing of social network data was designed as well. Similarly, Wu et al. [158] proposed the *k-symmetry* model to prevent identity disclosure. In a *k-symmetric* graph every vertex is structurally indistinguishable from at least $k-1$ other vertices. Cheng et al. [33] considered the same problem as Zou et al. [177], as they also tried to prevent general structural attacks on published graphs and protect against not only the disclosure of identity, but also those involving links and attributes. They proposed a *k-isomorphism* model, that forms k pairwise isomorphic subgraphs, to provide sufficient privacy guarantee by resisting all kinds of structural attacks. Specifically, they proposed a *k-secure privacy preserving network publication*, by publishing an anonymized graph with intact vertices, minimal anonymization cost and vertex information, whereas the graph satisfies *k-security*. They used a compound vertex ID mechanism and the anonymization method to deal with dynamically released networks. Information loss is qualified by anonymization cost that is measured by mainly *edit distance*. To ensure minimal anonymization cost, the number of different edges is minimized and the edit distance is minimized.

Zhang and Zhang [172] focused on the preservation of sensitive edges in social networks, which they called it *edge anonymity*. From their study of edge disclosure on two real world data sets, they believe that edge disclosure is more likely to happen in dense graphs, and the k-anonymous algorithm cannot guarantee complete protection against edge disclosure. They state that edge additions are more likely to cause edge disclosure, and edge deletion can always reduce linking probabilities. They proposed τ*-confidence* and degree-based algorithms to partition the naive-anonymized graph by degree, according to the pre-decided threshold of graph confidence τ, and then do edge addition or deletion to achieve better graph confidence. The authors [170, 168, 169, 88, 103, 174] considered link disclosure as well. Korolova et al. [88] studied the link disclosure by adversaries using a typical social network interface, and the information about links provided in terms of lookahead, and advised limiting the lookahead of social network interface to one or two. Ying et al. [170] considered adversaries' ability to infer sensitive edges on the anonymized graph, while in[168] the authors focused on both identity disclosure and link disclosure, assuming that all vertices and edges are sensitive. They investigated the relationship between the extent of randomization of the anonymization algorithm and the risk of disclosure, and found that, as expected, the more perturbation there is, the more privacy is preserved, and link protection needs far fewer perturbations than identity protection. Ying and Wu [169] analyzed the effects brought by a simple random edge adding or deleting algorithm and random edge switching algorithm. Randomization approaches for anonymization, such as those proposed in [109], have a significant impact on relevant topology features. They propose a randomization method that can preserve the spectrum by controlling the changes of eigenvalues of the adjacent matrix of the graph when adding or deleting or switching edges. Bonchi et al.[20] reconsidered the impact of randomization algorithms for identity disclosure and utility preservation from an information-theoretic perspective, and show that randomization techniques may achieve meaningful levels of obfuscation while still preserving characteristics of the original graph.

Several authors [175, 176, 171, 27, 40, 100, 171, 103, 46, 14] looked at graph models other than simple graphs, such as bipartite graphs. Bhagat et al. [14] modeled a social network as a rich interaction bipartite graph with entity set V and interaction set I. The edge between vertex in V and vertex in I indicates that the entity participates in the interaction. To preserve the privacy of interactions they proposed *Class safety*. The graph was anonymized by partitioning the vertices into classes of *Class safety* condition and revealing only the number of edges, or generating labels to replace the identifiers of vertices and grouping vertices into classes. Cormode et al. [40] modeled social networks as bipartite graphs too. They considered static attack and learned link attacks. They proposed to use $(k, l)-$groupings. The principle is to group vertex set V into groups of size at least k, and group vertex set W into groups of size at least l. After anonymization, the published edge set E' is isomorphic to the original edge set E, but the mapping information is hidden either partially or completely. Campan et al. [28] modeled social networks as a simple undirected graph $G = (N, E)$. N is a set of vertices with attributes that are partitioned into three categories: identifier attribute, quasi-identifier attribute and sensitive attribute. They considered both link disclosure and attribute disclosure. Vertices are divided into clusters according to the values of their quasi-identifier attributes (of categorical type or numerical type) with at least k entities, and generalized to one super-vertex for each cluster while edges are generalized inside clusters and across clusters. Compared with Zheleva et al's algorithm [174], the structural information loss caused by vertices clustering and generalization is slightly smaller in general. Liu et al.[103] modeled social networks as weighted graphs. To preserve the sensitive weights of edges, edge weights were perturbed according to Gaussian distribution. They also perturbed the edge weights of the input graphs by applying the Gaussian randomization multiplication strategy in order to achieve privacy preservation while reserving the globe structure of the graph such as the shortest path lengths. Zheleva et al. [174] modeled the social network as a graph with a single vertex type and multiple edge types, among which one of the edge types represents sensitive relationships. To prevent adversaries from predicting sensitive edges based on the observed non-sensitive edges, methods for different quantities of revealing edges were proposed and compared.

Zhou and Pei [175, 176] and Yuan et al. [171] were the first to consider modeling social networks as labeled graphs. To prevent re-identification attacks by adversaries with immediate neighborhood structural knowledge, Zhou and Pei [175] proposed a method that groups vertices and anonymizes the neighborhoods of vertices in the same group by generalizing vertex labels and adding edges. They enforced a *k-anonymity* privacy constraint on the graph, each vertex of which is guaranteed to have the same immediate neighborhood structure with other $k-1$ vertex. In [176], they improved the privacy guarantee provided by *k-anonymity* with the idea of ℓ-diversity, to protect labels on vertices as well. Yuan et al. [171] tried to be more practical by considering users' different privacy concerns. They divided privacy requirements into three levels, and suggestted methods to generalize labels and modify structure corresponding to every privacy demand. Nevertheless, neither Zhou and Pei, nor Yuan et al. considered labels as a part of the background knowledge. However, in the case which adversaries hold label information, the methods of [175, 176, 171] cannot achieve the same privacy guarantee. Moreover, as with the context of microdata, a graph that satisfies a k-anonymity privacy guarantee may still leak sensitive information regarding its labels [106].

Most methods focused on a single snapshot of social networks, and authors considered graph anonymization on dynamic network release [15, 177, 33], and Yang et al. [45] and Narayanan et al. [4] studied the problem of graph anonymization on multiple social networks containing common information.

Privacy protection comes with the cost of data utility. We empirically quantify such utility and privacy trade-off for two of the anonymization algorithms, k-degree anonymity algorithm [102] and k-automorphism algorithm [177] in Appendix A.

Chapter 3

Graph Anonymization

Mining social network data provides us with useful information, while privacy concerns arise when data is released. User identities may be disclosed. Sensitive connections may be disclosed. Malicious data recipients may use the data illegally or cantankerously [7]. For instance, providing users with information about their position among their peers in the network could help users in building their weak social ties and bridging social capital [97], i.e., expanding their social or professional circles in a desired direction. But this could pose privacy risks, i.e. the possible exploration of sensitive relationships by malicious users. The need arises for methods that provide privacy protection for sensitive data and therefore safeguard the interpersonal trust and institutional trust that users place in the owners and administrators of the SNS.

This chapter introduces our contribution to privacy protection for social network data in two aspects. First, we study the shortcomings of the k-degree anonymity algorithm, an algorithm that guarantees that after anonymization, the graph is capable of resisting attacks by adversaries with knowledge of degrees. To improve, we propose a new algorithm, which is not only more effective but also more efficient

(Section 3.1). Second, we focus on the privacy problem of revealing user data to end-users of SNSs to help them network better (Section 3.2). This problem is related to, but distinct from, the problem of revealing whole-network data to third parties. We propose a user-centric utility-driven paradigm, as opposed to the privacy-driven paradigm in previous research.

3.1 Fast Identity Anonymization on Graphs

3.1.1 Overview

Liu and Terzi [102] addressed the issue of identity disclosure of network users by adversaries with the background knowledge of vertices degree. To prevent such attacks they proposed the problem of *k-degree anonymity*. A graph is said to be *k-degree anonymous* when each vertex in the graph has the same degree as at least $k - 1$ other vertices. In other words, any vertex cannot be identified with a probability higher than $1/k$ if the adversary has the degree information of the graph. The degree sequence of such a graph is said to be *k-anonymous*. Next, the problem is to transform a non-k-degree anonymous graph into a k-degree anonymous graph by adding or deleting a minimum number of edges. For the sake of simplicity, we consider only the addition of edges. Liu and Terzi [102] proposed a two-phase algorithm. The first phase (degree anonymization) anonymizes the degree sequence of the original graph to be k-anonymous. They proposed a dynamic programming algorithm which reproduces the algorithm in [68]. The second phase (graph construction) constructs a k-degree anonymous graph with an anonymized degree sequence based on the original graph. We call this algorithm *K-Degree Anonymization* (KDA).

Typically, the degree distribution of large real world networks follows a power-law or exponential distribution [9, 38]. Consequently, there are few vertices with very large degrees and many vertices with the same small degrees. Moreover, the difference between consecutive large degrees is great.

The dynamic programming in the degree anonymization phase of KDA is designed to minimize the residual degrees, namely the difference between the original degrees and the degrees in the anonymized degree sequence. On large real world graphs, it generates a sequence at the expense of large residual degrees for large original degrees, as the differences between these large original degrees are great. It also generates a sequence with a small number of changes from the original degree sequence, as many vertices with small original degrees are already k-anonymous. It may then be impossible to compensate the large residual degrees. The sequence is thus unrealizable. Our experience suggests that, unlike what is claimed by Liu and Terzi, this situation occurs frequently. For instance, as illustrated in the example below, their dynamic programming in the degree anonymization phase does not generate a realizable degree sequence from the given data set.

Example 1. *Email-Enron is the network of Enron employees who have communicated through the Enron email. It is an undirected graph with 36692 vertices and 367662 edges. Each vertex represents an email address. An edge connects a pair of vertices if there is at least one email communication between the corresponding email users. The data set is available at http://snap.stanford.edu/data/email-Enron.html. The first 10 degrees of its degree sequence in descending order are 1383, 1367, 1261, 1245, 1244, 1143, 1099, 1068, 1026, 924. After the degree sequence is anonymized for* $k = 5$*, the 10 degrees become 1383, 1383, 1383, 1383, 1383, 1143, 1143, 1143, 1143, 1143. We see that the degree of the last vertex is increased by* $1143 - 924 = 219$*. This means that 219 vertices with residual degree are required, in order to compensate the residual degree of* 219*. However, during* *the anonymization, the number of vertices that have their degrees increased is* 212*. Moreover, most of these vertices are those with small original degrees which are already connected to that vertex. Thus, there are not enough vertices with residual degrees to be wired to the last vertex. The* k*-anonymous degree sequence is unrealizable.*

Moreover, even if the anonymized degree sequence is realizable, the graph construction phase of the algorithm may not succeed.

Liu and Terzi catered for these two situations by proposing a `Probing` scheme that enacts small random changes on the degree sequence until it is realizable and the graph is constructed. Our experience shows that a large number of `Probing` steps are, in effect, necessary to obtain a realizable sequence for practical graphs. After each `Probing` is invoked, the realizability-testing is conducted. The testing has a time complexity $O(n^2)$, where n is the number of vertices. As `Probing` is invoked for a large number of repetitions, the complete algorithm is very inefficient.

Motivated by the above observations, we study fast k-degree anonymization on graphs at the risk of marginally increasing the cost of degree anonymization, i.e., the edit distance between the anonymized graph and the original graph.

We propose a greedy algorithm that anonymizes the original graph by simultaneously adding edges to the original graph and anonymizing its degree sequence. We thereby avoid realizability testing by effectively interleaving the anonymization of the degree sequence with the construction of the anonymized graph in groups of vertices.

Our algorithm results in a larger edit distance on small graphs, but a smaller edit distance on large graphs, compared to the algorithm proposed by Liu and Terzi. Our algorithm is much more efficient than their algorithm.

3.1.2 Algorithm

The algorithm that we propose simultaneously adds edges to the original graph and anonymizes its degree sequence in groups of vertices.

The main idea of the algorithm is to cluster and anonymize the vertices of the original graph into several anonymization groups. Each group contains at least k vertices. The graph is transformed so that vertices in each group have the same degree. In order to achieve a low local degree anonymization cost, the vertices in each group should have similar degrees. For this reason, our algorithm sorts, examines and groups the vertices in the descending order of their degrees in the original graph. This choice is motivated by the observation that practical graphs often follow a power or exponential law with a long tail, according to which many vertices have and share a small degree. We therefore wire vertices with larger degrees to vertices with smaller degrees in groups until the degree sequence is k-anonymous, if it can be achieved.

Let $\boldsymbol{v}$ be the sorted vertex sequence. The `greedy_examination` algorithm clusters vertices into an anonymization group. An anonymization group is the smallest subset of $\boldsymbol{v}$ that has at least k members and whose members have a degree strictly higher than the remaining vertices. The cost of the subsequent anonymization of such a group is necessarily the sum of residual degrees after anonymization, namely, for an anonymization group $(v_i, \cdots, v_j)$ in descending order of degrees, $\sum_{l=i}^{j}(d_i - d_l)$, where d_l is the degree of vertex v_l.

The `edge_creation` algorithm adds edges in order to anonymize the vertices in a group. It wires vertices with insufficient degrees in the anonymization group to vertices with lesser degrees in $\boldsymbol{v}$ until all vertices in the group have the same degree d_i for an anonymization group $(v_i, \cdots, v_j)$ in descending order of degrees. However, we constrain the algorithm never to increase the degrees of vertices in and outside the group beyond that of the highest degree in the anonymization group, namely, d_i, for an anonymization group $(v_i, \cdots, v_j)$ in descending order of degrees. After adding edges, $\boldsymbol{v}$ is reordered according to the new degrees. At the next iteration, vertices outside the group may be further added to the newly anonymized group by `greedy_examination`, if their degree is d_i.

The anonymization group is now k-anonymous, because it contains at least k vertices with degree d_i.

The design choices in the algorithms above, in particular the wiring constraint, have been made in order to minimize the need for reordering $\boldsymbol{v}$ and to allow the processing of vertices and groups to be as sequential as possible .

Because of the wiring constraint, it is however possible that the above deterministic process does not find enough vertices to wire. Therefore, it does not construct a graph with an anonymized degree sequence. The `relaxed_edge_creation` algorithm caters for such possible failures. It relaxes the wiring constraint.

The complete algorithm, *Fast K-Degree Anonymization* (FKDA), combines the above three algorithms. FKDA always constructs a k-degree anonymous graph.

3.1.2.1 The `greedy_examination` Algorithm

At each iteration, the input to `greedy_examination` is a sequence of vertices $\boldsymbol{v}$ of length n, sorted in the descending order of their degrees, an index i such that the vertex sequence $(v_1, v_2, \ldots, v_{i-1})$ has been k-anonymous and the value of k. The output is a number n_a such that the vertices $v_i, v_{i+1}, \ldots, v_{i+n_a-1}$ are selected to be clustered into an anonymization group. Then `greedy_examination` passes $\boldsymbol{v}$, i and n_a to `edge_creation`.

The algorithm begins with an sequential examination of $\boldsymbol{v}$ starting from v_i, until v_j such that $d_j < d_i$. If there is no such v_j found, $v_i, v_{i+1}, \ldots, v_n$ have the

Algorithm 3.1: The `greedy_examination` algorithm

Input: $\boldsymbol{v}$: a sequence of n vertices sorted in the descending order of their degrees, i: an index, k: the value of anonymity.
Output: n_a: the number of consecutive vertices that are going to be anonymized.

Find the first vertex v_j *such that* $d_j < d_i$;
if v_j *is not found* **then**
    $n_a = n - i + 1$;
else
    if $d_i = d_{i-1}$ **then**
        if $n - j + 1 < k$ **then** $n_a = n - i + 1$;
        else $n_a = j - i$;
    else
        if $n - i + 1 < 2k$ *or* $n - j + 1 < k$ **then** $n_a = n - i + 1$;
        else $n_a = \max(k, j - i)$;
Return n_a;

same degree already. Below we show that there are at least k vertices from v_i to v_n. Thereby $\boldsymbol{v}$ is already k-anonymous. n_a is set to be $n - i + 1$, i.e., the number of all the remaining vertices. If v_j is found, there are two different cases depending on the result of comparison between d_i and d_{i-1}[1]. If $d_i = d_{i-1}$[2] which means that v_i has the same degree as the degree of the last anonymization group, `greedy_examination` clusters $v_i, v_{i+1}, \ldots, v_{j-1}$ in a group and merges them into the last anonymization group. Then n_a is set to be $j-i$. However, there is an exception when $n-j+1 < k$. This means that there are less than k vertices after the current group. These vertices cannot be transformed to be k-anonymous in a separated group. Thus `greedy_examination` has to cluster $v_i, v_{i+1}, \ldots, v_n$ into a group. n_a is set to be $n - i + 1$. In the other case where $d_i < d_{i-1}$, `greedy_examination` forms a new anonymization group starting from v_i. If $j - i \geq k$, which means there are at least k vertices having the same degree, `greedy_examination` clusters $v_i, v_{i+1}, \ldots, v_{j-1}$ into the new group. n_a is set to be $j-i$. Otherwise, there are less

[1] If $i = 1$, the comparison is between d_1 with n.
[2] This is caused by `edge_creation`.

than k vertices in the sequence $(v_i, v_{i+1}, \ldots, v_{j-1})$. Thereby `greedy_examination` clusters $v_i, v_{i+1}, \ldots, v_{i+k-1}$ in the new anonymization group. n_a is set to be k. However, there are also two exceptions when $n - i + 1 < 2k$ or $n - j + 1 < k$. The former means that $v_i, v_{i+1}, \ldots, v_n$ cannot form two anonymization groups. The latter means that $v_j, v_{j+1}, \ldots, v_n$ cannot be clustered into a separated group. In either exception, `greedy_examination` has to cluster $v_i, v_{i+1}, \ldots, v_n$ into an anonymization group. Then n_a is set to be $n - i + 1$.

The algorithm is described in Algorithm 1.

3.1.2.2 The `edge_creation` Algorithm

At each iteration, the input to `edge_creation` is a sequence of vertices $\boldsymbol{v}$ of length n sorted in the descending order of their degrees, an index i and a number n_a. The goal is to anonymize the vertices $v_i, v_{i+1}, \ldots, v_{i+n_a-1}$ to degree d_i by adding edges to the original graph. The output is an index, which equals $i+n_a$ if the anonymization succeeds, or equals j if v_j cannot be anonymized, where $i < j \leq i + n_a - 1$.

For each v_j in the vertex sequence $(v_i, v_{i+1}, \ldots, v_{i+n_a-1})$, `edge_creation` wires it to v_l for $j < l \leq n$, such that the edge (j, l) does not previously exist and $d_l < d_i$, until $d_j = d_i$. The former condition avoids creating multiple edges. The latter condition minimizes the need for reordering $\boldsymbol{v}$. If in the end `edge_creation` successfully anonymizes these n_a vertices, it reorders the new vertex sequence $\boldsymbol{v}$ in the descending order of their degrees. Otherwise, it returns the index j such that v_j cannot be anonymized with the wiring constraint. Then the repairing algorithm `relaxed_edge_creation` is invoked.

The algorithm is described in Algorithm 2.

We consider three heuristics to examine the candidate vertices in $\boldsymbol{v}$ for the creation of edges.

Algorithm 3.2: The `edge_creation` algorithm

Input: $\boldsymbol{v}$: a sequence of n vertices sorted in the descending order of their degrees, i : an index, n_a : the number of vertices that are going to be anonymized starting from v_i.

Output: j : an index.

for $j \in (i+1, i+n_a-1)$ **do**
    while $d_j < d_i$ **do**
        Create an edge (j,l) *where* $j < l \leq n$ *such that* (j,l) *does not previously exist and* $d_l < d_i$;
        if *The edge cannot be created* **then** *Return* j ;
Sort $\boldsymbol{v}$ *in the descending order of degree*;
Return j;

The first heuristic examines $\boldsymbol{v}$ from v_{j+1} to v_n, that is, in the decreasing order of their degrees, and creates the edge (j,l) whenever the constraint is satisfied. The second heuristic examines $\boldsymbol{v}$ from v_n to v_{j+1}. The last heuristic randomly selects a candidature v_l and creates the edge (j,l). Below we denote by 1, 2, and 3, respectively, the variants of the complete algorithm with these three heuristics.

Intuitively, the first heuristic incurs larger anonymization cost than the second heuristic does. This is because the first heuristic increases the degree of vertices with large original degree, so that the largest degrees in the some anonymization groups might be increased. In order to anonymize these groups, more edges will be added. The third heuristic should behavior in between. On the other hand, the first two heuristics construct deterministic anonymized graphs whereas the third heuristic can generate random anonymized graphs, which has consequences on the preservation of utility.

3.1.2.3 The `relaxed_edge_creation` Algorithm

The `edge_creation` algorithm is not guaranteed to output a k-degree anonymous graph. The failure occurs when an edge (j,l) with the wiring constraint cannot be created for some j. In this case, `relaxed_edge_creation` is invoked. It relaxes the wiring constraint.

The algorithm examines $\boldsymbol{v}$ from v_n to v_1 and iteratively creates an edge (j, l) if only the edge does not previously exist, until $d_j = d_i$. Then `relaxed_edge_creation` returns the index l. Notice that this iteration can always stop because in the worst case v_j will be wired to all the other vertices. Finally `relaxed_edge_creation` sorts the new vertex sequence $\boldsymbol{v}$ in the descending order of degree and feeds it as the input of `greedy_examination` in the next iteration.

The algorithm is described in Algorithm 3.

Algorithm 3.3: The `relaxed_edge_creation` algorithm

Input: $\boldsymbol{v}$: a sequence of n vertices sorted in the descending order of their degrees, i, j : two indices.
Output: l : an index.

for $l = n$ to 1 **do**
  if v_j *and* v_l *are not connected* **then**
    Create an edge (j, l);
    if $d_j = d_i$ **then**
      Sort $\boldsymbol{v}$ *in the descending order of degrees*;
      Return l;

Notice that this process may compromise the k-degree anonymity of the vertex sequence $(v_1, v_2, \ldots, v_{i-1})$ if the returned l is less than i, i.e., v_j is wired to some vertex that has been anonymized. In this case, `greedy_examination` needs to examine $\boldsymbol{v}$ from the beginning in the next iteration, i.e., i is set to be 0. In the other case where $l > i$, `greedy_examination` still examines $\boldsymbol{v}$ starting from v_i in the next iteration. However, as `relaxed_edge_creation` examines $\boldsymbol{v}$ from small degree to large degree, there is a high probability that $(v_1, v_2, \ldots, v_{i-1})$ is still k-anonymous.

3.1.2.4 The Fast K-degree Anonymization Algorithm

The FKDA algorithm combines the `greedy_examination`, `edge_creation` and `relaxed_edge_creation` algorithms. The input to FKDA is a graph G with n vertices and the value of k. The output is a k-degree anonymous graph G'.

FKDA first computes the vertex sequence $\boldsymbol{v}$ of G in the descending order of degree. Then at each iteration, it invokes `greedy_examination` to compute the number n_a and passes it with i to `edge_creation`. If `edge_creation` successfully anonymizes the n_a vertices, FKDA updates the value of i as $i + n_a$. Then FKDA outputs the anonymized graph G' if $i > n$, or enters the next iteration otherwise. If `edge_creation` fails to construct the graph, FKDA invokes `relaxed_edge_creation` and updates the value of i according to the value of l returned by `relaxed_edge_creation`. Notice that FKDA can always output a valid k-degree anonymous graph, because in the worst case a complete graph is constructed.

The complete algorithm is described in Algorithm 4.

Algorithm 3.4: The Fast K-Degree Anonymization algorithm

Input: G : a graph of n vertices, k : the value of anonymity.
Output: G' : a k-degree anonymous graph constructed from G.

$\boldsymbol{v}$=*the vertex sequence of G in the descending order of degree*;
$i = 1$;
while $i \leq n$ **do**
  n_a =`greedy_examination`$(\boldsymbol{v}, i, k)$;
  j =`edge_creation`$(\boldsymbol{v}, i, n_a)$;
  if $j = i + n_a$ **then**
    $i = i + n_a$;
  else
    l =`relaxed_edge_creation`$(\boldsymbol{v}, i, j)$;
    if $l < i$ **then** $i = 0$;
Return G';

We provide the approximate bounds of the edit distance to the original graph produced by FKDA. Suppose ideally the original vertex sequence $\boldsymbol{v}$ is clustered as follows. The sequence $(v_1, v_2, \ldots, v_{ik})$ is clustered into i groups, each of which contains k vertices, i.e., the $(j+1)^{th}$ group contains the vertices $v_{jk+1}, v_{jk+2}, \ldots,$ $v_{(j+1)k}$, $0 \leq j \leq i-1$. The sequence $(v_{ik+1}, v_{ik+2}, \ldots, v_n)$ is already k-anonymous[3]. In the best case (which is encountered in the second heuristic of `edge_creation`), the vertices in the sequence $(v_1, v_2, \ldots, v_{ik})$ are only wired to the vertices in the sequence $(v_{ik+1}, v_{ik+2}, \ldots, v_n)$ by `edge_creation`. Suppose the latter sequence is still k-anonymous after anonymization. Then we get the lower bound which is $bound_l = \sum_{j=0}^{i-1} \sum_{l=1}^{k} (d_{jk+1} - d_{jk+l})$. In the worst case (which is encountered in the first heuristic of `edge_creation`), each vertex in the sequence $(v_1, v_2, \ldots, v_{ik})$ is wired to all of its antecedent vertices. Then the largest degree of the $(j+1)^{th}$ group becomes $d_{jk+1} + jk$. Therefore the upper bound is $bound_u = \sum_{j=0}^{i-1} \sum_{l=1}^{k} (d_{jk+1} + jk - d_{jk+l}) = \frac{i \times (i-1)}{2} k^2 \times bound_l$.

3.1.3 Performance Evaluation

3.1.3.1 Experimental Setup

We implement KDA and three variants of FKDA, FKDA 1, FKDA 2 and FKDA 3, corresponding to the three heuristics in C++. We run all the experiments on a cluster of 54 vertices, each of which has a 2.4GHz 16-core CPU and 24 GB memory.

3.1.3.2 Data Sets

We use three data sets, namely, **Email-Urv**, **Wiki-Vote** and **Email-Enron** (descriptions in Appendix A.4).

We conduct experiments on these three graphs. The different sizes of the three graphs illustrate the performance of KDA and FKDA on small (1133 vertices),

[3] This is the usual case for large graphs.

medium (7115 vertices) and relatively large (36692 vertices) graphs.

3.1.3.3 Effectiveness Evaluation

We compare the effectiveness of the algorithms by evaluating the variation of several utility metrics: edit distance, clustering coefficient and average shortest path length(following [102]).

We vary the value of k in the range $\{5, 10, 15, 20, 25, 50, 100\}$. For each value of k, we run each algorithm 10 times on each data set and compute the average value of the metrics.

Figure 3.1-3.3, 3.4-3.6 and 3.7-3.9 show the results on Email-Urv, Wiki-Vote and Email-Enron, respectively.

Figure 3.1, 3.4 and 3.7 show the evaluation results of the normalized edit distance on the three graphs.

We see that FKDA adds more edges to Email-Urv but fewer edges to Wiki-Vote and Email-Enron, compared to KDA. In Email-Urv, which is a small graph with 1133 vertices, the differences between large degrees are not large. By using KDA, the residual degrees of the anonymized vertices with large original degrees can be compensated by enough number of anonymized vertices with residual degrees, that is, the anonymized degree sequence is realizable, with only a small number of repetitions of `probing`. Thus, the minimum edit distance found by dynamic programming is still less than the edit distance produced by FKDA. On the contrary, Wiki-Vote and Email-Enron are two relatively larger graphs with 7115 and 36692 vertices, respectively. The differences betweens large degrees of either graph are considerably large. Therefore by using KDA, `Probing` is invoked a significant number of times before a k-degree anonymous graph is constructed. Moreover, by comparing our `relaxed_edge_creation` algorithm with `Probing`, we find that

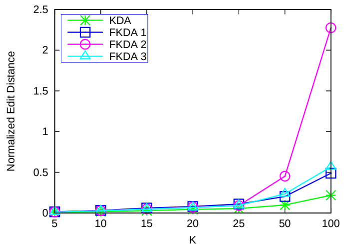


Figure 3.1: ED: Email-Urv.

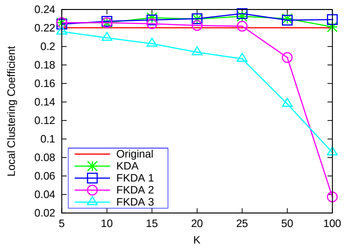


Figure 3.2: CC: Email-Urv.

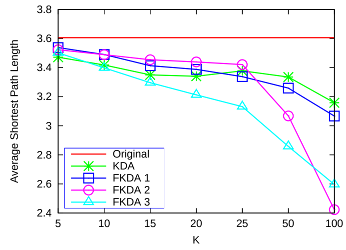


Figure 3.3: ASPL: Email-Urv.

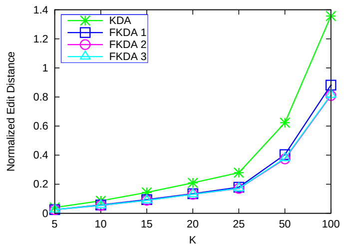


Figure 3.4: ED: Wiki-Vote.

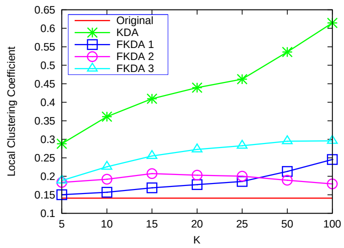


Figure 3.5: CC: Wiki-Vote.

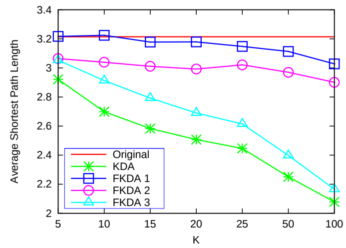


Figure 3.6: ASPL: Wiki-Vote.

`relaxed_edge_creation` increases a small degree only if the corresponding vertex can be wired to an anonymized vertex with residual degree. On the contrary, `Probing` randomly increases a small degree regardless the actual structure of the graph. The corresponding vertex may not be able to be wired to an anonymized vertex with residual degree, because an edge between the two vertices might already exist. Consequently, more repetitions of `probing` are invoked. Thus we believe that eventually, `Probing` adds more noise than `relaxed_edge_creation` does to the degree sequences of the two large graphs. Therefore, FKDA adds less edges than KDA does to the two graphs.

Figure 3.2, 3.5, 3.8 and Figure 3.3, 3.6, 3.9 show the evaluation results of the clustering coefficient and average shortest path length, respectively. The constant line shows the value of corresponding metric in the original graph.

We see that FKDA produces less similar results with that in the original graphs on Email-Urv and more similar results on Wiki-Vote and Email-Enron than KDA

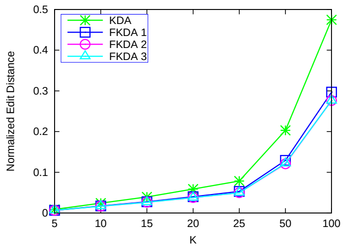


Figure 3.7: ED: Email-Enron.

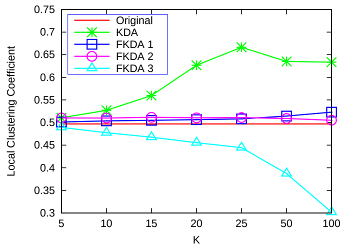


Figure 3.8: CC: Email-Enron.

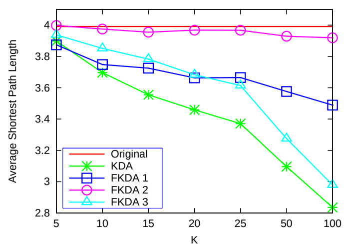


Figure 3.9: ASPL: Email-Enron.

does. This is generally consistent with the evaluation results of edit distance, since FKDA adds more edges to Email-Urv and fewer edges to Wiki-Vote and Email-Enron than KDA does.

We further compare the performances of the three variants of FKDA.

In Section 3.1.2.2 we stated that the first heuristic incurs larger anonymization cost, i.e. edit distance, than the second heuristic does, and the third heuristic performs in between. The results in Figure 3.4 and 3.7 support this claim, although the differences are small. However, in the small graph Email-Urv, we observe that FKDA 2 incurs a much larger edit distance than the other two variants and FKDA 1 incurs the smallest edit distance, for $k = 50$ and $k = 100$. The reason is as follows. When k increases, after anonymization, the residual degrees of the vertices with large original degrees become larger. Therefore, more residual vertices with smaller original degrees are required to compensate these large residual degrees. As FKDA 2 creates edges by wiring the anonymized vertices to the vertices from with a small degree to those with large degree, it makes the degrees of the anonymized vertices and the degrees of the subsequent vertices closer to each other than FKDA 1 does. Because of the wiring constraint in `edge_creation`, at some point there are not enough residual vertices to compensate the residual degree of an anonymized vertex. Then `relaxed_edge_creation` is invoked. When k is too large for the number of vertices (for example, $k = 50, 100$ and $n = 1133$ in Email-

Urv), `relaxed_edge_creation` is invoked several times by FKDA 2. Then the edit distance to the original graph is enlarged. On the contrary, FKDA 1 creates edges by wiring the anonymized vertex with a large residual degree to the vertices with large degrees to small degrees. It maintains a sufficient gap between the degrees of the anonymized vertices and the degrees of the subsequent vertices. The residual degree of the anonymized vertices can be compensated under the wiring constraint in `edge_creation`, without invoking `relaxed_edge_creation`. Therefore the edit distance is small. FKDA 3 creates edges by wiring the anonymized vertices to random residual vertices, so that it incurs the edit distance to the original graph in between.

The abilities of the three heuristics on the preservation of utility of the original graph differ from each other, depending on the structure of the original graph. For example, Figure 3.6 shows that FKDA 1 incurs a larger average shortest path length in the anonymized Wiki-Vote than FKDA 2 does. This suggests that the vertices in Wiki-Vote with similar degrees are more connected than the vertices with very different degrees. So creating edges by wiring an anonymized vertex to the vertices from with large degrees to those with small degrees (similar degrees to different degrees) in the `edge_creation` of FKDA 1 does not reduce the average shortest path length by much. On the contrary, FKDA 2 links vertices with very different degrees in `edge_creation`, which results in a significant reduction in the average shortest path length. However, Figure 3.9 shows the reverse result in the anonymized Email-Enron, which suggests that the vertices in Email-Enron with similar degrees are less connected than the vertices with very different degrees. The overall results show that FKDA 1 and FKDA 2 preserve the utilities of the original graph better than FKDA 3 does. Nevertheless, FKDA 3 has an interesting property, which is that it can generate a random k-degree anonymous graph.

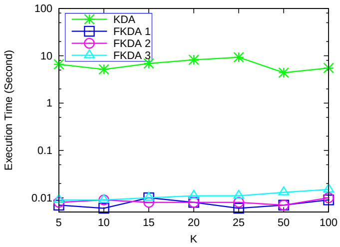


Figure 3.10: Execution time on Email-Urv.

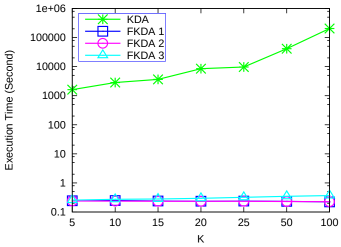


Figure 3.11: Execution time on Wiki-Vote.

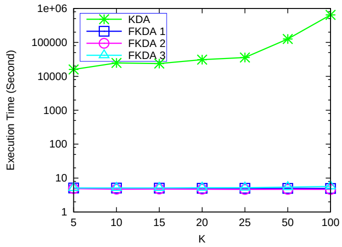


Figure 3.12: Execution time on Email-Enron.

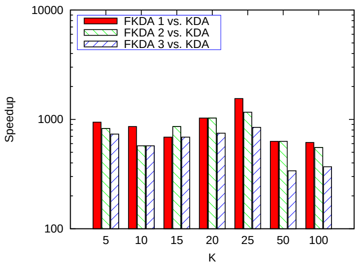


Figure 3.13: Speedup of FKDA vs. KDA on Email-Urv.

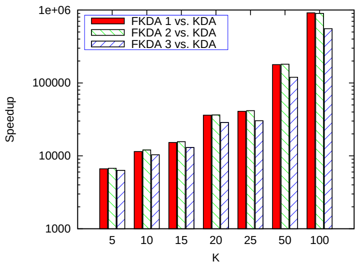


Figure 3.14: Speedup of FKDA vs. KDA on Wiki-Vote.

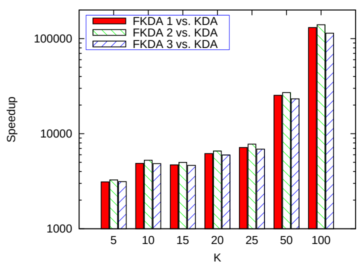


Figure 3.15: Speedup of FKDA vs. KDA on Email-Enron.

3.1.3.4 Efficiency Evaluation

We compare the efficiency of the algorithms by measuring their execution time.

We vary the value of k in the range $\{5, 10, 15, 20, 25, 50, 100\}$. For each value of k, we run each algorithm 10 times on each data set and compute the average execution time. We also compute the speedup of FKDA versus KDA for each parameter setting.

Figure 3.10, 3.11 and 3.12 show the execution times on Email-Urv, Wiki-Vote and Email-Enron, respectively. Figure 3.13, 3.14 and 3.15 show the corresponding speedups.

We see that FKDA is significantly more efficient than KDA. The speedup varies from the hundreds to one million on different graphs. The inefficiency of KDA is due to the decoupling of the checking of realizability of the anonymized degree sequences from the construction of graph.

The efficiency of the three FKDA variants is similar. FKDA 1 and FKDA 2 are slightly faster than FKDA 3. This is because FKDA 3 maintains an additional list of candidate residual vertices in `edge_creation`.

3.1.4 Summary

In this section, we propose a greedy k-degree anonymization algorithm that anonymizes a graph by simultaneously adding edges and anonymizing its degree sequence in groups of vertices.

The algorithm is designed to overcome the shortcomings of the KDA algorithm proposed in [102]. The simultaneity of degree anonymization and graph construction in the new FKDA algorithm eliminates the need for realizability testing, which, as confirmed by our experiments, is a significant factor in the poor efficiency of the KDA algorithm.

We proposed three variants of the algorithm, corresponding to three wiring heuristics. The comparative empirical performance evaluation on three real world graphs shows that the three variants of FKDA are significantly more efficient than KDA and more effective than KDA on large graphs.

We do not claim that our solution is a panacea for the anonymization of graphs in general, that objective being anyway a chimerical target given the generality of background knowledge potentially available to adversaries. It is, however, a very effective and efficient solution for the protection of privacy in the presence of background knowledge about vertex degrees. More importantly, our solution shows that it is possible to knit realizability and construction into one anonymization process tightly, and therefore pave the way to the development of algorithms catering for a variety of background structural knowledge.

3.2 Graph Anonymization with Reachability Constraints

3.2.1 Overview

The facility to ease the creation of social ties online is a central feature in any SNS [21]; such facility requires that *some* information about users is made available to both known others, and strangers. This tension between confidentiality and facility is especially pertinent in sites like LinkedIn or Xing, specializing in professional networking that eases the formation of weak ties.

Consequently, the need arises for a method that reveals network graph data in a *discretionary* manner, with the deterrence of malicious users in mind, while at the same time providing a certain utility for benevolent users; thus the problem of achieving *discretionary user-centric network data release* emerges. This problem is related to, but distinct from the problem of revealing whole-network data to third parties. We focus on the problem of revealing user data to end-users with the aim of helping them to network better. The end-user derives utility from such data revelation, and may thus willingly choose to participate in such a scheme. We aim to guarantee such utility while releasing data in a discretionary manner.

Existing research in the area follows a *privacy-driven* paradigm: it formulates a certain *privacy principle*, and develops techniques that bring the network data to a form that abides thereby, while keeping the associated loss of utility low [7, 81, 102, 15, 33]. The transformed data is then ready to be released. However, the extent to which such techniques maintain the information utility of the network and structure thereafter is vague. These studies suffice to measure ad hoc *utility metrics*; unfortunately, such metrics do not capture the extent to which an object as complex as a graph maintains its original properties. Nevertheless, in the case where the information recipients are end-users of the social network site, aiming to utilize it for networking purposes, they would like to have a guarantee precisely on the *utility* of the released data, in terms of certain *graph properties* that may be valuable for networking, no less than they would desire a certain *privacy guarantee* about preventing their own information from being revealed to others.

A network is modeled as an undirected graph $G = (V, E)$, where V is a set of vertices (nodes) representing entities, and E is a set of edges representing relations between entities. Nodes and edges may be annotated with attributes (e.g, occupation or interests for nodes, type or weight for edges), yet in this work we consider the most basic graph model.

A *naive anonymization* of G would substitute all entity identifiers in G using synthetic identifiers. However, such an anonymization does not suffice to conceal the identities behind the published graph, as the structural information in the network can itself serve to identify nodes [7, 177, 81, 102]. Thus, a *structural anonymization* is called for. Besides, a privacy threat is not posed by the identification of nodes in the network per se, but rather by the disclosure of the positions of such identified nodes with respect to each other. We contend that, when the data recipient is an end-user, a structural anonymization would suffice to provide the confidentiality that users require, while other identifying information can still be published, as it may be valuable for purposes such as professional networking.

3.2.1.1 A Practical Example

We envisage a scenario in which an SNS user requests to see the network subgraph involving one's connections up to a certain number of hops. Such a subgraph would provide the user with an overview of her position in the broader network neighborhood of her contacts and their contacts. Thus, it could provide ideas as

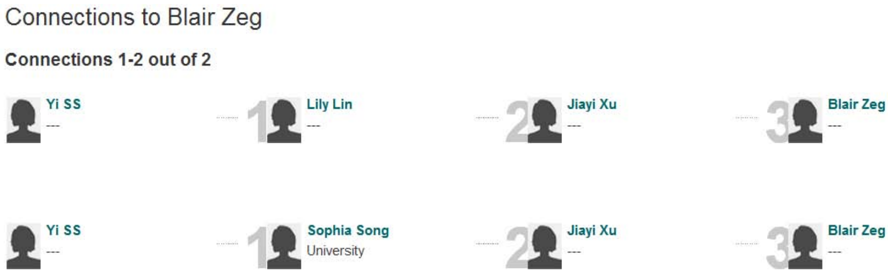


Figure 3.16: Visualization of connections in Xing

to whom she might be able to connect to next. To be truly useful, this subgraph should correctly reveal the identities of individuals within its scope and also provide some indication as to the relative positions of such individuals. However, for the sake of confidentiality, the subgraph should not reveal the *precise* relationships of such individuals among each other.

Currently, many SNS platforms, such as LinkedIn[4] and Xing,[5] provide a functionality through which users can see information about a path connecting them to other persons; in some cases, one can also see individuals along that path. This service offers valuable information to networkers. Yet, this practice poses problems, both from a privacy and a utility point of view: From a privacy perspective, the revelation of individuals along the path poses a risk to them, as the relationships among distant connections to the querying user may be sensitive and can be exploited by a malicious user. On the other hand, from a utility viewpoint, the published information is limited; a user may wish to view her position relative to a whole neighborhood, so as to identify nodes of interest; single paths do not provide such information.

Figure 3.16 shows a screen shot of the information provided by Xing in an example we have created using fictional names. Likewise, Figure 3.17 shows an example of the type of information provided by LinkedIn, again with fictional

[4]http://www.linkedin.com/
[5]http://www.xing.com/

names. While the provided information indicates the existence of a connection, it is limited to a single path, and does not reveal other graph neighborhood information that may be of legitimate interest to the user.

Figure 3.17: Visualization of connections in LinkedIn

Noticeably, Xing shows all intermediate connections, and even provides names along a single path, in contrast to LinkedIn. If taken further, i.e., to longer paths, as it stands, this practice would arguably compromise the privacy of users involved. Nevertheless, inspired by this practice, we envisage that a user could ask for a presentation of a fuller view of the network's neighborhood structure around the presented path, or, more generally, for the presentation of any network subgraph of interest. Such a service should be *discretionary*, not revealing too much information about the network's microstructure that would compromise individual users' confidentiality, yet at the same time it should be informative.

3.2.1.2 The potential for structural attacks

Nevertheless, revealing a network's structural information can render users vulnerable to attacks. A malicious user may create a set of fake accounts and attempt to forge direct links between those accounts and to one or more targets, so as to directly elicit private information from them, or to create a unique structure that can be later identified in a revealed graph. This observation is the basis of the structural attack introduced in [7]. We aim to design a utility-driven data revelation scheme that can foil such attacks.

In concrete terms, an attacker who naturally knows the identity of her targets could contact those targets directly and try to gain their trust. The chances of success at securing such targets' trust will increase if she can present herself as sharing a mutual friend, implying an endorsement of her request to connect to the target. When the path to the target is published, it becomes easier for the attacker to exploit such "friend-of-a-friend" trust. Platforms like LinkedIn and Xing appear to be vulnerable to such exploits as they publish partial, or full, path information. However, our approach will obstruct the attacker, as she will not know with certainty who is connected to whom. A guess at the exact chain that leads to her target will then be risky; if she mistakenly presents herself as a friend of a friend to any node in the chain, the chances of gaining that node's trust will be diminished. On the other hand, a benevolent networker, truthful about her intentions, will be able to solicit the assistance of users along the path to the target; as long as those users assess that she has a legitimate reason to reach her target, they will forward her request to the next hop.

3.2.1.3 Our proposal

Motivated by the above discussion, we suggest a methodology for revealing social network data to relevant users following a *utility-driven* paradigm. Through our scheme, network data is manipulated under certain constraints, aiming to preserve the structural properties of the underlying graph, while otherwise distorting the graph's microstructure to the farthest extent allowed by those constraints. In this manner, the trade-off between data utility and data privacy is addressed in a novel manner, adhering to a utility guarantee. We define the structural constraints in terms of distance properties between pairs of nodes, and demonstrate that the resulting graphs can withstand attacks by adversaries possessing prior structural background knowledge, as suggested in [7]. Specifically, in the experimental section we measure the success rate for *any* attack based on the identification of an embedded subgraph in the distorted graphs, as a function of the amount of distortion incurred on it; as we discussed, such an embedded graph may consist of fake accounts created before graph releasing, and connected among themselves and to other victim nodes, so as to follow a unique and identifiable pattern.

In our approach, we publish a subgraph of the network graph, containing *nodes of interest* with respect to the querying user (possibly along with identifying information, depending on the application at hand). This subgraph is constructed so as to faithfully preserve the *reachability* information in the true subgraph: if a node is reachable from another node by a path of length lower than a threshold k, then it should also be similarly reachable in the released graph. However, the subgraph is otherwise distorted, so as to conceal exact node-to-node relationships, to the extent allowed by the reachability constraint. Thus, a querying user cannot confidently infer the potentially sensitive relationships among distant connections. Yet the same querying user obtains a wide view of her own and her peers' position in the overall network. Thereby, a benevolent user is able to obtain valid information that is relevant in determining how to expand her network, while a malicious user is prevented from drawing accurate inferences about the relationships among people she is not closely related to, and is consequently deterred from attempting to utilize such information in order to gain their trust towards malicious ends (see also the discussion in section 3.2.4). We contend that such reachability-preserving graph transformation maintains crucial information with regard to graph structure that is valuable to the SNS user (as well as the a researcher or social network analyst), while distorting the graph in a way that renders it proof against structural attacks. Thus, the data release model we propose provides both higher utility and higher security than the naive path revelation model discussed in Section 3.2.1.1.

3.2.2 Reachability Preservation

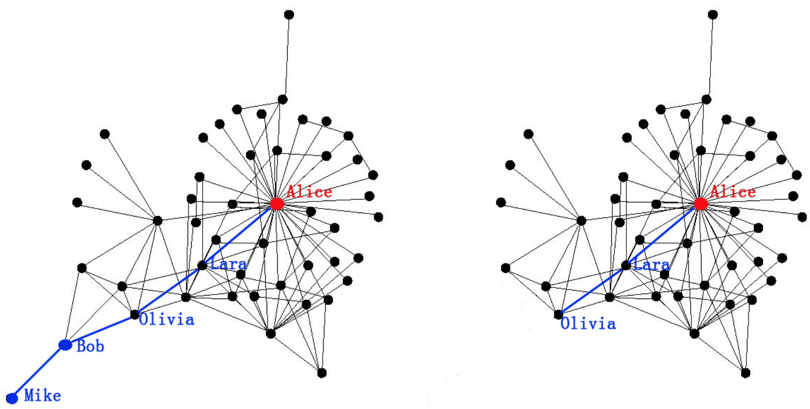
Real-world social networks of a certain size are usually *connected*; any two individuals in them are bound to be linked by a sufficiently large path. The *distance* between two individuals, i.e., the length of the shortest path connecting them, is usually rather small, not exceeding *six* steps. Milgram's small world experiment [110] suggested that the social networks of people in the United States are characterized by such short distances, of approximately three friendship links, on average, without considering global linkages; Watts [154] recreated Milgram's experiment on the internet and found that the average number of intermediaries via which an e-mail message can be delivered to a target was around six; Leskovec and Horvitz [94] found the average distance among users of an instant messaging system to be 6.6; Goel et al. [71] tested the extent to which pairs of individuals in a large social network can actually *find* paths connecting them; they introduced a rigorous way of estimating true chain (i.e., *search distance*) lengths in a messaging network, and found that roughly half of all chains can be completed in 6-7 steps.

In view of this *connectedness* of real-world social networks, we deduce that no previously unknown information is disclosed when the mere *existence* of a path among two entities in a network is revealed. Thus, an objective of thwarting the inference of any linkage *whatsoever*, as in [33], would set an unnecessarily high goal and irretrievably alter the nature of the network. Besides, a bona fide SNS user can reasonably expect to be able to learn whether other individuals in the same network are *reachable* at up to a certain distance threshold and also gain a glimpse of the nature of the network that stands between them. Such information is vital to SNS users, e.g., job seekers in a professional network, newcomers in a city, or professionals looking for new partners. On the other hand, a *discretionary* revelation of such reachability information should *not* reveal the exact relationships among people in the exposed neighborhood, as malicious users can may take advantage thereof to launch attacks and gain access to potentially sensitive information.

As we discussed, professional networking platforms provide a function that concerns us: when users search for someone, they can see the path that leads from their node to the searched-for person, possibly under the condition that the path is not longer than 3 hops. Thus, Alice can see that the path $Alice \rightarrow Lara \rightarrow Olivia \rightarrow Bob$, connects her to Bob. An extension of this functionality to paths of arbitrary length would endanger users' confidentiality, as Alice would then acquire intimate knowledge about the relationships of people she is not acquainted with. Yet, Alice has a legitimate interest to find out whether she is connected to a certain individual by a path longer than the ones she is already allowed to see, as well as to identify individuals in her extended neighbourhood, and thereby possibly attempt to expand her social circle.

Motivated by such needs, we propose a *discretionary* graph publication model that provides useful connectivity and reachability information, along with other rich graph information, without correctly revealing the graph's microstructure concerning individuals lying along the presented connections. The connections shown in a graph published by our method are not necessarily true. Still, the published graph is constructed so that it *does* provides fairly correct reachability information. In effect, a bona fide user can use such information to explore the possibility of connecting to others, and attempt such a connection by whatever means a given SNS platform provides. Still, a malicious user would not be able to exploit the presented network view without risking being exposed. In effect, graph reachability information is made available in a way that preserves certain properties of the underlying graph, while confining the potential that sensitive information is exposed.

Furthermore, by our proposal, users in the network can specify a distance threshold parameter d, so that they can quantify their own comfort zone. Figure 3.18(a) depicts an example of a graph shown to user Alice, in which it is revealed that another user, Mike, is reachable within 4 hops. This happens *under the condition* that Mike has agreed to have the information about being reachable by 4 hops available to such other users; i.e., Mike has set his personal distance threshold to $d = 4$. Alice then gets the highlighted path information if she wants to see her position relative to Mike's position, even though this particular path may *not* be the *exact* path between Alice and Mike. Figure 3.18(b) shows what Alice would see in case Mike has not opted to make his information available to users within 4 hops. To encourage users' participation, Alice's ability to view Mike's information can be made conditional on her making her own information available to users within 4 hops, i.e. her own personal distance threshold being at least 4.



(a) (b)

Figure 3.18: Example of path revelation.

Given such a facility, we expect that users will be willing to accept the discretionary revelation of their own presence in the network, as they stand to gain in terms of increased networking functionality. Naturally, when releasing network data to third parties, we expect end-users to be primarily concerned with the protection of their confidentiality rather than with the utility of the released data.

However, when network data is released among SNS end-users themselves, as in our primary motivating scenario, we expect that these end-users will have a stake in data utility and be willing to opt in such a scheme, as they will be among the beneficiaries of the information that will be provided. By setting a personalized exposure distance threshold d, users can tailor the tradeoff to their own needs and sensibilities. In the following discussion, it is always assumed that we are dealing with a set of users whose distance threshold permits their inclusion in the revealed graph.

3.2.2.1 Problem Definition

Let $G = (V, E)$ be a simple undirected graph that represents part of a social network; such a graph can consist of the network neighborhood of a querying user's node. V is a set of vertices representing entities in the network, and E is a set of edges representing relations between entities. We start out by providing the following definition.

Definition 1. *The* k-reachability graph *of* G*,* G^k*, is a graph having the same vertices* V *as* G*, such that an edge between two vertices exists in* G^k if and only if *the distance between them is at most* k*.*

For example, the 2-reachability graph of the graph G_1 at the left side of Figure 3.19 is the graph in the middle of the figure. If k is set to be the longest distance (i.e., the *diameter*) in G, then the k-reachability graph becomes trivially similar to the *transitive closure* of G. However, for intermediate values of k, G^k is rich in information, showing which entities in the network share connections of up to a certain length.

Our main claim is that, given a network neighborhood G and a certain k of interest, a graph G', having the same vertices, equal number of edges, and the

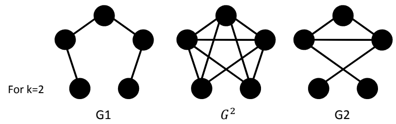


Figure 3.19: Graphs G_1 and G_2 having the same G^2

same k-reachability graph G^k as G, while differing from G in as extensive a way as possible otherwise, provides high-utility information about G in a manner *discretionary* with respect to the confidential information of the users involved. We aim to devise a method that generates G' given G. We define the following problem:

Problem 1. *Given a graph $G(V, E)$ and an integer k, produce a graph $G'(E', V)$, such that $|E| = |E'|$ and $G^k = G'^k$, while the difference between G and G', measured as the edit-distance of their edge sets, $Dist(G, G') = \frac{|E \cup E' \setminus E \cap E'|}{|E|}$, achieves a required value θ.*

In this problem, the graph G represents the network neighborhood around a querying user's node u. The parameter k defines the view of that neighborhood that a user wishes to obtain. By definition, the obtained graph G' effectively reveals which users are within k hops of u or of each other.

Furthermore, we propose that each user u in the network may set: (i) an one-to-one distance threshold d_u, which defines that any user u' lying at most d_u away from u can obtain information about their connection; and (ii) a universal distance threshold k_u, which defines that the information of u lying k_u *or more* hops away from any user u' can be revealed to a third user u''. A cautious user u would set a *low* d_u threshold (i.e, would prefer to reveal distance information only to close connections), and a *high* k_u threshold (i.e., would prefer not to let one's close connections to be accurately known by strangers). Generous default values could be set as $d_u = 3$ and $k_u = 2$.

For a user's node in the network, say u', let $d(u, u')$ be the actual network distance between u' and the querying user u. Then, if $k \geq d(u, u') > d_u$, i.e., if u'

has not given consent for her network distance from u to be revealed to u, then u' shall not be included in the neighborhood graph G we examine, as presented to u. Furthermore, if $k_u > k$, i.e., if u' has not given consent for the information that her network distance from any other node is at most k hops to be revealed to *third* parties, then, once again, u' is not to be included in G, as presented to such third parties. In effect, G, as presented to a querying user u would only contain the nodes of those users who are comfortable having their distance from u, being less than k, revealed to u (or whose distance from u is larger than k) *and* are comfortable with information about their at-most-k-hop connections to other users being revealed to u as well. Thus, users can define their own privacy objectives [79].

The requirement that $G^k = G'^k$ in Problem 1 defines our ideal objective. A graph G' that satisfies this *reachability requirement* for a large value of θ may not exist, and, even if it exists, may be hard to find. After all, this reachability requirement is strict, and does not allow much flexibility. In many practical circumstances, a more flexible version of the same requirement may still satisfy our objectives. Therefore, we suggest such a *relaxed* version of the reachability requirement that would be easier to satisfy while still maintaining much of the information we wish to preserve.

3.2.2.2 Relaxing the Reachability Requirement

Let $d(v_1, v_2)$ ($d'(v_1, v_2)$) be the distance of vertex v_2 from vertex v_1 in G (G'). Then the standard reachability requirement, i.e., the requirement that $G^k = G'^k$, can be analytically expressed as follows:

Definition 2. Reachability Requirement (RR) *A graph $G'(V, E')$ is said to satisfy the* reachability requirement *with respect to an original graph $G(V, E)$ for a given integer k, if and only if $|E| = |E'|$, and, for any pair of nodes $v_1, v_2 \in V$, it* *holds that* $d(v_1, v_2) \leq k \Leftrightarrow d'(v_1, v_2) \leq k$.

The strictness of the standard reachability requirement emanates from the fact that a distance that does not exceed k in G should not exceed k in G' either, and vice versa. A slightly less rigorous version of this requirement would impose a lighter constraint by allowing for some laxity in the preservation of distances with a definite threshold k. In effect, we can relax the requirement by demanding only that a distance not exceeding $k-1$ in G does not exceed k in G', and vice versa. This relaxation is twofold: First, we reduce the amount of distances involved, as we now care only for distances in the range $[1, k-1]$ instead of the range $[1, k]$. Second, we introduce some laxity in the preservation of distances within this range, by allowing that each distance in the range $[1, k-1]$ in G is mapped to a distance in a wider range, namely the range $[1, k]$ in G', and vice versa. We express this relaxed requirement as follows:

Definition 3. Relaxed Reachability Requirement (RRR) *A graph* $G'(V, E')$ *satisfies the* relaxed reachability requirement *with respect to an original graph* $G(V, E)$ *for a given integer* k*, if and only if* $|E| = |E'|$*, and, for any pair of nodes* $v_1, v_2 \in V$*, the following implications hold:*

$$
\begin{aligned}
d(v_1, v_2) < k &\Rightarrow d'(v_1, v_2) \leq k \\
d'(v_1, v_2) < k &\Rightarrow d(v_1, v_2) \leq k
\end{aligned}
$$

Under this relaxation, G' still presents representatively small distance values (i.e., values $d' \leq k$) for short distances in G (i.e., $d < k$) and avoids the misrepresentation of longer distance values in G (i.e., values $d > k$) as short in G' (i.e., as $d < k$). Thus, we contend that a graph G' satisfying the relaxed, instead of the standard, reachability requirement with respect to G provides slightly less precise, but still rich, information about the distances between vertices of interest, yet allows for much-desired higher flexibility in modifying the graph, which allows for a higher degree of protection against structural attacks. In the following section we present an algorithm that generates graphs satisfying either the RR or the RRR with respect to an original graph G, and hence provides an avenue for revealing a modified, utility-preserving and discretionary version of G.

3.2.2.3 Algorithm

The problem could be tackled by an exhaustive-search algorithm that would try out all the combinations of edges that could make a modified graph. However, such an exhaustive search becomes computationally prohibitive as the size of the graph grows. Instead, our Similar Reachability Graph (SRG) algorithm (Algorithm 5) modifies the graph step by step, by alternatively adding or deleting one edge at a time. At each step, we opt for a modification that satisfies the standard (or relaxed) reachability requirement. As long as modifications that satisfy the requirement are possible, we keep updating the graph, while keeping track of the distortion inflicted thereon (i.e., the number of edges altered). Once the inflicted distortion reaches a desired level θ, the algorithm terminates and the modified graph is output.

Our SRG algorithm makes use of a basic operation that computes the distance matrix $\mathcal{D}$ of a graph G. Having the $\mathcal{D}$ of the original graph G, as well as the distance matrix $\mathcal{D}'$ of a modified graph G', we can check whether the standard or relaxed reachability condition is satisfied, and calculate the respective k-reachability graphs G^k and G'^k as well. To that end, we employ the Warshall-Floyd algorithm [60], with extra pruning and optimization provisions, so as to eschew the computation of distances larger than the k threshold, which is, unnecessary for our problem.

At first, SRG constructs lists of edges that are candidates for addition (deletion).

All edges in G are candidates for deletion, while edges that are candidates for addition are those that do not exist in G, but exist in G^k. In more detail, SRG starts out with the original graph G, and proceeds to perform iterative modification steps. At each iteration, it progressively checks all allowed combinations of λ edges to delete and λ edges to add, starting with $\lambda = 1$ and increasing λ progressively, until it detects an add/delete combination that produces a modified graph G' satisfying the (relaxed) reachability requirement, (R)RR, with respect to G. Having succeeded in this iteration, it proceeds to modify the obtained graph G' further in the next iteration.

Algorithm 3.5: SRG

Input: graph G with V vertices and E edges;
reachability k; distortion threshold θ;
Result: Modified Graph G'

compute distance matrix $\mathcal{D}(G)$;
initialize G' as G;
initialize delete-candidate edge list $\mathcal{L}_1$, length ℓ_1;
initialize add-candidate edge list $\mathcal{L}_2$, length ℓ_2;
while $Dist(G, G') < \theta$ **do**
    for $\lambda \leftarrow 1$ **to** $\min\{\ell_1, \ell_2\}$ **do**
        for *each edge set* $C_1 \leftarrow \binom{\ell_1}{\lambda}$ **do**
            for *each edge set* $C_2 \leftarrow \binom{\ell_2}{\lambda}$ **do**
                delete C_1 from and add C_2 to G';
                if G' *satisfies (R)RR wrt* G **then**
                    update $\mathcal{L}_1$ and $\mathcal{L}_2$;
                    Break for loops;
                else
                    add back C_1 and delete C_2;
Return $G'(V, E')$;

We emphasize that the satisfaction of the (R)RR is always checked with respect to the original graph G, not to the modified graph of the preceding step. Thus, throughout the modification iterations, we always maintain a modified graph G' that satisfies the (R)RR with respect to G.

These modification iterations terminate when the modified graph G' has achieved a *desired* difference from the original graph G, for the sake of withstanding structural attacks. We measure the difference between graphs $G(V, E)$ and $G'(V, E')$ in terms of *distortion*, defined as the ratio of the number of edges they do *not* share to $|E|$: $Dist(G, G') = \frac{|E \cup E' \setminus E \cap E'|}{|E|}$; since $|E|$ is not changed by the algorithm, the distortion depends on the amount of edges altered, $|E \cup E' \setminus E \cap E'|$. Distortion values near 100% (i.e., half the maximum possible value of 200%) provide the highest obfuscation, as one cannot tell with confidence neither that an edge in G' also appears in G, nor that it does not. This metric has also been used as a vague way of measuring *information loss* in previous research [102]; we employ it here simply as a measure that show how much a graph is being distorted, without making any claim that correctly captures any other quality.

Our SRG algorithm works with both the standard reachability requirement (RR) and the relaxed one (RRR). The satisfaction of this requirement is checked in Step 10, by comparing the distance matrix of the modified graph, (G'), to that of the original graph. In the next section we proceed to an experimental study, in which we opt for using the RRR; this choice allows for higher flexibility, while still preserving, as we will show, rich structural information.

The SRG algorithm is a heuristic, and its practicability rests largely on the expectation that a modified graph G' satisfying the (R)RR will be arrived at early, before the value of λ grows beyond value 2. This expectation is verified by our experiments. For the sake of completeness, we provide a worst-case complexity analysis. In a worst-case scenario, half of the possible edges are present in the graph, i.e., $\ell_1 = \ell_2 = \ell = \frac{n(n-1)}{4}$, yielding $\sum_{\lambda=1}^{\ell} \binom{\ell}{\lambda} = O\left(2^{\frac{n^2}{4}}\right)$ selections of edge sets for addition and removal, hence $O\left(2^{\frac{n^2}{2}}\right)$ graph modifications in total. Since the distance matrix computation by the Warshall-Floyd algorithm costs $O(n^3)$, the overall complexity is $O\left(2^{\frac{n^2}{2}} n^3\right)$. In practice, we expect our algorithm to terminate without raising such high computational demands, as soon as a graph G' satisfying the (R)RR is discovered (Lines 10-12).

3.2.3 Experimental Evaluation

In this section we evaluate our algorithm using real data sets. The experiments ran on an Intel Core, 2 Quad CPU, 2.83GHz, 4GB machine running Windows 7. The algorithm was implemented in Standard C, while computations of matrix utility measures were conducted in Python.

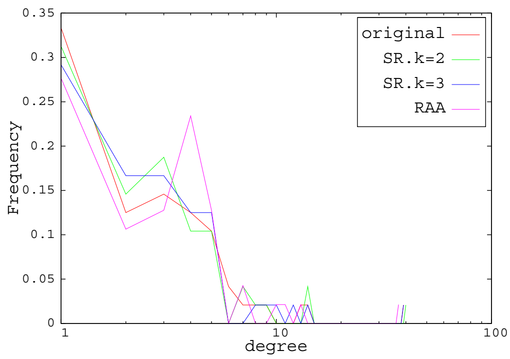
3.2.3.1 Data Description

We used two real data sets, representative of social network graphs, which are made freely available for research purposes. The former, Flickr[6][59], contains user-to-user links in an online social network for image and video hosting. Five subgraphs used in our experiments are uniformly sampled, with 50 vertices and around 100 edges for each. The latter data, Gnutella[7], describes a peer-to-peer file sharing network. Nodes represent hosts in the network topology and the edges connections between hosts. We uniformly sample 5 connected subgraphs of the 2002 Gnutella network snapshot, containing 50 vertices and around 52 edges for each subgraph. We emphasize that the data sizes we test are representative of the small neighborhoods graphs that arise in the applications we envisage. We focus on how the *structure* of such graphs can be published in a discretionary and information-rich manner. We are not making any assumptions on how, and to what extent, other information in those graphs, e.g., node attributes, may be revealed. The problem of publishing such attributes can be treated using techniques for microdata anonymization [31], as in [33], and is orthogonal to the problem of publishing the graph structure.

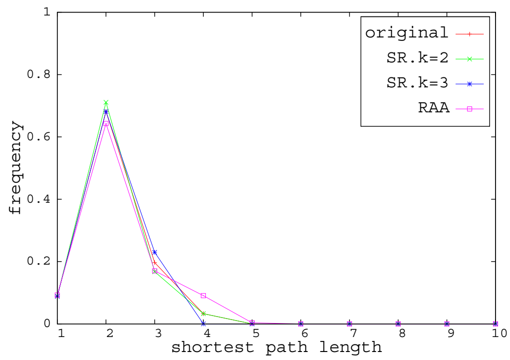
[6]Available online at http://socialnetworks.mpi-sws.org/
[7]Available online at http://snap.stanford.edu/data/

3.2.3.2 Utility Assessment

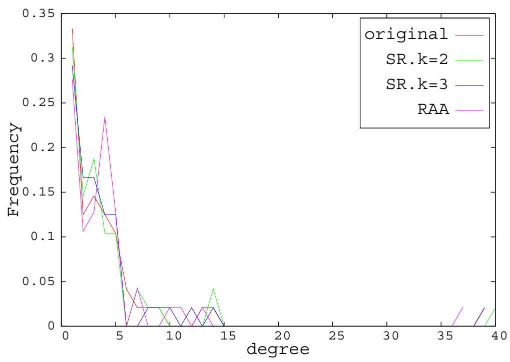
We claim that, apart from, and because of, satisfying the reachability constraint, graphs generated by our SRG algorithm preserve other structural properties of the original graph G. To demonstrate our claim, we compare graphs obtained by our methods to graphs *of the same distortion* obtained via the randomized anonymization technique proposed by Hay et al. [109]. This technique modifies the original graph by randomly deleting a prescribed number of edges and randomly adding the same number of edges; thus, the resulting graph has the same number of edges as the original graph. We refer to the algorithm of Hay et al. as "RAA".



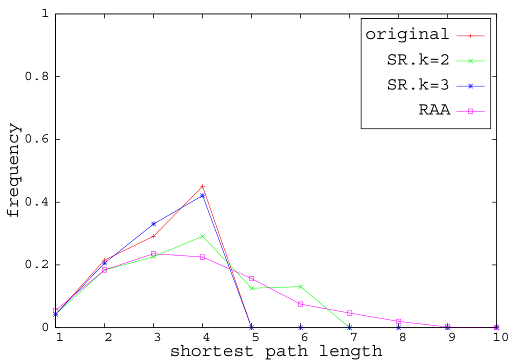
(a) DD, Flickr, distortion 0.16



(b) GD, Flickr, distortion 0.16



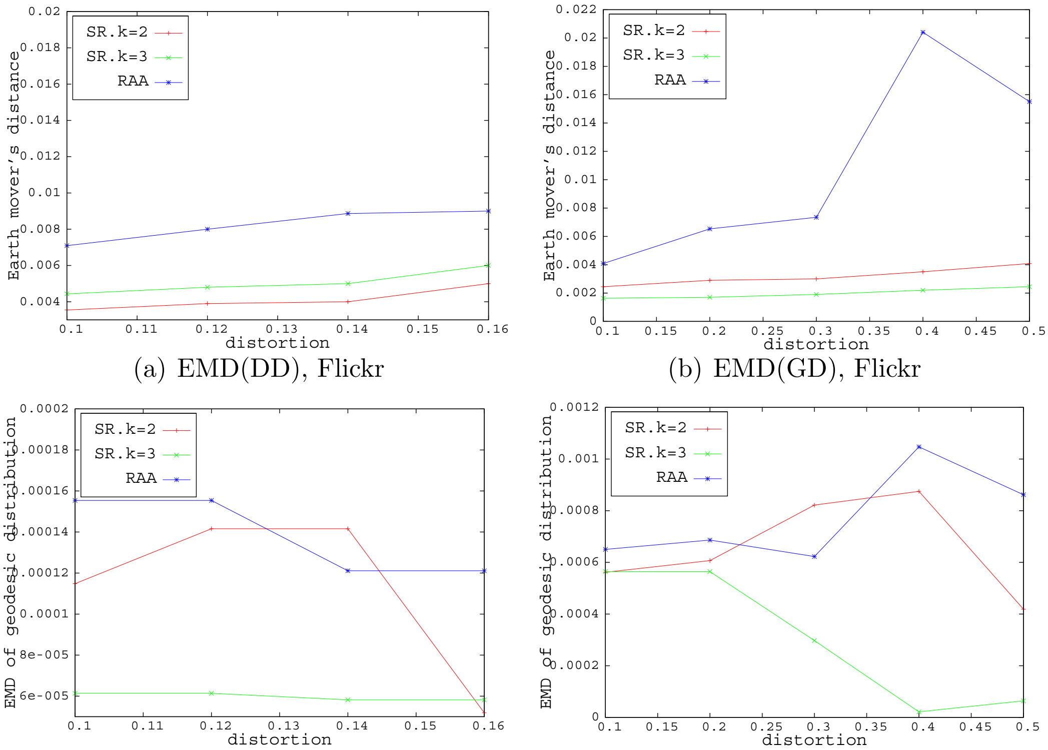
(c) DD, Gnutella, dist 0.5



(d) GD, Gnutella, dist 0.5

Figure 3.20: Degree distribution (DD) and geodesic distribution (GD) results

In our first experiment, we present the degree distribution and distribution of pairwise shortest-path (geodesic) distances, of the original Flickr graph, its SRG-generated modification (SR), and a random perturbation of the same graph by



(a) EMD(DD), Flickr (b) EMD(GD), Flickr

(c) EMD(DD), Gnutella (d) EMD(GD), Gnutella

Figure 3.21: Earth mover's distance of degree distribution and geodesic distribution

RAA having the same distortion. Figure 3.20 (a)(b) shows the results for graphs in which we allow distortion 0.16 and set $k = 2$ and $k = 3$ separately. We observe that, as expected, the distribution of those features with SRG resembles those of the original graph more faithfully than those of RAA.

Next, we present results on the same measures with the Gnutella data. Figure 3.20 (c)(d) presents our measurements when we allow distortion to reach 0.5 and set $k = 2$ and $k = 3$ separately. In both figures the distributions for SRG graphs stand relatively closer to those of the original graph. This outcome further confirms our contention that our method provides a solid way of keeping other structural graph properties under tight control. Interestingly, we observe how, even under the relaxed reachability requirement, the SRG graph with $k = 3$ does not allow any shortest-path distance to exceed the original graph's diameter 4 (Figure 3.20 (d)).

Next, to obtain a more precise estimation of the degree to which SRG graphs resemble the original ones, we measure the metric that express their structural divergence: the Earth-Mover's Distance (EMD) [132] between the original and modified degree distributions, for different distortion values. Figure 3.21 (a)(c) shows the EMD between the degree distributions on SRG graphs with $k = 2$ and $k = 3$, and RAA-perturbed versions of the original Flickr and Gnutella graphs, respectively, and the original ones, as a function of their distortion, while Figure 3.21 (b)(d) shows the EMD between the geodesic distributions. We observe that, as expected, the measured metric on the SRG graphs diverge from those of the original graph much less than those on the RAA graph, even though all graphs are obtained with the same distortion.

Remarkably, the SRG graph with $k = 2$ fares better than that for $k = 3$ with Flickr data, but not with Gnutella data. This deviation is not surprising; the parameter k that allows for the best preservation of other structural graph properties under the same distortion depends on the nature of the data at hand; in some cases, a lower k may be advantageous, as it enforces the preservation of short-distance links; in other cases, a higher k may be preferable, as it encompasses more vertex pairs under its scope.

Then, we assess the divergence between original and anonymized graphs on other graph properties: the average local clustering coefficient, the average shortest path length, the graph diameter and radius. For each data set, the results are averaged over 5 subgraphs, with 5 runs for each subgraph. Figure 3.22 shows the results for the Flickr and Gnutella data. Again, we observe that the SRG graphs produce measures that fall much closer to those of the original graphs than the RAA graphs do. These results corroborate our claim that SRG graphs can maintain the properties of the original despite the inflicted distortion.

Given that we employ the relaxed reachability requirement in our experiments, the results to reachability queries are expected to have a slight error. We end our utility assessment by quantifying this error in terms of *precision* and *recall* measures on reachability queries, in which a user asks whether a target node is reachable within a certain number of k hops. In addition, we present our measures of *false negatives* and *false positives* under the same settings.

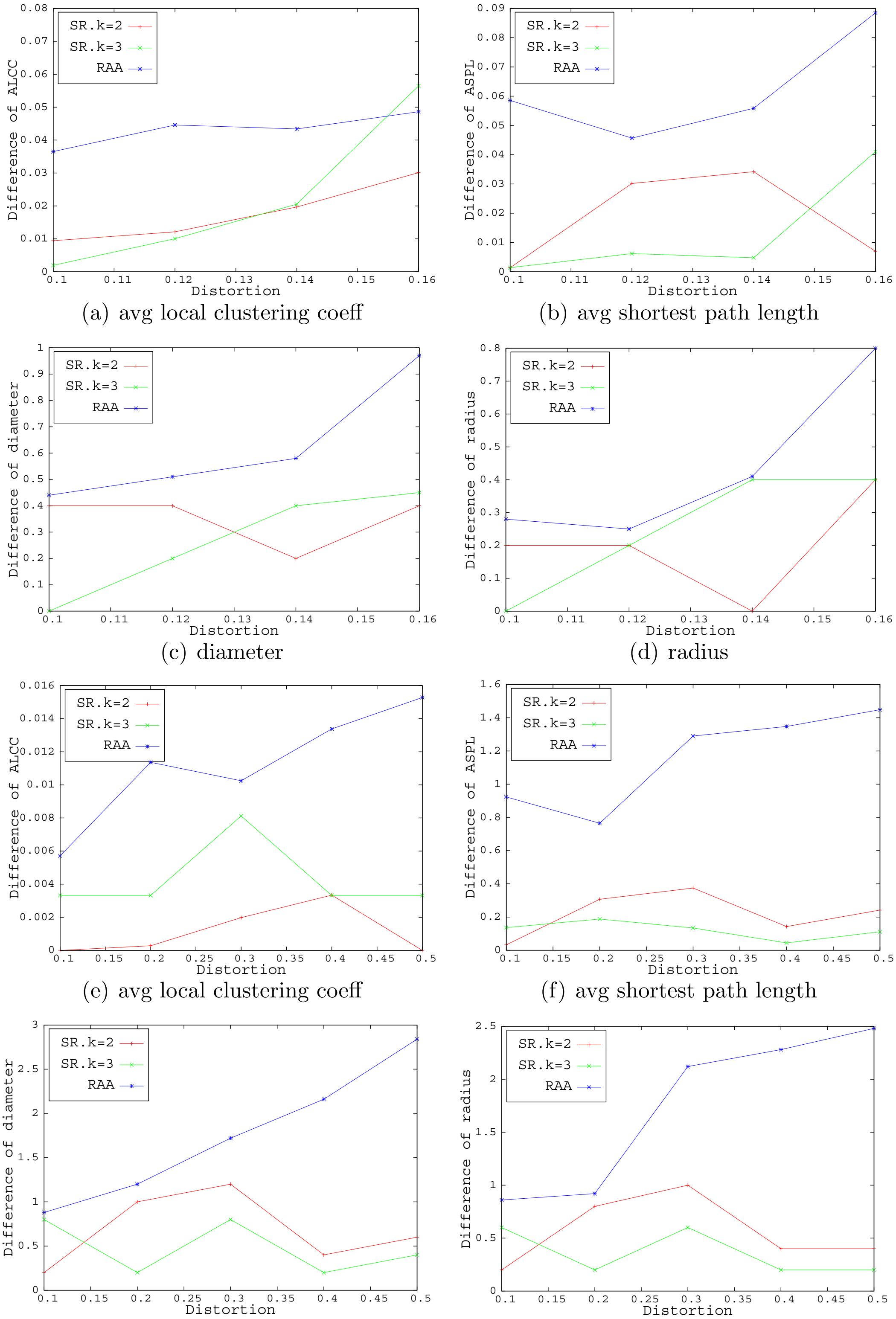
In particular, let V_o (V_m) be the set of vertices within k hops of the querying node in the original (modified) graph. The precision $\mathcal{P}$ and recall $\mathcal{R}$ are measured as follows:

$$\mathcal{P} = \frac{|V_o \cap V_m|}{|V_m|} \quad \mathcal{R} = \frac{|V_o \cap V_m|}{|V_o|} \tag{3.1}$$

Similarly, our *false negatives* and *false positives* metrics are measured as:

$$\mathcal{FN} = \frac{|V_o \setminus V_m|}{|V|} \quad \mathcal{FP} = \frac{|V_m \cap V_o|}{|V|} \tag{3.2}$$

where V is the graph's complete vertex set. We measure each of these metrics on each vertex and average our results over all vertices in the graph. Figure 3.23 shows our results with both the Flickr and Gnutella data, for graphs modified by the SRG and RAA algorithms, for queries involving number of hops $k = 2$ and $k = 3$. For example, each dot on the red line in Figure 3.23(a) represents the average precision for 2-hop queries. As in our previous measurements of graph properties, all results are averaged over 5 extracted subgraphs and 5 runs for each subgraph, so as to diminish the effect of randomness. In all examined cases, the SRG algorithm achieves higher precision and recall measures, and lower false negatives and positives, than RAA; the difference is more conspicuous with the Gnutella data. This outcome reconfirms that the SRG algorithm preserves reachability information more accurately than random distortion does, which is exactly the aim this algorithm is made for.



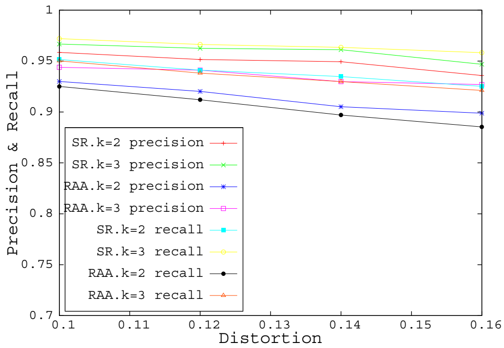
(a) avg local clustering coeff (b) avg shortest path length

(c) diameter (d) radius

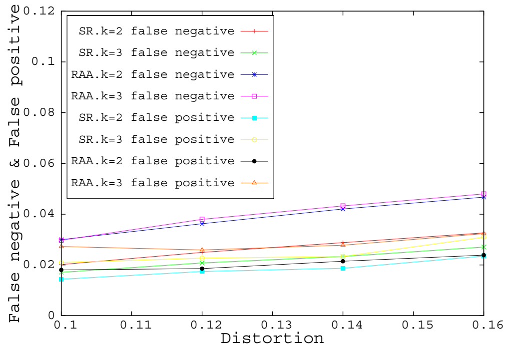
(e) avg local clustering coeff (f) avg shortest path length

(g) diameter (h) radius

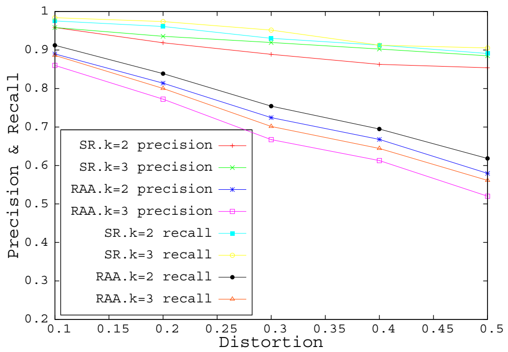
Figure 3.22: Graph properties with increasing distortion, Flickr (a-d) and Gnutella (e-h)



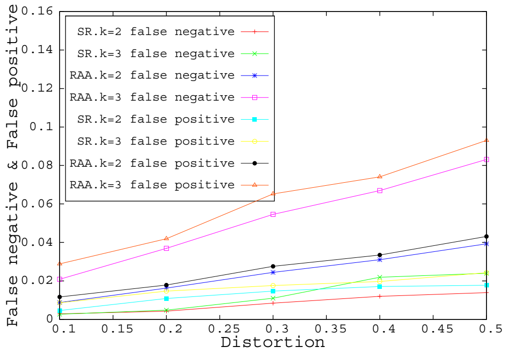
(a) precision and recall



(b) false negatives/positives



(c) precision and recall



(d) false negatives/positives

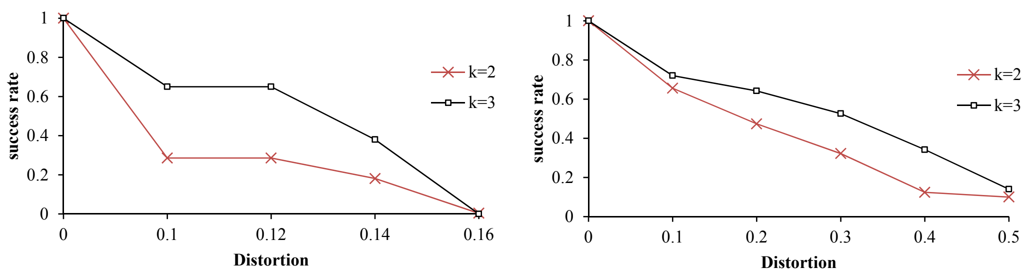
Figure 3.23: Precision and Recall, False negatives and False positives, Flickr (a-b) and Gnutella (c-d)

3.2.3.3 Resistance to Structural Attacks

We now turn our attention to assessing the extent to which are graphs can resist attacks based on an adversary's structural knowledge. The resistance to such attacks ensures that the network's structure is released in a way that does not allow the inference of individual users' identity, while at the same time providing the utility that we expect, as we have witnessed in the previous section. We contend that the graphs released by our method are capable of withstanding structural identification attacks with a high probability, hence providing a measurable amount of protection on that front.

To illustrate this protection, we experimentally measure the extent to which our distorted graphs can resist structural attacks of the kind suggested in [7]. While [7] proposed a specific attack algorithm, the *walk-based attack*, we go one step further and measure the success rate for *any* attack based on the identification of an embedded subgraph in the distorted graphs, as a function of the amount of distortion incurred on it. Such a structural attack is assumed to succeed *if* the adversary can identify an embedded graph in the released graph; as we have discussed, such an embedded graph may consists of fake accounts created before graph releasing and which are connected among themselves and to other, victim nodes, so as to follow a unique and identifiable pattern.

The identification of the maliciously embedded subgraph depends on the information of degree and internal structure. Intuitively, the more distorted a graph is, the less likely it becomes that a structural attack will succeed, and hence higher protection of individual users is afforded. Besides, the more distorted a graph is, the less it can be relied upon to provide truthful information at its microstructure. Arguably, a graph that presents high distortion at its microstructure while still maintaining truthful overall structural properties at its macrostructure would satisfy our purposes. On the other hand, in case all edges in the embedded subgraph are preserved after the transformation process and no others are added, then the attack can be launched successfully. We contend that this state of affairs rarely arises and its likelihood drops with increasing distortion.



(a) Flickr (b) Gnutella

Figure 3.24: Success rate of structural attack

We measure the *success rate* of the described structural attack vs. the distortion of the graph in which a malicious subgraph has been embedded. For each data set, we embed 50 different subgraphs prior to the graph's distortion. For each of the resulting *attacked* graphs, we conduct 10 separate runs of SRG perturbation, where we randomly shuffle the order in which edges are examined so as to obtain non-deterministic results; thus we obtain 10 different distorted versions of the original attacked graph, at the same desired distortion level. The success rate of the attack on the original data, for the obtained distortion, is measured as the total ratio of successful attacks over the total 10×50 runs.

Figure 3.24 (a) and (b) shows our results, for the Flickr and Gnutella data sets, respectively, and for two different values of the reachability parameter k. Our results confirm our expectations: as distortion grows, it becomes harder for the attack to succeed. Remarkably, we obtain low success rates even at distortion levels in which, as our utility assessment experiments show, we also preserve structural graph properties with satisfactory fidelity.

3.2.4 Discussion

Previous suggestions on discretionary social network publication follow a similar format: they define a privacy principle the published graph should obey, and proceed to alter the given graph so as to satisfy this constraint. Still, they do not offer a respectively comprehensible utility guarantee; as a consequence, they do not focus on providing data that a social network user may find useful. After all, such techniques are designed with the assumption that the whole-network data is published to an external data recipient, e.g., a researcher; their aim is not to enable user-centric revelation of network subgraphs to SNS users themselves.

We suggested a user-oriented alternative, aiming to provide a picture of a subgraph of interest that preserves certain structural properties, thereby offering a *utility guarantee*. The subgraph consists of an end-user, as a central user, and a neighborhood of other users that the central user is interested in. In this case, users' confidentiality is catered for by distorting the graph as far as possible under a utility constraint. Unlike the common scenario in the literature, our method is mainly designed for daily usage scenarios where SNS users want to assess their position among their peers and their ability to expand their online network in a desired direction. This facility would be especially useful for users that subscribe to SNSs with the aim of expanding their social or professional circles. Users who value such information would be able to opt in such an information revelation scheme in a give-and-take manner, as they would also be willing to disclose some of their own information to gain from the networking potential the scheme provides.

The graph is distorted in some respects by our approach, so as to forestall attacks by adversaries with structural knowledge, yet it preserves certain topological properties in other respects. Arguably, our methodology facilitates certain desirable user behaviors. Eventually, we argue that we can promote networking in an SNS by benevolent users, while protecting such users from malicious attackers aiming to exploit the same information to undesirable ends.

Our results show that by using our proposed approach, not only is reachability information guaranteed, but other structural properties are also preserved, which means that users can be provided with views of their extended neighborhood that will be representative of the real network, even if these are somewhat distorted in order to thwart malicious users. We envision that such views could consist of abstracted visual representations of one's extended neighborhood, e.g. in the form of concentric circles, or clouds that will indicate reachability and overall structure, so that a user may assess the distance to another user of interest, as well as the density, complexity, clustering and other structural properties of the network neighborhood. Our method allows for the creation of various such network views that would be beneficial to the benevolent user. For instance, one looking at the graph formed by one's friends can accurately infer how tightly connected those friends are with each other; for example, a large diameter implies one's social connections are wide spread, while a small one implies that one is connected only to people already well-connected with each other. Such information may be of particular utility to a public personality (election candidate, actor, athlete) visualizing ones fan club. Alternatively, someone using a network to promote their work (e.g., a musician) may be interested in identifying the most influential nodes in that network, and the number of such nodes. The preservation of properties such as degree distribution is instrumental for that purpose.

3.2.5 Summary

This method addresses the problem of social network data sharing under confidentiality concerns, from a utility-oriented standpoint, focusing on revealing a subgraph of connections in a user's neighborhood. We defined a utility guarantee involving a reachability property and suggested a method to distort the graph to a desired extent while observing this requirement. Our technique preserves crucial properties while blurring individual linkages; thus, it offers a perturbed, albeit informative, view of the network. Our experimental study confirms that (i) graphs obtained with our scheme *do* preserve large-scale structural properties of the original graphs more faithfully than graphs that have undergone the same amount of distortion by random perturbation, while (ii) they also pose satisfactory resistance to structural attacks.

Chapter 4

Community Detection

In the previous chapter we study graph structure for network data privacy problems. We propose novel algorithms for solutions. In this chapter, we study the utility of network data, the achievements of which are expected to benefit various applications, i.e., building of weak social ties and bridging social capital. To do this, we focus on the analysis that explores connectedness on graphs. Specifically, we study the graph structure for the problem of structural community detection.

One of the building blocks of community detection, be it in application domains such as sociology [56], biology [138, 54], politics [141] or marketing [128] where the underlying connectivity structure can be naturally thought of as a graph, is the ability to recognize clusters of vertices. From this strict structural point of view, communities correspond to sets of vertices that are more densely interconnected to each other than they are to the rest of the graph. The structure of a community help us to understand the individual behaviors and information diffusion in social networks.

We model a network as a simple graph $G(V, E)$. We propose three approaches that detect communities based a wide range of graph structural properties, including degree and clustering coefficient, and closeness among the vertices and the physical forces among the vertices when simulating the whole graph as a physical system.

4.1 Force-directed Layout Community Detection

4.1.1 Overview

The main idea that underscores the process of finding communities in this work is to obtain a representation of the graph in a Euclidean space and then cluster the vertices based on the Euclidean distance. This is different from common graph clustering algorithms, in that most of them cluster the graph and detect communities directly or indirectly according to geodesic distance. We use Fruchterman-Reingold's force-directed algorithm (FR) [64].This graph layout approach transforms the connections among vertices, based on attractive forces and repulsive forces pulling vertices together and pushing them apart, respectively, into proximity in a Euclidean space.

This hints that the vertices within one community are placed relatively closer since it is denser inside the community. In other words, vertices within communities have more connections to each other than connections to the vertices in other communities. It is thus a good opportunity to adopt the techniques in data clustering to look for the communities based on the graph layout. Furthermore, we extend FR from two dimension to one dimension and three dimensions, and even higher dimensions as well. We evaluate the significance of the number of dimensions on our method's effectiveness and efficiency. For disjoint community detection, the data clustering technique that we take advantage of is the *k-means clustering (KM)*, while for overlapping communities, we employ the *Fuzzy C-mean clustering (FCM)*, which can indicate the strength between each vertex and communities, and thus does not restrict each vertex to belong only to one group. *FCM* is a variant of *KM*. All these algorithms' complexities are not high and neither is *FR*'s. Our method building on these techniques is thus efficient for large social networks.

We evaluate effectiveness by measuring *modularity*. For graphs with known community structures, we measure the precision as well, by comparing memberships to communities that our approach discovers with those to the known communities.

4.1.2 Background

The idea of *force-directed algorithms* is to achieve an "aesthetically pleasing" graph layout by simulating the whole graph as a physical system. Edges in the graph are seen as springs binding vertices. Vertices are virtually pulled closer together or pushed further apart according to physical forces. The positions of the vertices are adjusted, and this procedure continues until the the system comes to an equilibrium. In addition, Fruchterman and Reingold's force-directed algorithm [64] aims to achieve even vertex distribution. The authors define the attractive force and the repulsive force as $f_a(d) = d^2/k$ and $f_r(d) = -k^2/d$, where $k = C\sqrt{\frac{area}{number\_of\_vertices}}$, and d is the distance between every pair of vertices. *area* is the area size for display the graph.

K-means clustering [104] partitions objects to k clustering, assigns each object the cluster with the nearest mean and adjusts their membership until an optimum is reached. As a soft version of k-means, Fuzzy C-means clustering (*FCM*) [13] assigns a fuzzy degree of membership to each cluster to each object . Instead of belonging to only one cluster, objects classified via this algorithm can belong to several clusters with different strengths. As a general version of k-means, the expectation-maximization algorithm (*EM*) [5] models clusters using statistic distributions. The reason we adopt k-means, rather than *EM*, is that k-means is effective enough for this problem and k-means is more efficient. We experimentally show this in Section 4.

4.1.3 Algorithm

We propose an algorithm that can systematically enumerate all possible number of clusters and find the configuration with the highest modularity. Therefore, the algorithm iterates by changing the value of k from 1 to $|V|$ which is the number of vertices in the network. We show the changes of modularity with a change in k values. If the number of clusters is prior knowledge, we can set the number of iterations to be 1 to this number.

Algorithm 4.1: Force-directed Layout Community Detection Algorithm

Input: graph G with n vertices, the number of trials t, $t \leq n$;
Result: Clusters C_i, $i \in (1, 2, ..., k')$

$v = Fruchterman\_Reingold(G)$, $v \in R^{n*2}$, $v =[v_1;v_2;...;v_n]$;
Sort_degree(G);
$k \leftarrow 1$;
for *each* $k \leq t$ **do**
    $C'_i = K$-means(v);
    $C_i = Refinement(C'_i)$;
    Calculate modularity and record the maximum;
Return C_i,$i \in (1, 2, ..., k')$ with the maximum modularity;

Our method starts from the *FR* algorithm. The inputs for the algorithm are limited to only the graph edges. The output is the coordinates of vertices in Euclidean Space. Then we sort the degrees of the vertices and initialize the centers of the clusters for the clustering using the vertices with highest degrees. The idea is that the vertices with a high degree have a higher chance of being the community centers. The centers may change during the clustering. We refine the clusters after the data clustering in Euclidean space. If there is any vertex that does not have any connection with other vertices in the same cluster, or it has fewer connections inside its cluster than outside its cluster, then it is grouped to the cluster where it has the maximum number of connections. In other words, this vertex is grouped to the cluster that has the greatest number of immediate neighbors. The refinement process may change the number of clusters, which is actually good for those who only roughly know the number of clusters. They can input the maximum number of clusters they believe exists, and let our method find out the exact number of clusters in the network without trying all the values of k from 1 to $|V|$.

Algorithm 4.2: Refinement

Input: Clusters C_i, $i \in (1, 2, ..., k)$;
Result: Clusters C'_i, $i \in (1, 2, ..., k')$;

for i *from 1* **to** k **do**
  for $v \in C_i$ **do**
    find the cluster C_j where v has the maximum number of immediate neighbors;
    if $i \neq j$ **then**
      Cluster v into C_j;
Return C'_i,$i \in (1, 2, ..., k')$;

We call the above algorithm *FR-KM* for the experiments. The other two versions of the algorithm are similar to *FR-KM* but depend on different clustering methods. We name the one using the expectation-maximization algorithm *FR-EM* and the one using the fuzzy c-means algorithm *FR-FCM*. For *FR-FCM*, there's no refinement of the memberships for the vertices, since we intend to deal with overlapping communities.

4.1.4 Experimental Evaluation

We conduct experiments on both synthetic and real world graphs including two benchmark graphs for the community detection algorithm. The experiments run

Table 4.1: Performance Comparison between *FR-EM*,*FR-KM* and *GN*

	KarateClub		AmericanFootball		EmailURV	
	modularity	running time	modularity	running time	modularity	running time
GN	0.4013	0.016	0.5976	1.014	0.5323	3193.532
Walktrap	0.3944	0.0000001	0.6015	0.015	0.5250	0.92
InfoMap	0.402038	0.015	0.599176	0.047000	0.521420	5.912000
FR-KM	0.417406	0.020000	0.601731	2.179000	0.542659	15.388000

on an Inter Core, 2 Quad CPU, 2.83GHz, 2GB machine running Windows 8 OS. The algorithms are implemented in C.

4.1.4.1 Data Sets

We use a batch of benchmark graphs [92] to evaluate the effectiveness of our method. The real-world benchmark graphs we use are Zachary's Karate Club data and American College Football data (see Section 4.2.3.1). We also test on the Email-URV data set, Wikipedia data set, and Facebook data set (see Appendix A.4). They represent large online social network data.

4.1.4.2 Analysis of non-overlapping community detection

We compare our method to the algorithms of Girvan and Newman(*GN*) [69, 114], which is one of the state-of-the-art algorithms in community detection. We also compare our method with *Walktrap* algorithm [123] and *InfoMap* algorithm [129], which has been shown to perform quite well for community detection (see [62]).

Table 4.1 shows the performance of the algorithms. In this comparison, we use the normal two dimension *FR* algorithm with 400 iterations for KarateClub and AmericanFootball data and 1000 for Email-URV data. The number of trials is set to 30. For all three graphs, our method produces partitions with the highest modularity among the four algorithms. Although *Walktrap* and *InfoMap* are faster than our method, and *GN* is faster than our method for smaller graphs, the running

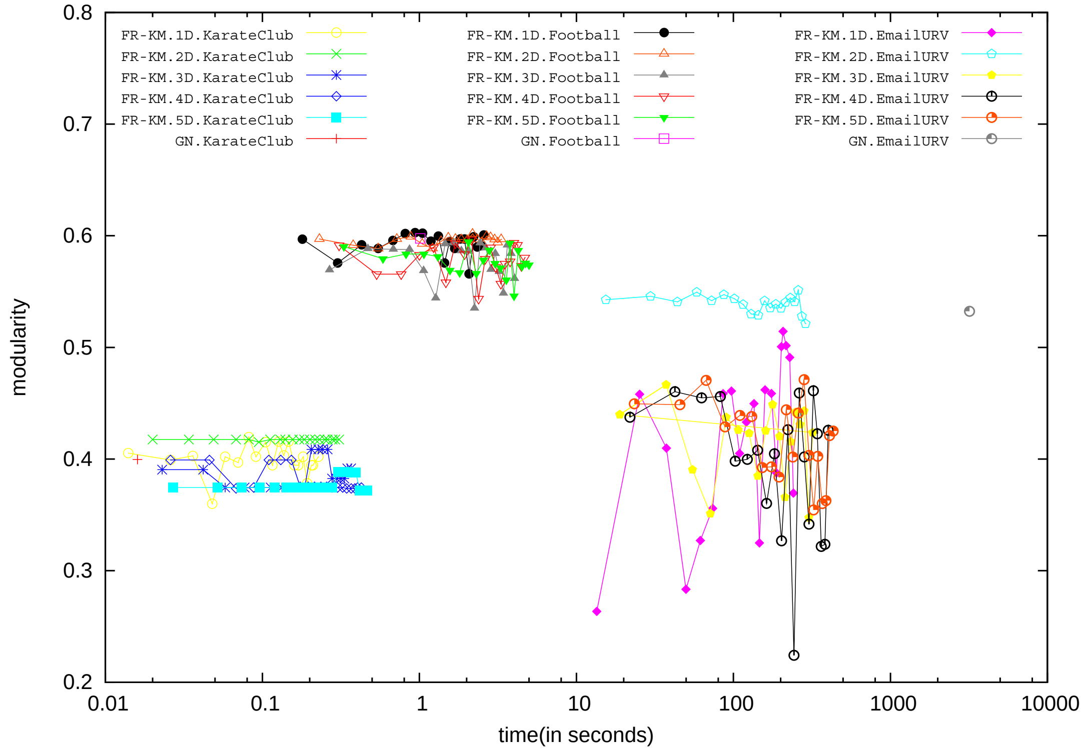
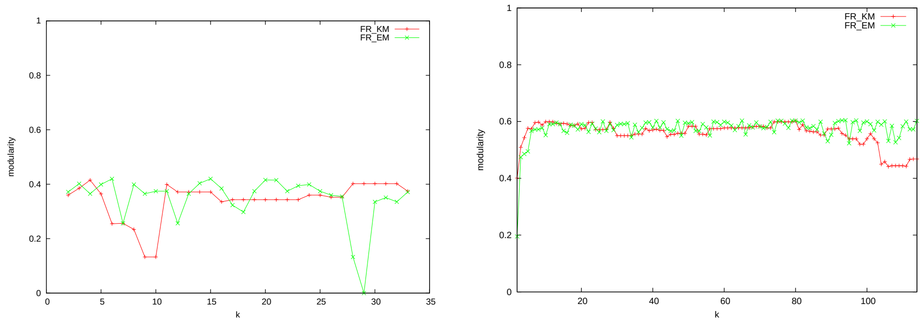


Figure 4.1: Performance Comparison between multiple dimension *FR-KM* and *GN*

time of our method is still tolerable. As the size of graph becomes larger, our method becomes faster. If the number of clusters is known in advance, then the number of trials is 1, instead of the 30 that we set. If so, our method takes much less time. *GN* is much slower for larger graphs. For the other two real-world data sets, Wiki-Vote and Facebook, we are unable to make the comparison due to *GN*'s scalability, but we will show the running time of clustering these two graphs by our method.

Figure 4.1 shows the performance comparison between multiple dimension *FR-KM* and *GN*. We extend the normal two dimension FR algorithm to one dimension and three, four, and five dimensions. We set the number of trials as 30. For karate club data, the number of trials is equal to its number of vertices. We run each *FR-KM* with the number of iterations of *FR* changing from 100 to 2000 with an interval



(a) Karate Club data (b) Football data

Figure 4.2: Modularity for varying number of clusters

of 100. We find that the larger the number of iterations of *FR-KM*, the longer time it takes. However, the number of iterations of *FR* does not have a decisive influence on the modularity. This suggests that there is no need to increase the number of iterations to get higher modularity. In terms of dimension, we find that for small graphs, projecting them to one dimension or three dimensions may get a higher modularity sometimes, but for large graphs, the two dimension *FR-KM* performs best. It is faster, and clusters graphs with higher modularity. That is why we shall adopt the normal two dimensional *FR* in our algorithm when it comes to large graphs. *FR-KM* outperforms in both effectiveness and efficiency with large graphs compared with *GN*.

Figure 4.2 shows the modularity when the initial input number of clusters k varies. The final number of clusters may be different from the values of k on the x-axis here. Our method changes the number of clusters during cluster refinements, which produces a local optimum number of clusters. Therefore, we can see from the result that the trend of the line is horizontal in general. This suggests that we can find a local optimum around initial k value even without knowing the number of clusters beforehand. This local maximum is probably the global optimum or close to the global optimum.

Figure 4.3 shows the running time for varying number of clusters for Email-

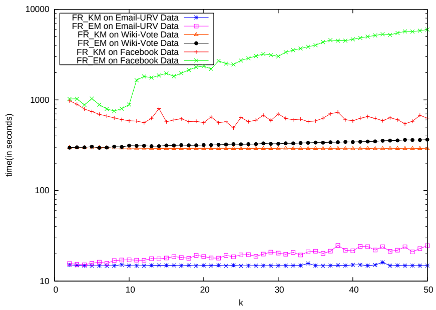


Figure 4.3: Running time for varying number of clusters for Email-URV, Wiki-Vote, and Facebook data set

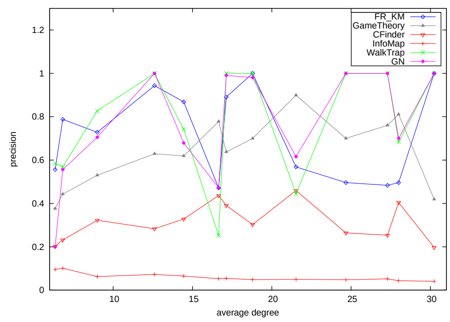


Figure 4.4: Precision for varying average degree of synthetic graphs

URV data, Wiki-Vote data and Facebook data. For each data set, the time for projecting the graph onto Euclidean space is the same, but the clustering time differs. *KM* running time remains the same in general as the initial number of clusters increases while *EM*'s running time linearly increases as the initial number of clusters increases. Compared with *KM*, *EM* takes much more time. The trends are similar among the results for the three data sets.

We compare our method with *GN*, *InfoMap* and *WalkTrap* algorithm, and two other community detection algorithms, CFinder ([67]) and the game-theory algorithm ([32]). Figure 4.4 shows the precision achieved by the algorithms on the generated graphs with different average degrees. Since the community structures are known, precision is obtained by counting the number of correctly clustered vertices. The results show that our method outperforms the CFinder, *GN* and *InfoMap*, and produces results comparable with the game-theory algorithm and *WalkTrap*. The reason for CFinder having a low precision may be that not every vertex in the graph is clustered. The clusters consists of 3-cliques only in our experiment. The reason for *InfoMap* having the low precision may be that the number of community this method detects is large and most of the communities are of a small size. Many communities are of size of two vertices only.

4.1.4.3 Analysis of overlapping community detection

In Figure 4.5 we show that by integrating soft clustering algorithm, *FCM*, in our algorithm, we find that the two communities are overlapping on four vertices (vertex 3, 9, 10, 31) that are marked both in green and blue. This is under the assumption that if the membership strength is larger than 0.8, then the vertex belongs to that cluster only. Figure 9 shows the membership strength of each vertex to each cluster. If we take 0.7 as a threshold for belonging strength, then the clusters will have more overlapping vertices. [91] points out that vertex 3, 9, 10, 14 and 31 are often misclassified by traditional algorithms. We believe that vertex 3, 9, 14 and 31 are shared between the two groups if overlapping is allowed.

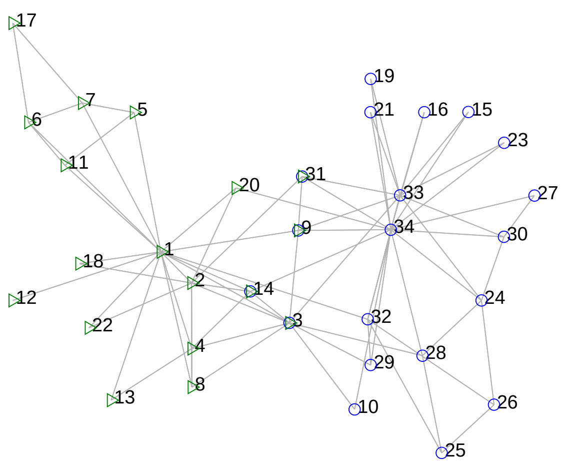


Figure 4.5: Zachary's Karate Club data partitioned into overlapping clusters

vertexID	C_1	C_2
1	0.996227	0.003773
2	0.940114	0.059886
3	0.598883	0.401117
4	0.882118	0.117882
5	0.929731	0.070269
6	0.823111	0.176889
7	0.863318	0.136682
8	0.809490	0.190510
9	0.473149	0.526851
10	0.344961	0.655039
11	0.865513	0.134487
12	0.914518	0.085482
13	0.877715	0.122285
14	0.771624	0.228376
15	0.045550	0.954450
16	0.045550	0.954450
17	0.751788	0.248212
18	0.931399	0.068601
19	0.223629	0.776371
20	0.797011	0.202989
21	0.045550	0.954450
22	0.974779	0.025221
23	0.045550	0.954450
24	0.023890	0.976110
25	0.061673	0.938327
26	0.045550	0.954450
27	0.045550	0.954450
28	0.157053	0.842947
29	0.293805	0.706195
30	0.045550	0.954450
31	0.427176	0.572824
32	0.151637	0.848363
33	0.022016	0.977984
34	0.029300	0.970700

Figure 4.6: Membership strength

4.1.4.4 Complexity

Our method's ability to work with large social network data contributes to low complexities in the basic techniques we utilize. *FR*'s complexity is $t(\mathcal{O}|E| + \mathcal{O}(|V|^2))$, where $|E|$ is the number of edges, $|V|$ is the number of vertices and t is the number of iteration. Time complexity of k-means clustering is $\mathcal{O}(kt|V|)$, where $|V|$ is the number of vertices and t is number of iterations. Time complexity of refinement is $\mathcal{O}(d|V|)$, since we check each vertex's immediate neighbors, the number of which is, at most, $|V| - 1$. d is the max degree of the vertices. Therefore, for the whole algorithm, time complexity is $t \cdot (\mathcal{O}|E| + \mathcal{O}(|V|^2)) + \mathcal{O}(kt|V|) + \mathcal{O}(d|V|)$. For large networks, $|V|$ and $|E|$ are the most significant elements that affect time complexity.

4.1.5 Summary

We propose a graph-layout based community detection algorithm. We use Fruchterman-Reingold algorithm to project the graph onto a Euclidean space and we cluster the vertices according to their Euclidean distance. Then we refer to the original graph information to refine the communities detected. We evaluate the effectiveness and efficiency on both real-world data and synthetic data. For disjoint community detection, the results show that *FR-KM* is more effective on both small graphs and large graphs than *GN*, and is much more efficient than *GN* on large networks. *FR-KM* is also more effective than *Walktrap* and *InfoMap* algorithms, in terms of modularity. Compared with *GN*, *CFinder*, *InfoMap*, *WalkTrap* and *game-theory* algorithms on the synthetic graphs with known communities in advance, our method is more effective than *GN* and *CFinder* and has a good performance comparable to the *WalkTrap* and *game-theory* algorithms according to the results of precision testing. For overlapping detection, the result for Karate Club data shows that *FR-FCM* is reasonably effective.

4.2 Fast Disjoint and Overlapping Community Detection

4.2.1 Overview

The idea of this method is for each vertex to seek the community to which it belongs by visiting its neighbour vertices. Decisions are made based on the degrees, clustering coefficients of the neighbors, and the number of common neighbors. Degree and clustering coefficient are two importance properties of graph topology. Clustering coefficient measures the cliquishness of neighborhood, and thus indicates clustering in the graph locally [82, 155].

This method starts from a micro perspective, which is different from that of the previous work in the last section. Considering the size of networks in modern applications, we try to design a scalable method in order to deal with the large graphs within a reasonable time. Therefore, we try to minimize the number of pair-wise computations among vertices. Instead of comparing all pairs of vertices in a graph, we only explore each vertex's immediate neighbourhood. Indeed, vertices in the same community are more likely to be neighbours [70]. This significantly reduces the complexity except in the case of dense graphs. In our method, as vertices can independently explore their neighbourhood and join a community by following an immediate neighbour, the algorithms are intrinsically data parallel. We devise a parallel algorithm for disjoint community detection and implement it on a *Graphics Processing Unit* (GPU). In the case of overlapping community detection, a vertex is allowed to belong to several communities if strong connections exist between the vertex and any of those communities.

We empirically evaluate the performance of our algorithm with both real world networks and synthetic networks. We evaluate the quality of communities using metrics from different classes [165], as well as one metric recently proposed in [125]. The metrics include modularity, conductance, internal density, cut ratio, weighted community clustering, and *Normalized Mutual Information* [91]. The metrics indicate the community quality from different perspectives. We measure the efficiency by running time. We compare our algorithms with several state-of-the-art algorithms.

4.2.2 Algorithm

We propose an algorithm that delegates the job of finding communities to individual vertices. Each vertex seeks its community independently. The decisions of which community to join are made based on the degrees and clustering coefficients of neighbours, as well as on the number of common immediate neighbours. We hypothesize that vertices tend to join groups with more connections. In other words, the vertices try to attach themselves to dense structures, i.e. structures with more connections among vertices in this structure.

4.2.2.1 Fast Disjoint Community Detection

The algorithm starts by calculating the degree and local clustering coefficient for each vertex (line 1). The local clustering coefficient is defined as

$$cc[i] = \frac{e_{jk} : j, k \in V, e_{jk} \in E}{degree[i] * (degree[i] - 1)}$$

It is the ratio between the number of edges between vertices within its neighborhood and the number of edges that could possibly exist between them. It quantifies how closely the vertex connects with its neighbors.

Then each vertex looks around its immediate neighbours. If the degree of the

Algorithm 4.3: Fast Community Detection

Input: graph $G(V, E)$ with $|V|$ vertices, $|E|$ edges;
Result: Clusters C_i, $i \in (1, 2, ..., k')$

```
Compute degree[v] and cc[v], v ∈ V;
for each v do
    if degree[v]<degree[v_j] then                    /* v_j ∈v_neighbour */
        g[v] ← v_i, where degree[v_i] = max(degree[v_j]) ;
    else
        g[v] = v;
for each v do
    if g[v] = v and degree[v] = degree[v_i] then
        if v and v_i has more than half common vertices;
        then
            g[v] ← v_i, if v_i has smaller id;
    else
        v_g ← g[v];
        c1 ← number of common neighbours between v and j;
        c2 ← number of common neighbours between v and (v_neighbour \ v_g);
        if c1 < c2 then
            g[v] ← v_i, where degree[v_i] = max(degree[v_j]), v_j ∈(v_neighbour \ v_g)
for each v do
    if g[v] ≠ v then
        i ← g[v];
        repeat
            i ← g[i];
        until g[i] = i; find standalone vertex
        g[v] ← i;
k ← different numbers in g[v];
for i from 1 to k do
    for v ∈ C_i do
        find the cluster C_j where v has the maximum number of immediate
        neighbours;
        if i ≠ j then
            Cluster v into C_j;
Return C_i,i ∈ (1, 2, ..., k');
```

vertex, for example vertex v, is the largest among its immediate neighbours, vertex v stands alone and does not follow other vertices. If the degree of vertex v is not the largest among its immediate neighbours and itself, vertex v follows the neighbour with the largest degree among v's immediate neighbours (line 2-6). If more than one vertex among the immediate neighbours have the largest degree, then vertex v follows the one with the largest clustering coefficient, compared to other neighbours.

In the second round, each vertex adjusts their decisions (line 7-17). If the standing-alone vertex v has neighbours with the same degree, check the number of common neighbours of vertex v and v's neighbour that has the same degree. If there are enough common neighbours, these two vertices are assumed to be in the same community. If the vertex v does not stand alone but follows some neighbour, we check the number of common neighbours vertex v has with the vertex that it follows, and the number of common neighbours it has with the other neighbours. If vertex v has more common neighbours with its other neighbours than the one it follows, then vertex v turns to the vertex with the second largest degree in the neighbourhood or stands alone if it itself has the second largest degree.

In the third round, each vertex finalizes the community which it desires to join (line 18-24). If the vertex that vertex v follows is also following vertex v_i, then vertex v also turns to vertex v_i. In the end, each vertex follows a vertex that stands alone. With all the other vertices that follow this vertex, they form a community.

After each vertex chooses its community (line 25), we post-process the memberships to refine the communities (line 26-30). If any vertex has more connections outside the community than inside the community, it changes its membership. This refinement process may change the number of communities from the last step.

The only input of the algorithm is the graph itself. No pre-defined number of

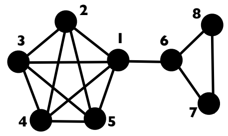


Figure 4.7: Example

communities is needed. In the experiments, the graph is given as an edge list. The output is the communities.

Figure 4.7 shows a graph with 8 vertices and 14 edges. After the first round, vertex 2, 3, 4, 5, 6 all follow vertex 1 (g[1]=1, g[2]=1, g[3]=1, g[4]=1, g[5]=1, g[6]=1), while vertex 7 and 8 follow vertex 6 (g[7]=6, g[8]=6). In the second round for each vertex, the status of vertex 1 is unchanged. The status of vertex 2, 3, 4, 5 is also unchanged, because they have more common neighbours with vertex 1, that they follow, than with other vertices ($\{$vertex 2, 3, 4, 5$\}\backslash$themselves), vertex 7 and 8 still follow 6, while vertex 6 changes to stand alone instead of following vertex 1 because vertex 6 has more common neighbours with 7 and 8 than with vertex 1. No more changes happen in the third round and the refinement, and thus, the final result is that we find two communities: one community is labelled by vertex 1, and has vertex 1, 2, 3, 4, 5; the other community is labelled by vertex 6, and has vertex 6, 7, 8.

We also devise a parallel version. Both the first and second rounds are parallelized. In the first round the vertices look for the vertex with the largest degree in the neighbourhood at the same time. In the second round, each vertex makes a decision concurrently. The rest of the algorithm is sequential.

4.2.2.2 Fast Overlapping Community Detection

For the case of overlapping communities, we extend *FCD* with modifications in the second round and post-processing, with an additional input parameter θ.

In the second round, each vertex adjusts its decision (line 7-16). If the vertex v

does not stand alone but follows some neighbour, and vertex v has more common neighbours with its other neighbours than the one that it follows, then vertex v turns to stand alone, so that vertex v leaves the opportunity of finding its communities to the post-processing part. This aims to cluster controversial vertices after other vertices choose their communities, and therefore there are clear local pictures for the controversial vertices to make decisions.

When post-processing the memberships to refine the communities (line 25-29), the number of connections of each vertex v with each cluster is counted. N_i^v is the number of immediate neighbours that v has in C_i, representing the number of connections. For any vertex v, N_{max}^v equals $max(N_i^v)$ where $1 \leq i \leq k$ (line 27). It is the maximum number of immediate neighbors of vertex v that it has with some cluster. Each vertex is grouped into the cluster with the most connections, and the clusters that have significant number of connections compared with the maximum number, satisfying the criteria of $N_{max}^v - N_i^v \geq \theta$. The parameter θ, overlapping factor, determines the degree of overlapping. If θ equals 0, vertex v is grouped to the clusters that have N_{max}^v connection with v. If θ equals 1, vertex v is grouped to the clusters that have N_{max}^v or N_{max}^v-1 connections with v. The larger θ is, the more clusters one vertex may be clustered into, and thus the more overlapping vertices there are. A vertex changes its membership if the community to which it currently belongs does not have enough connections with it. Note that overlaps may still exist if θ equals 0.

4.2.2.3 Complexity Analysis

The time complexity for calculating the clustering coefficient is $\mathcal{O}(n \cdot d^2)$, where n is the number of vertices and d is the average degree of vertices in the graph. The complexity for the first round is $\mathcal{O}(n \cdot d)$. The complexity for the second round is

$\mathcal{O}(n \cdot d^2)$. The complexity for the third round is $\mathcal{O}(n^2)$ in the worst case which is very unlikely to happen. The usually complexity for this part is $\mathcal{O}(\alpha \cdot n)$ where α is smaller than the graph diameter generally and presents a value less than 2 in our experiments. The complexity for the refinement is $\mathcal{O}(n \cdot d^2)$. Therefore the time complexity for the whole algorithm is $\mathcal{O}((d^2 + \alpha) \cdot n)$ in the worst case. For the parallel version, the complexity for the first round is $\mathcal{O}(d)$. The complexity for the second round is $\mathcal{O}(d^2)$. The rest is the same as that of the sequential version. Thus the time complexity for the whole parallel algorithm is $\mathcal{O}(d^2 + \alpha \cdot n)$ in the worst case.

The two algorithms can be applied to the networks according to the preliminary knowledge of communities, e.g. whether they are disjoint or overlapped.

4.2.3 Experiment

We conduct experiments on both synthetic and real world graphs, including three benchmarks for community detection. We ran the sequential algorithms on an 2.83GHz Inter Core, 2 Quad CPU machine with 2GB of main memory under Windows 8 OS. The parallel algorithm ran on the same machine with a GeForce GTX 560 Ti graphics card having 2048 MB of global memory, 8 multiprocessor and 48 CUDA cores per multiprocessor. The algorithms are implemented in Visual C++ 10.0. The parallel algorithm is implemented using the application programming interface CUDA for the C language. CUDA [42], the C language Compute Unified Device Architecture, is provided by NVIDIA and works on NVIDIA graphic cards. The CUDA programming model consists of a sequential host code combined with a parallel kernel code.

We compare our algorithm for disjoint community detection with three state-of-the-art algorithms: *InfoMap* [129], *WalkTrap* [123] and Girvan and Newman

Algorithm 4.4: Fast Overlapping Community Detection

```
Input: graph $G(V, E)$, parameter $\theta$;
Result: Clusters $C_i$, $i \in (1, 2, ..., k')$
Compute degree[$v$] and cc[$v$], $v \in V$;
for each $v$ do
    if degree[$v$]<degree[$v_j$] then                /* $v_j \in v_{neighbour}$ */
        g[$v$] ← $v_i$, where degree[$v_i$] = max(degree[$v_j$]) ;
    else
        g[$v$] ← $v$;
for each $v$ do
    if g[$v$] = $v$ and degree[$v$] = degree[$v_i$] then
        if $v$ and $v_i$ has more than half common vertices;
        then
            g[$v$]← $v_i$, if $v_i$ has smaller id;
    else
        $v_g$ ← g[$v$];
        $c1$ ← number of common neighbours between $v$ and $j$;
        $c2$ ← number of common neighbours between $v$ and ($v_{neighbour} \setminus v_g$);
        if $c1$ < $c2$ then g[$v$] ← $v$;
for each $v$ do
    if g[$v$] ≠ $v$ then
        $i$ ← g[$v$];
        repeat
            $i$ ← g[$i$] ;
        until g[$i$]= $i$; find standalone vertex
        g[$v$] ← i;
$k$ ← different numbers in g[$v$];
repeat
    for each $v$ do
        find clusters $\{C_i | N^v_{max} - N^v_i \geq \theta, 1 \leq i \leq k\}$;
        if $v \notin C_i$ then Cluster $v$ into $C_i$;
until reach equilibrium;
Return $C_i$,$i \in (1, 2, ..., k')$;
```

(GN)[69][114]. *InfoMap* is based on information theory. *Walktrap* is based on random walk. *InfoMap* has been empirically shown to have better performance, compared to other algorithms for community detection [62]. We compare our algorithm for overlapping community detection with two algorithms: the *game-theory* algorithm and speaker-listener label propagation algorithm ($SLPA$)[161], which show good performance [160, 161]. In the experiment, we directly use the original C++ code of the *game-theory* algorithm provided by author of [32] and Java executable file of *SLPA* provided by author of [161].

4.2.3.1 Data sets

We generate a batch of benchmark graphs [92] with known community structure, number of vertices, the average degree, maximum degree, minimum and maximum size of micro and macro community due to the hierarchical structure, and fraction of edges between vertices belonging to the same or different communities. The first set of graphs are generated with 2,000 vertices and different average degrees while the other parameters remain the same. They have no overlapping communities. For overlapping communities, we generate two sets of graphs. The first set of graphs has 10,000 vertices and different average degrees, while the other parameters are the same. Every five graphs have a similar average degree. We run the algorithm on all the graphs and we take and compare the average values. The second set of graphs generated have a varying number of vertices from 10,000 to 50,000, and for every number of vertices, five graphs are generated.

The real-world benchmark graphs used are listed as follows. Among them, Zachary's Karate Club data, American College Football data and Dolphin network are widely used for evaluating community detection algorithms.

Karate Club data is a social network of karate club members studied by the sociologist Wayne Zachary. The network has 34 members (vertices) and they are separated into two different groups due to a controversy between one of the instructors and administrator of the club.

American College Football data is a network with 115 teams (vertices) which are separated into 12 conferences. An edge exists between two vertices if there is a match between two teams. More games happen among teams within the same conference than teams from different conferences.

Dolphin Network is collected by David Lusseaua [105]. The network represents frequent associations between 62 dolphins (vertices) in a community living off Doubtful Sound, New Zealand.

Email-URV data is collected by Guimer et al. [1]. The network contains user-to-user (address- to-address) links from the network of e-mail interchanges among faculty and graduate students at Rovira i Virgili University of Tarragona, Spain. It is available on Alex Arenas Website [1].

Arxiv HEP-PH, collected by Leskovec et al. [96], is a collaboration network containing scientific collaborations between authors who submitted papers to High Energy Physics. It is available on the SNAP website [136].

Wiki-Vote, collected by Leskove et al. [95], contains user-to-user (who-vote-whom) links from the Wikipedia network. It is available on the SNAP website [136]. Each vertex represents a user. An edge is created from a user to a candidate if a user votes for Wikipedia admin candidates.

Email-Enron data set contains user-to-user (address-to-address) links. It was made public by the Federal Energy Regulatory Commission during its investigations. We obtained it from [136]. Each vertex represents an email address. An edge exists between vertex i and vertex j if address i sends at least one email message to address j.

Table 4.2: Description of data sets

	Number of Vertices	Number of edges
Karate Club	34	78
Dolphin	62	159
American College Football	115	610
Email-URV	1,133	5,451
Wiki-Vote	7,066	100,736
Arxiv HEP-PH	11,204	117,649
Email-Enron	33,696	180,811
Epinions	119,130	704,276

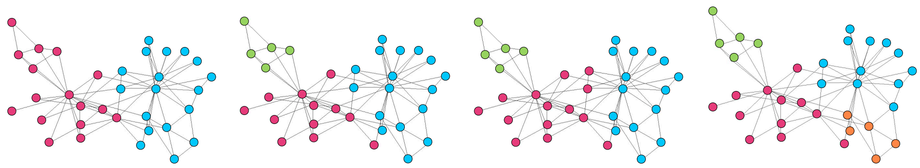
Epinions data set contains user-to-user (who-trust-whom) links from Epinions network. It was collected by Epinions staff P. Massa. We obtained it from *trustlet* website [146, 108]. Each vertex represents a user. An edge corresponds to a trust or distrust statement from one user to another user.

We extract the largest component of the networks that have more than one component. The number of vertices and the number of edges of each data set are listed in Table 4.2

4.2.3.2 Metrics

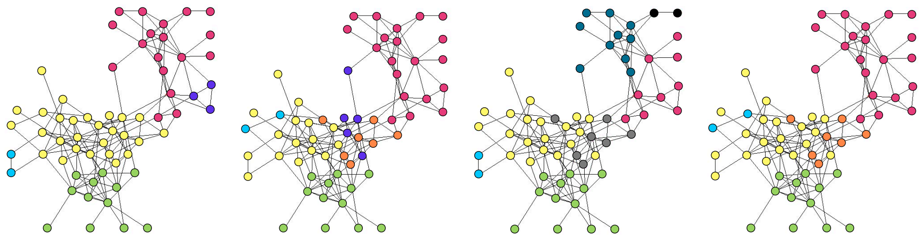
We use five metrics to qualify the disjoint communities: modularity, conductance, internal density, cut ratio and weighted community clustering (Equ. 2.1 - 2.6). Modularity, conductance, internal density and cut ratio are selected from four classes of metrics for community [165] so that we can eliminate the bias of having only one kind of metric. Weighted community clustering is a recently proposed metric [125]. We use the revised modularity (Equ. 2.2) to measure the quality of overlapping communities. In our experiments, we take the average of the conductances of communities found for the conductance of the whole network, and it is the same for the other metrics, except modularity.

We compute the *NMI* (Equ. 2.7) value of the set of communities detected, and the known set of communities of the graphs that we generate. *NMI* works



(a) FCD (b) InfoMap (c) WalkTrap (d) GN

Figure 4.8: Communities for Karate Club data by different algorithms



(a) FCD (b) InfoMap (c) WalkTrap (d) GN

Figure 4.9: Communities for Dolphin data by different algorithms

in the same way for the comparison of disjoint community sets and overlapping community sets.

4.2.3.3 Experimental Assessment for Disjoint Community Detection

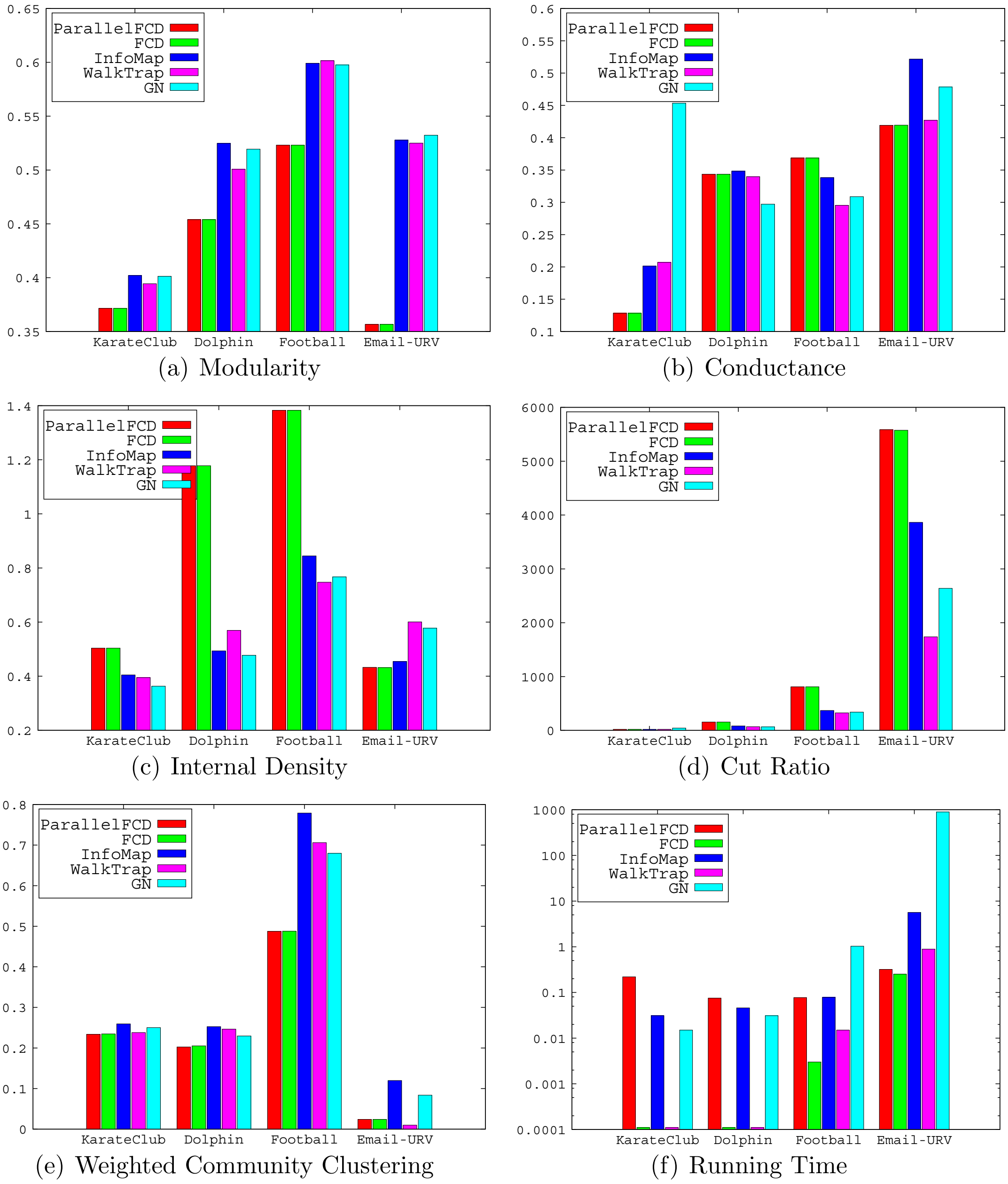
Figure 4.8 shows the communities found in the Karate Club network by each algorithm. Figure 4.9 shows the communities found in the Dolphin new network by each algorithm. Vertices of the same color are in the same community.

Figure 4.10 shows the measurement results on the four real data sets. The x-axis is labelled by the names of data sets. The y-axis is the value of metric. For each data set, the metric values for the communities detected by each algorithm are compared. Figure 4.10 (a) shows that the communities that FCD and ParallelFCD found have a lower modularity on these four data sets. However, this does not indicate that our algorithm is not better than the other three algorithms. Figure 4.8 shows that our algorithm identifies two communities, that coincides with the truth that the members of the Karate Club separated into two different groups due to a controversy, and thus the result of our algorithm is actually more reasonable than the other three algorithms even though the modularity values are lower. Figure 4.10

(b) shows the conductance results. The lower the conductance, the better the communities found. In this case, our algorithm has the lowest conductance on two data sets and highest conductance on the other two data sets. Figure 4.10 (c) shows the internal density results. The higher the internal density, the better the communities found. In this case, our algorithm has the highest internal density in three of the four data sets, and the lowest in one data set. Figure 4.10 (d) shows the cut ratio results. The lower the cut ratio the better the communities found. In this case, our algorithm has the lowest cut ratio in one of the four data set, and the highest in the other three data sets. Figure 4.10 (e) shows the weighted community clustering results. The higher the *WCC*, the better the communities found [125]. In this case our algorithm has a lower *WCC* in three of the four data sets. Figure 4.10 (f) shows the running time. For the four data sets, *FCD* performs the fastest among the algorithms. *ParallelFCD* performs faster than *InfoMap*, *WalkTrap* and *GN* on the Email-URV data. Comparing the performances of the same algorithm on the four data sets, we can see the big differences which are due to the different graph structures, e.g. different number of vertices, number of edges, different densities.

To sum up the results on these four real data sets, our algorithm, *FCD* and its parallel version, finds communities with better values in terms of internal density and conductance, but not with the other metrics. However, as we can see from the results for Karate Club, the communities detected by our algorithm stay more truthful than those of the other algorithms. In this sense, our algorithm is effective. From the comparison of running time, *FCD* is obviously more efficient than the others.

Figure 4.11 shows the results on the first set of benchmark graphs. It shows that the metric value changes as the graphs increase in average degree. The x-axis is the average degree of the graphs. The y-axis is the value of metrics. Each dot



(a) Modularity

(b) Conductance

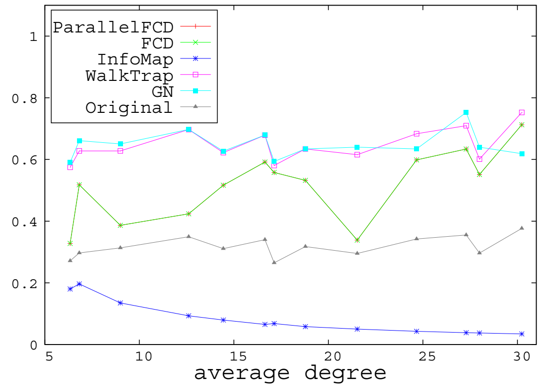
(c) Internal Density

(d) Cut Ratio

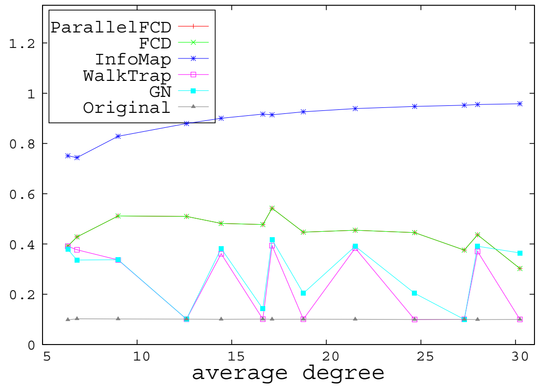
(e) Weighted Community Clustering

(f) Running Time

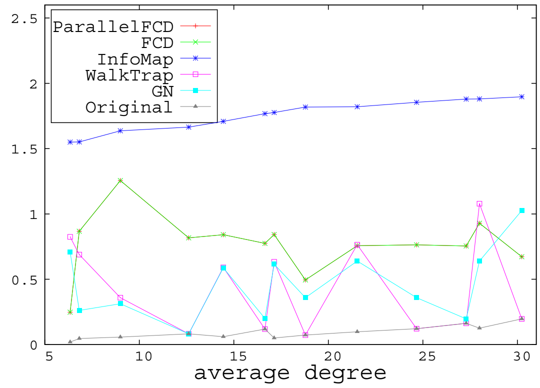
Figure 4.10: Measurements on real world graphs



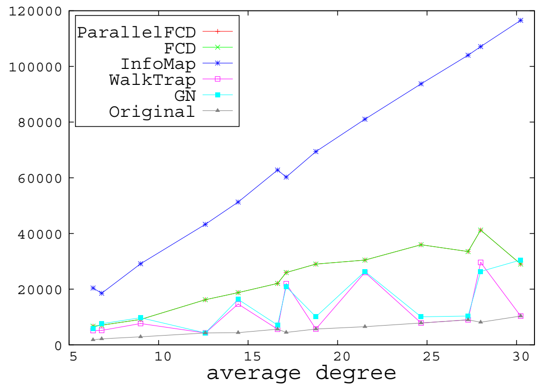
(a) Modularity



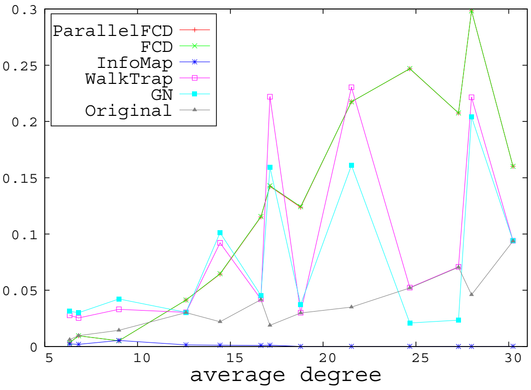
(b) Conductance



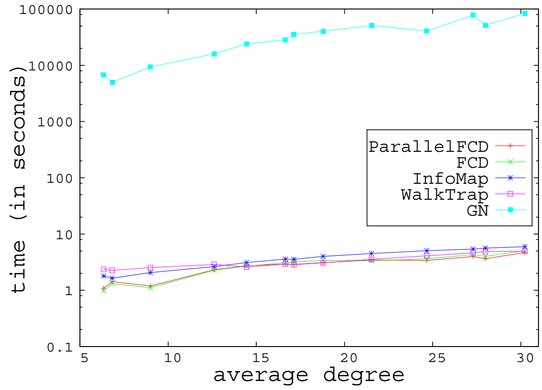
(c) Internal Density



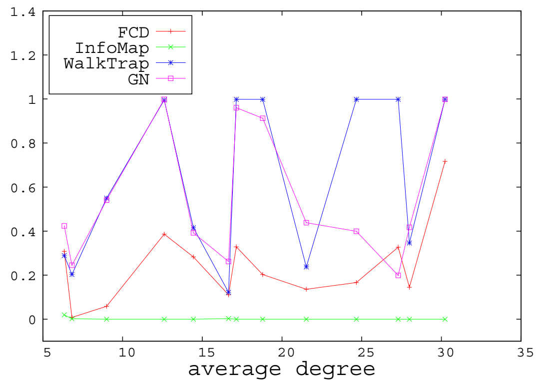
(d) Cut Ratio



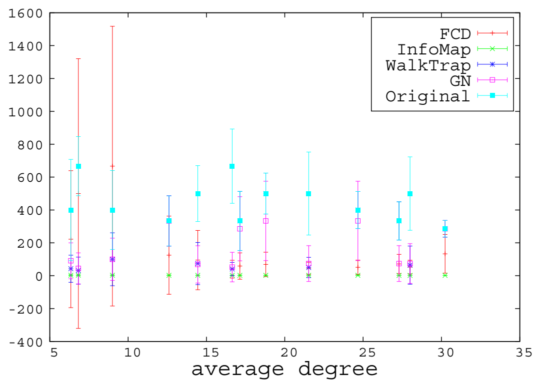
(e) Weighted Community Clustering



(f) Running Time

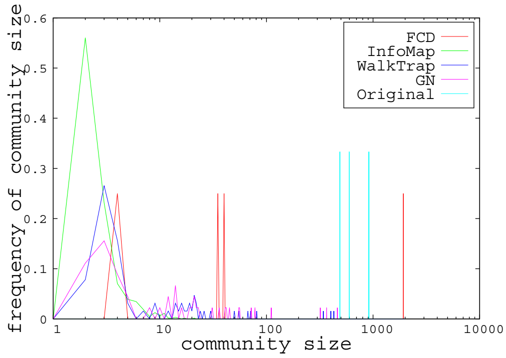


(g) NMI

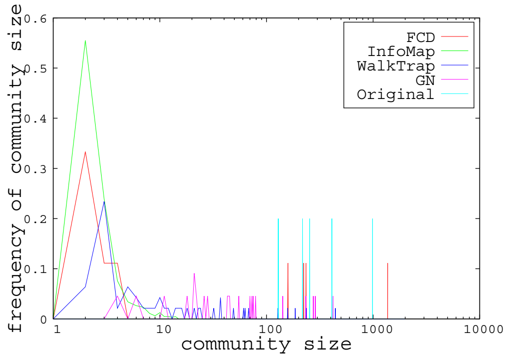


(h) Average Community Size

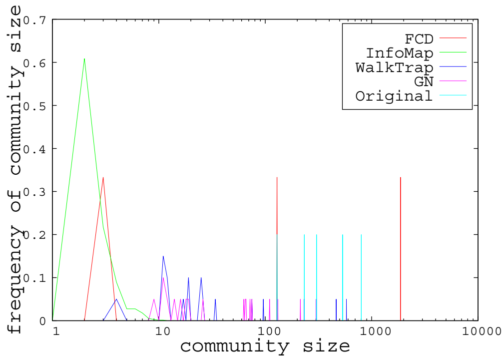
Figure 4.11: Measurements on synthetic graphs



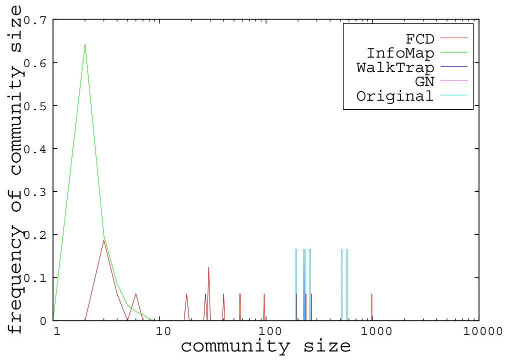
(a) Graph 1



(b) Graph 2

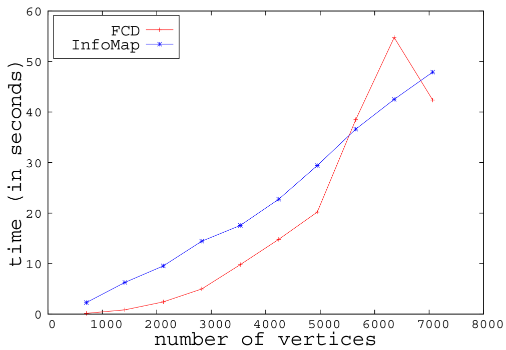


(c) Graph 3

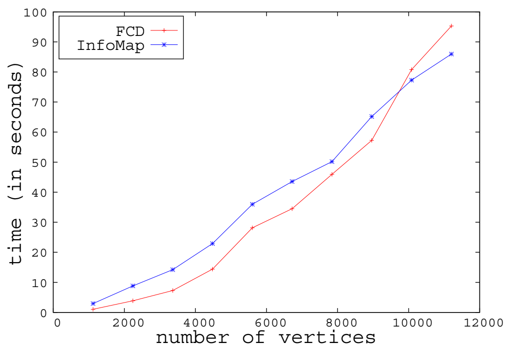


(d) Graph 4

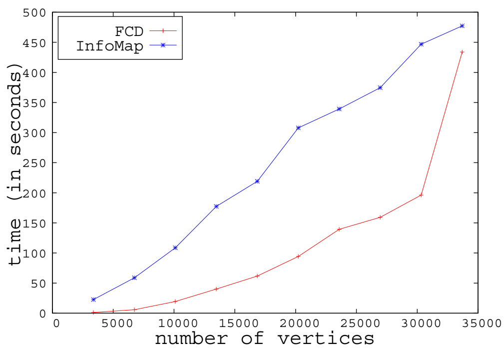
Figure 4.12: Community distribution



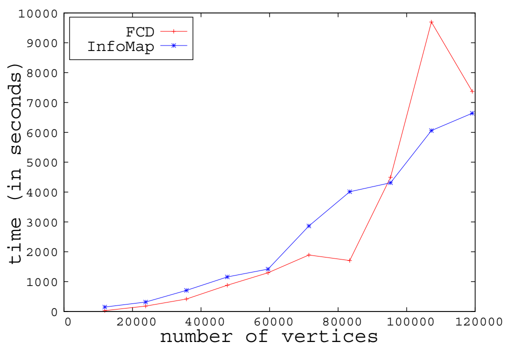
(a) Wiki Vote Network



(b) Arxiv HEP-PH network



(c) Email-Enron Network



(d) Epinion Network

Figure 4.13: Running time for large graphs

represents one metric value for the communities detected by one algorithm. Figure 4.11 (a) shows the modularity results. It shows that *WalkTrap* has the highest modularity in general, although in some cases, *GN* and *FCD* have the highest modularity, and *FCD* has a higher modularity than *InfoMap*. Figure 4.11 (b) shows the conductance results. It shows that *InfoMap* has the highest conductance and *GN* has the lowest. Figure 4.11 (c) shows the internal density results. It shows that *InfoMap* has the highest internal density, and *GN* has the lowest density. Figure 4.11 (d) shows the cut ratio results. It shows that *InfoMap* has the highest cut ratio, and *GN* has the lowest. Figure 4.11 (e) shows the *WCC* results. It shows that *FCD* and *WalkTrap* have a higher *WCC*, and *InfoMap* and *GN* have a lower *WCC*. As *FCD* and *ParallelFCD* detect the same communities, the green line and the red line overlap in Figure 4.11 (a)-(e). Figure 4.11 (f) shows the running time. *FCD* and *ParallelFCD* are shown to be faster in most cases. *GN* is much slower than *InfoMap*, *WalkTrap* and *FCD*. *ParallelFCD* is not obviously faster than *FCD*, due to the data communication between the host *CPU* and device *GPU*. Figure 4.11 (g) shows the measurement of *NMI*. It shows that both *InfoMap* and *WalkTrap* display higher *NMI* values. Figure 4.11 (h) shows the average and deviation of community size. The results reveal that the average size of communities is the closest to the ground truth when the average degree of the graph is about 10 or less than 10. In other words, *FCD* shows better performance in sparse graphs.

Comparing the metric values of the communities found by algorithms and the ground truth, we can see that in some cases *FCD* finds communities closer to the ground truth while in the other cases *GN* and *WalkTrap* find communities closer to the ground truth.

Figure 4.12 shows the distributions of sizes of communities in four randomly picked graphs. The x-axis is the size of the community. The y-axis is the frequency of community size. The results show that *FCD* and *WalkTrap* find communities of closer sizes to the ground truth relatively in general while in the last case, *GN* finds the communities of the most similar sizes as the known ones.

To sum up the results on these synthetic graphs, *FCD* (*ParallelFCD*) is more stable than *InfoMap* and *GN* in terms of effectiveness. *InfoMap* is the best in terms of internal density but the other three algorithms are better in terms of conductance, cut ratio and *WCC*. *GN* and *WalkTrap* are the best in terms of conductance and cut ratio but the other two algorithms are better in terms of internal density. Comparing the detected communities with the ground truth gives a different evaluation of detected community quality, as the good metric value does not always indicate the closeness of the detected communities to the ground truth. The running time shows that *FCD* is faster than the other three in general.

Another set of experiments demonstrating the running time are carried out on Wiki-Vote, Arxiv HEP-PH, Email-Enron, and Epinion network. We sample subgraphs from the networks. Every subgraph contains k percentage vertices of the original networks, where $k = 10, 20, ..., 90$. We run the *FCD* and *InfoMap* algorithms on these subgraphs and the original graphs. The running time is recorded. Figure 4.13 shows the running time changing, as the number of vertices of networks increases. Each figure shows the results for one data set. The x-axis is the number of vertices. The y-axis is the time measured in seconds. Due to *WalkTrap* and *GN* algorithms' scalability on large graphs, we only compare the *InfoMap* and *FCD* algorithms here. The results show that both algorithms are able to work with graphs with more than 100,000 vertices. For graphs such as Email-Enron with 33,696 vertices, the algorithms are able to finish the task in a few minutes. In most cases *FCD* is faster than *InfoMap*.

4.2.3.4 Experimental Assessment for Overlapping Community Detection

We set the parameter of θ to be 0 in this set of experiments, as we do not expect a large amount of overlaps in our synthetic graphs. We also examined higher values of θ on some graphs randomly chosen from the data sets and comparisons indicate a lower quality of detected communities with higher values of θ.

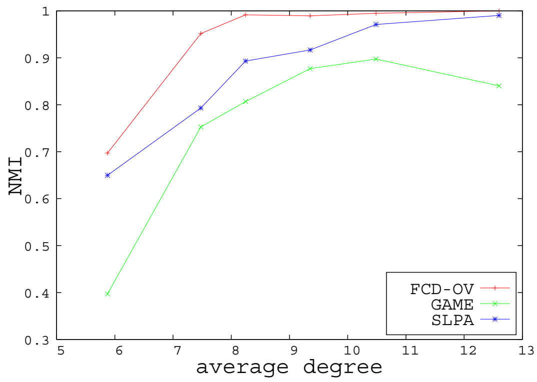
Figure 4.14 shows the results for the graphs with varying average degree. The x-axis is the average degree of the graphs. The y-axis is the value of metric. We conduct experiment on five graphs with similar average degree and then take the average of the values to reduce bias against different graph structures. Thus, each dot represents one metric value averaged over five values of the communities detected by one algorithm. Figure 4.14 (a) shows the results for *NMI*. It shows that our algorithm *FCD-OV* results in the highest *NMI* value compared to the *game-theory* and *SLPA* algorithm, which indicates that the communities found by *FCD-OV* are the closest to the true community structure in the input graphs. Figure 4.14 (b)-(f) show the measurement results for community quality. As the community structure of the generated graphs are known, we compare the quality of the communities detected by the three algorithms and the quality of the known communities that is labelled as original in the figures. It is obvious that *FCD-OV* results in the values that are closest to the original ones, suggesting that *FCD-OV* has a better capability to find true communities. Figure 4.14 (g) shows the average size of the set of communities found as well as the original average size of the communities in each graph. *FCD-OV* finds the communities with the average sizes that are closest to the known ones.

Figure 4.15 shows the same measurements as Figure 4.14 on the graphs with varying size. The x-axis is the number of vertices in the graphs. The y-axis is the value of metric. As the graph size increases from 10,000 to 50,000, we measure the values of each metric for communities found in each graph. We conduct experiment on every five graphs with the same size and then take the average of the values to reduce bias against different graph structures. Figure 4.15 (a) shows the results of *NMI*. It shows that *FCD-OV* has the highest values. Figure 4.15 (b)-(f) show the measurement results on community quality. Figure 4.15 (g) shows the average size of the set of communities found as well as the original average size of the communities in each graph.

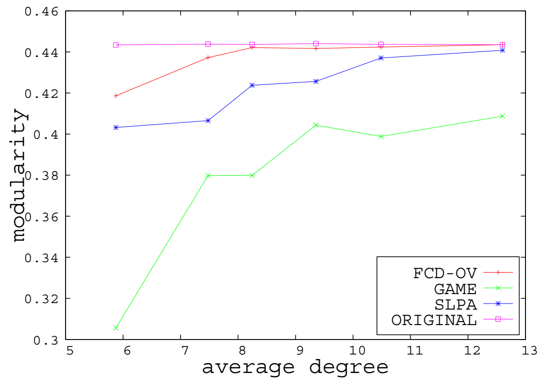
The results in Figure 4.15 suggest that the communities found by *FCD-OV* are the most truthful to the known communities. They also suggest that the change of metric values for the communities is almost independent of the size of graphs except the cut ratio.

Figure 4.16 shows the average size and standard deviation of each set of the communities in graphs of different average degrees. Figure 4.17 shows the average size and standard deviation of each set of communities in graphs with different sizes. In Figure 4.16, the x-axis is the average degree of the graphs, and the y-axis is the size of the community. In Figure 4.17, the x-axis is the size of the graph, and the y-axis is the size of the community. It shows that the average size of the communities detected by *FCD-OV* is closer to the average size of known communities.

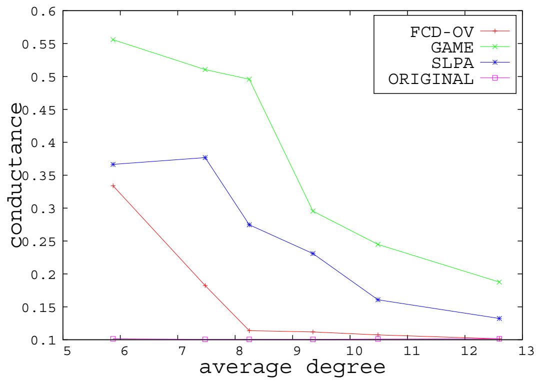
Figure 4.18 shows the plots of community distribution for six randomly selected graphs. The x-axis is the size of the community. The y-axis is the ratio of the number of communities of certain size and the total number of communities in the graph, i.e., the frequency of community size. The known communities are mostly within size 100 to 200, while many communities detected by the *game-theory* and *SLPA* algorithms are of a size smaller than 100. In comparison, many lines for



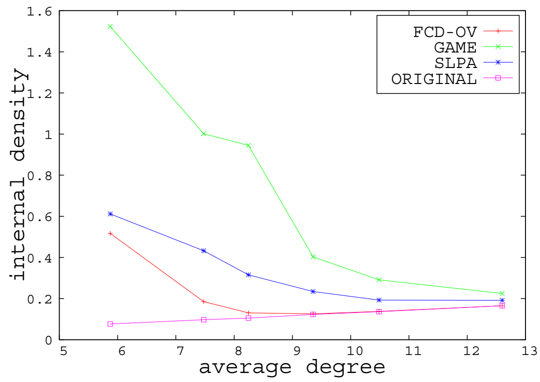
(a) NMI



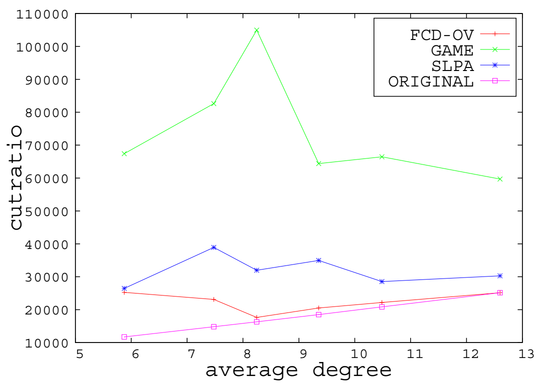
(b) Modularity Q_{ov}^{E}



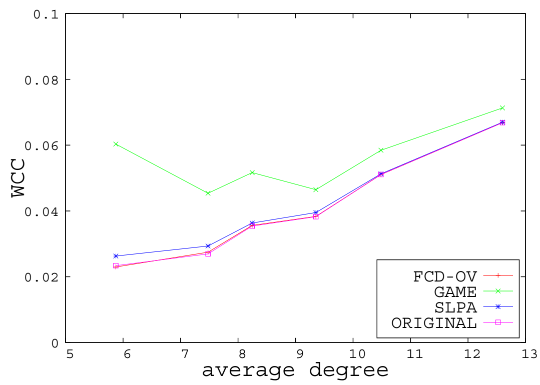
(c) Conductance



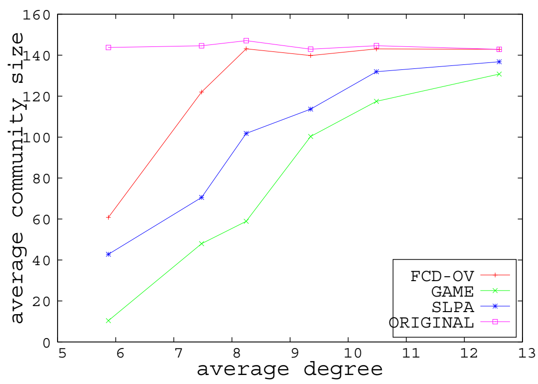
(d) Internal Density



(e) Cut Ratio

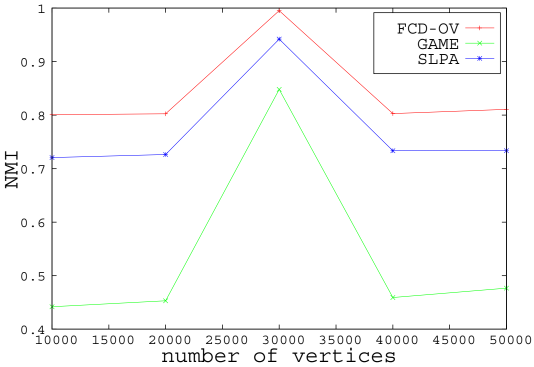


(f) Weighted Community Clustering

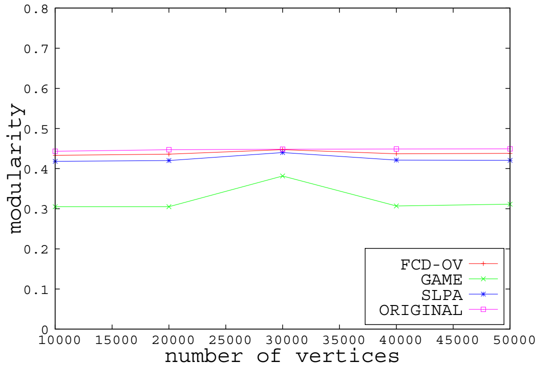


(g) Average Community Size

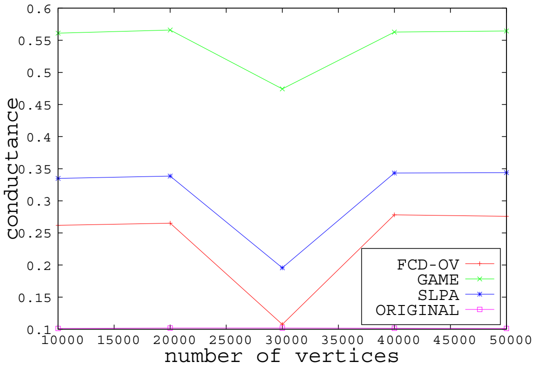
Figure 4.14: Measurements on graphs with varying average degree



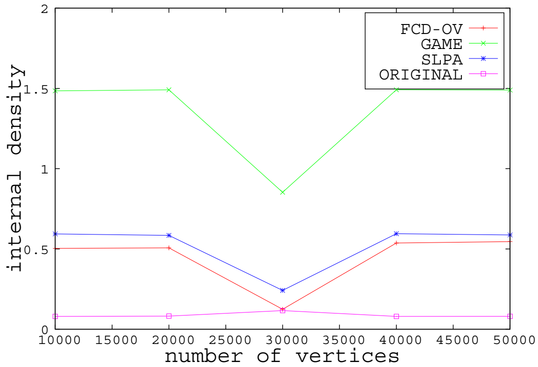
(a) NMI



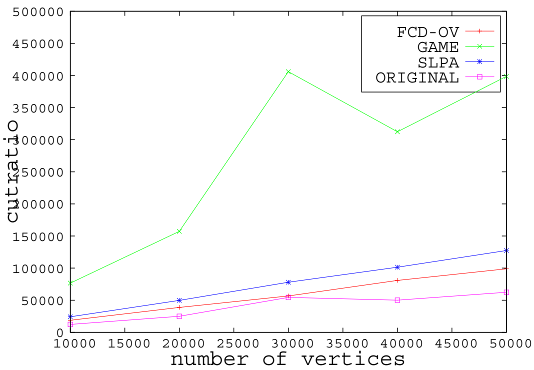
(b) Modularity Q_{ov}^E



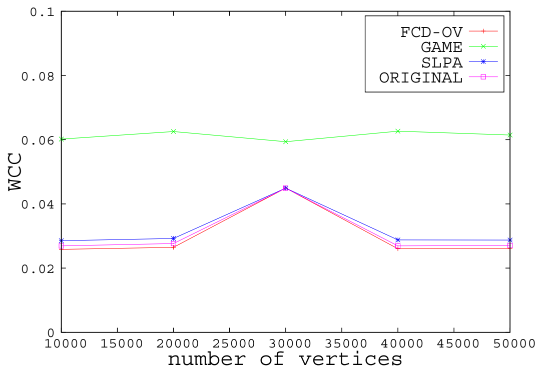
(c) Conductance



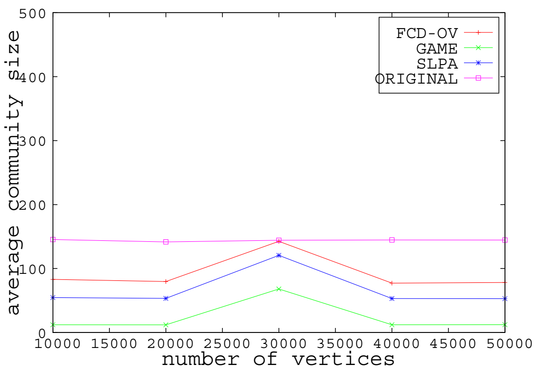
(d) Internal Density



(e) Cut Ratio

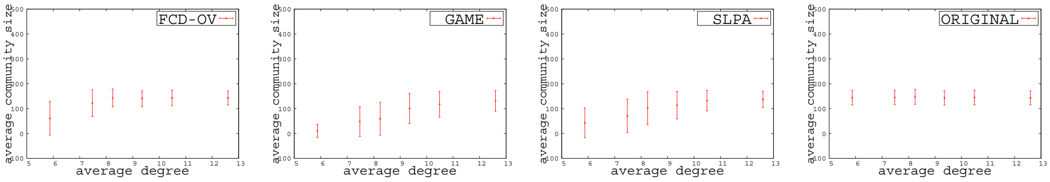


(f) Weighted Community Clustering



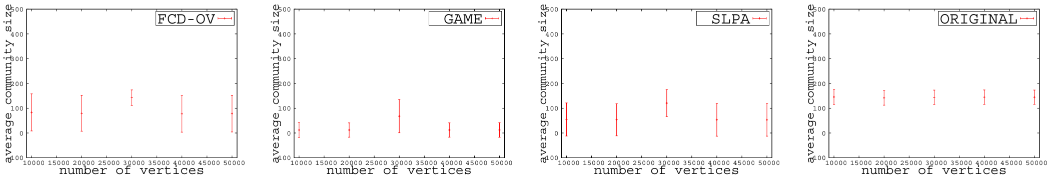
(g) Average Community Size

Figure 4.15: Measurements on graphs with varying size



(a) FCD-OV (b) GAME (c) SLPA (d) ORIGINAL

Figure 4.16: Measurements on graphs with varying average degree



(a) FCD-OV (b) GAME (c) SLPA (d) ORIGINAL

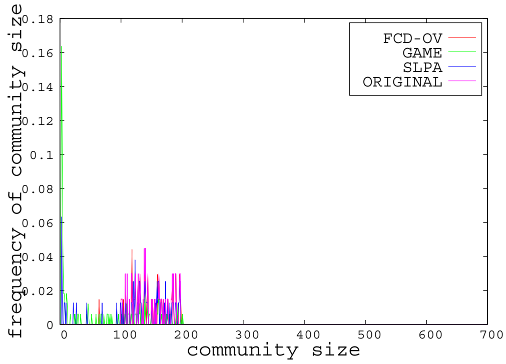
Figure 4.17: Measurements on graphs with varying size

FCD-OV overlap with the lines for the known communities, and this indicates that most of the communities detected by *FCD-OV* are of the sizes of the known communities or close to the sizes of the known communities.

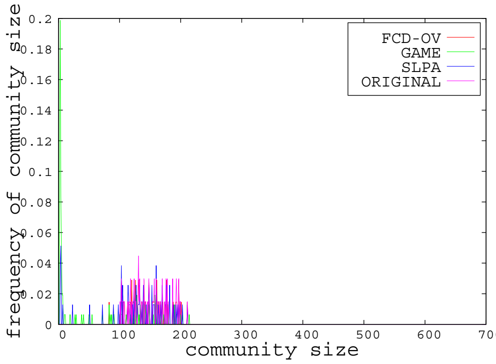
Figure 4.19 shows the running time of three algorithms on the two sets of generated graphs. In both cases, *FCD-OV* costs the least time compared to the *game-theory* and *SLPA* algorithms. The running time also shows that *FCD-OV* detects community in graphs with 50,000 vertices within one and a half minutes. The high efficiency of *FCD-OV* is exhibited.

To sum up, we empirically evaluate *FCD* algorithms. For disjoint community detection, we examine *FCD* on four real graphs and a set of synthetic graphs. Knowing few ground truths about the communities in the real graphs, we measure the community quality by calculating the values of chosen metrics. For synthetic graphs, we measure the extent to which the detected communities match the ground truths. Compared to the *InfoMap*, *WalkTrap* and *GN* algorithms, *FCD* is the fastest while it produces results of comparable quality. *FCD* shows better performance on several metrics.

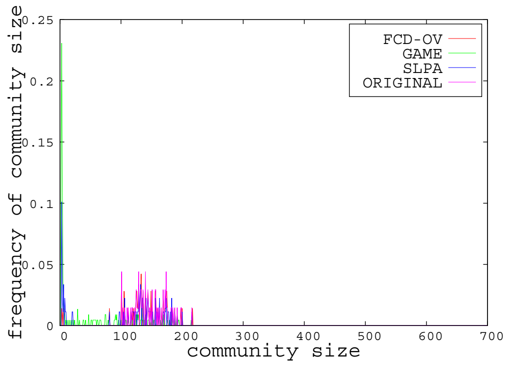
For overlapping community detection, we examine *FCD* on synthetic graphs.



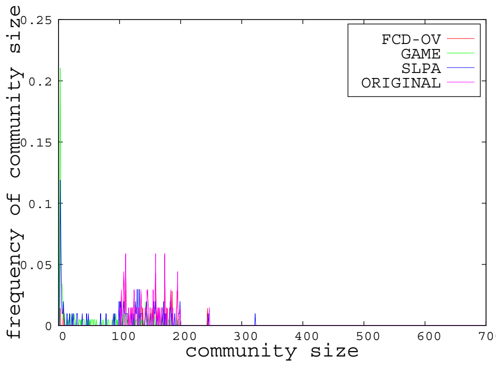
(a) Graph 1



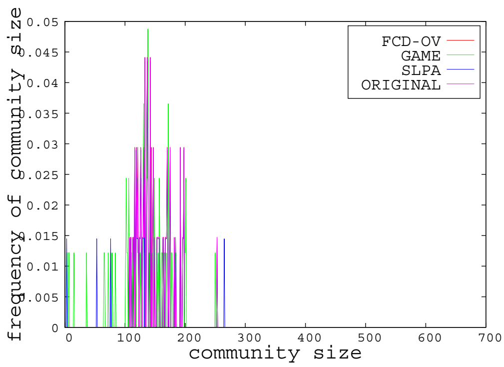
(b) Graph 2



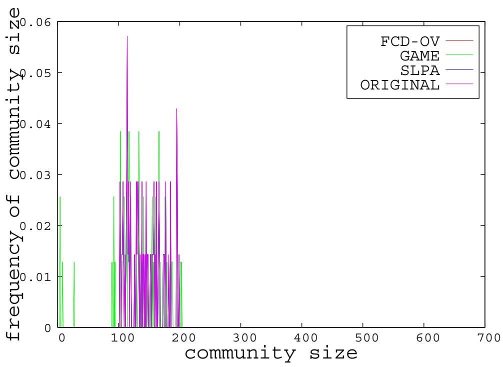
(c) Graph 3



(d) Graph 4

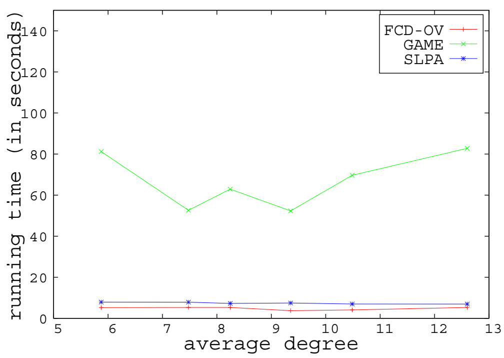


(e) Graph 5

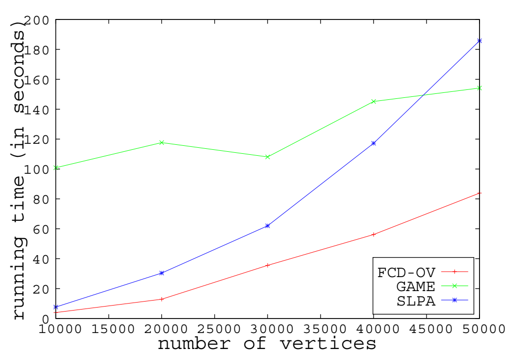


(f) Graph 6

Figure 4.18: Community distribution



(a) Running Time



(b) Running Time

Figure 4.19: Running time comparison

We measure the community quality by calculating values of the metrics and compare the detected communities with the ground truths. Compared to the *game-theory* and *SLPA*, *FCD* identifies communities closer to the ground truths. *FCD* also takes less time to find the communities.

4.2.4 Summary

In this section, we propose two fast community detection algorithms, one for disjoint community detection and the other for overlapping community detection. They initiate each vertex to independently seek out the community in its neighbourhood. Each vertex chooses its community and peers, based on a knowledge of degrees and clustering coefficients of neighbours and the number of common neighbours. The algorithms are parallelizable and thus we devise a GPU version of the algorithm for disjoint community detection for parallel computation. In the case of disjoint community detection, we empirically evaluate the performance of *FCD*, and compare it to the *InfoMap*, *WalkTrap* and *GN* algorithms. We find that *FCD* is the fastest, while it produces results of comparable quality. We assess effectiveness based on the values of modularity, conductance, internal density, cut ratio, weighted community clustering, and normalized mutual information as well as community size. In the case of overlapping community detection, we empirically compare the performance of *FCD* for overlapping communities with the *game-theory* and *SLPA*. We find that *FCD* for overlapping communities is more efficient, and more effective.

4.3 Local Closeness Community Detection

4.3.1 Overview

We propose an algorithm for the detection of structural communities in simple graphs. The algorithm is able to detect overlapping communities. The algorithm is based on a local notion of closeness centrality [18, 63]. Closeness centrality measures how close a vertex is to all the other vertices, and indicates the importance [119] of vertices in a graph. We utilize such measurement of importance for locating densely connected members of communities. We try to propose an algorithm for direct community detection using the metric of closeness centrality. However, we observe a problem in efficiency. The computation of closeness centrality is costly. Therefore, we alleviate the problem by defining a local notion of closeness centrality.

We leverage the fundamental concepts used in the *local outlier factor* algorithm by Breunig et al. [24]. The local outlier factor algorithm finds outliers for clusters with nonuniform density. We adapt the idea of a local density to the definition of a local notion of closeness centrality. This is done by computing distance, reachability distance and density of a vertex within its restricted neighbors, as in the local outlier factor algorithm local reachability density and local outlier factor are computed with the nearest neighbors. To account for the graph structure, we also define the adjusted geodesic distance, which is the geodesic distance between a vertex and its surrounding neighbor with adjustment regarding the number of shortest paths. Based on this definition of distance, a local notion of closeness centrality and a local closeness factor are defined. Vertices are paired with their neighbors based on their respective local closeness factor to form the communities. We find that the local notion of closeness centrality yields a more effective and efficient algorithm for community detection than closeness centrality does.

We compare the performance of our algorithm with that of three state-of-the-art community detection algorithms for overlapping communities: a label propagation algorithm [161], a game theory algorithm [32] and a probabilistic model-based algorithm [166]. We empirically evaluate the performance of our algorithm with varying parameters. We calculate effectiveness by calculating the *normalized mutual information* [91] and *omega index* [39] between the set of communities found and the known set of communities.

We show that our algorithm displays competitive performance on both generated graphs and real world graphs. It is more effective and efficient than the algorithms compared for sparse graphs on a large scale.

Our contribution is an algorithm and its evaluation for the detection of potentially overlapping communities. The algorithm localizes the closeness centrality with the notions of local reachability density that we have adapted to the case of graphs and geodesic distance from the local outlier factor algorithm. The closeness factor we introduce is local, in the sense that only a restricted neighborhood of each vertex is taken into account.

4.3.2 Motivation

In this section, we introduce the concepts of closeness centrality and local outlier factor, and how these concepts constitute the idea of our community detection method.

4.3.2.1 Closeness Centrality

In graph theory, a natural distance metric that is defined by the length of shortest path exists between any pair of vertices. It is called geodesic distance. The farness of a vertex is defined as the sum of its geodesic distances to all the other vertices, and its closeness is defined as the inverse of the farness [133]. Thus, the more central a vertex is, the smaller its total geodesic distance to all other vertices. The classic definition of Closeness Centrality (CC) [18] is shown below:

$$CC\left(v\right)=\sum_{x\neq v,x\in V}\frac{1}{d\left(x,v\right)} \tag{4.1}$$

V is the set of vertices of a graph. The closeness centrality of a given vertex v is the sum of the inverse of distance of between v and all the other vertices in the graph. The geodesic distance between vertex x and vertex v is denoted by $d(x,v)$. Closeness centrality indicates the importance of a vertex [119] in terms of its closeness to the rest vertices.

The naive algorithm that we design for community detection integrates this indicator. The experiment demonstrates the effectiveness of closeness centrality. It shows that this closeness centrality based community detection algorithm rivals the three algorithms that it is compared with. However, it suffers from an efficiency issue. The complexity for computing closeness centrality is $O(n^3)$. n is the number of vertices in the graph. This is because the computation of closeness centrality involves the computation of geodesic distances between all pairs of vertices. This complexity is considered to be a computational high for large graphs.

4.3.2.2 Local Outlier Factor

We desire a notion of local closeness centrality to alleviate the above problem. Therefore we design a such notion by adapting the idea of local density from LOF.

LOF is originally proposed to find outliers for clusters with non-uniform density, where density is estimated by Euclidean distance among data points. Regions of similar density are identified by comparing the local density of each data point to those of its neighbours. Data points are considered to be outliers if they have a

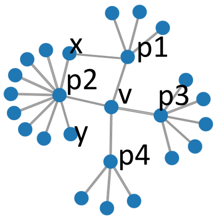


Figure 4.20: Vertex V has 4 one-step neighbors and 20 two-step neighbors.

lower density than their neighbours. LOF measures the extent to which a point is an outlier based on the relative density of its local neighbourhood. This enables LOF to spot outliers that cannot be observed from a global view.

The complexity of LOF computation for each object mainly depends on the parameter k, which specifies the number of nearest neighbor involved in the calculation of LOF value for each object. The complexity is reduced by restricting the calculation to *k-distance* neighbors of each object. The *k-distance* of an object p is denoted as *k-distance* (p). It is defined as the distance $d(p, o)$ between p and an object $o \in D$ such that:

1. for at least k objects $o' \in D \setminus \{p\}$ it holds that $d(p, o') \leq d(p, o)$, and

2. for at most $k-1$ objects $o' \in D \setminus \{p\}$ it holds that $d(p, o') < d(p, o)$

The *k-distance* neighborhood of p contains every object whose distance from p is not greater than *k-distance*, i.e. $N_{k\text{-}distance(p)}(p) = \{q \in D \setminus \{p\} | \; d(p, q) \leq k\text{-}distance(p)\}$. The objects $\{q\}$ are called the k-nearest neighbors of p.

The definition of a *k-distance* neighborhood defined in Euclidean space, however, is not directly suitable for measuring local density in a graph. Distance in Euclidean space is usually a continuous value. But the geodesic distance between two vertices in graph is usually an integer value. So it is more likely to find that many pairs of vertices have exactly the same distance in graphs, compared to Euclidean space. For example, vertex v in figure 4.20 has 4 one-step neighbors and

20 two-step neighbors. When $k = 4$, *k-distance* neighborhood of v is p1, p2, p3 and p4. When $k = 24$, *k-distance* neighborhood of v is the whole set of vertices in the graph except v. When $k = 15$, we face the problem of choosing part of the two-step neighbors as the *k-distance* neighborhood of v. This leads to a situation in which many vertices have the same measured distances, while they are indeed structurally distinguishable in that pairs of vertices may have a different number of shortest paths, e.g. vertex x and y. We consider that a pair of vertices with a greater number of shortest paths have better reachability, and are intuitively closer.

The choice of neighborhood would be important yet hard because it directly affects the measurement of importance of vertex v. Hence, we define our restriction of neighborhood by geodesic distance to the vertex, rather than *k-distance*, and we define a distance that integrates a local graph structure. Such considerations increase the disparity of closeness, and facilitates the comparison of closeness and community discovery. Then, we define the local notion of closeness centrality called local reachability density, and local closeness factor for detecting communities.

4.3.3 Local Closeness Factor

In this section we describe the definition of local closeness factor and present the upper and lower bound of LCF. We also analytically prove that LCF value is an effective indicator of the importance of a vertex.

4.3.3.1 Definitions

We extend the concept of local density to graphs for finding communities. We define the local closeness factor beginning with the notion of *within-k-step neighbors*.

The *within-k-step* neighbors of vertex v are the vertices that have a geodesic distance that is smaller, or equal to, k from vertex v. For vertices with the same geodesic distance to vertex v, we consider the vertices that have a greater number of shortest paths with vertex v nearer to v. For example, for vertices that are two steps to v, we consider vertices that have more common neighbors with vertex v as being the nearer to v. Therefore, we define a distance between vertex v_i and v_j as follows:

Definition 4.

$$distance(v_i, v_j) = g(v_i, v_j) - \frac{\sigma(v_i, v_j)}{\Sigma_{v_t \in N_{v_i}} \sigma(v_i, v_t)} \tag{4.2}$$

where $g(v_i, v_j)$ *is the geodesic distance of vertex* v_i *and* v_j. $\sigma(v_i, v_j)$ *is the number of shortest paths between* v_i *and* v_j. N_{v_i} *is the set of* v_i*'s within-k-step neighbors.*

To calculate distances for *within*-1-*step* neighbors or *within*-2-*step* neighbors, the formula can be simplified and approximated by replacing $\Sigma_{v_t \in N_{v_i}} \sigma(v_i, v_t)$ as d_i, the degree of v_i. The subtrahend in Definition 2 should be in the range of 0 to 1 so that the following conditions are satisfied:

1. $distance(v_i, v_j) > 0$, and
2. $g(v_i, v_{j1}) \leq g(v_i, v_{j2})$ if and only if $distance(v_i, v_{j1}) \leq distance(v_i, v_{j2})$.

For *within*-1-*step* neighbors, $\sigma(v_i, v_j)$ is the same for any j and so is $distance(v_i, v_j)$. For *within*-2-*step* neighbors, $\sigma(v_i, v_j)$ measures the number of common neighbors between vertex i and j. This definition is integrated with graph structure, and thus facilitates comparison of local density.

Next, we define *reachability distance*. Reachability in graph is usually defined as the number of steps a vertex v_i needs to take to reach another vertex v_j, which is measured by shortest path length (geodesic distance). Here, we define reachability distance as follows:

Definition 5. ***reachability distance*** *of a vertex* v_i, $rd_k(v_i)$, *is the maximum of* *the distance between vertex v_i and v_i's within-k-step neighbors.*

The objective of reachability distance is to reduce the fluctuations of the distances between vertex v_i's nearest vertices and vertex v_i, so that vertices within the same neighborhood will have a similar reachability distance.

Local reachability density is then defined as:

Definition 6. ***Local reachability density*** *of a vertex v_i is the inverse of the average reachability distance of the within-k-step neighbors.*

$$lrd_k(v_i) = \frac{|N_{v_i}|}{\sum_{v_j \in N_{v_i}} rd_k(v_j)} \tag{4.3}$$

where N_{v_i} is the *within-k-step* neighbors of v_i, $rd_k(v_j)$ is reachability distance of vertex v_j, and $|N_{v_i}|$ is the number of the *within-k-step* neighbors of v_i, which varies for different v_i.

Local reachability density is defined based on the same idea as closeness centrality. Local reachability density restricts the local neighborhood to approximate closeness centrality.

Definition 7. ***Local Closeness Factor*** *of a vertex v_i is the average of the ratio of the local reachability density of vertex v_i and those of v_i's within-k-step neighbors.*

$$LCF(v_i) = \frac{\sum_{v_j \in N_{v_i}} \frac{lrd_k(v_j)}{lrd_k(v_i)}}{|N_{v_i}|} \tag{4.4}$$

The *local closeness factor* is low if local reachability density of vertex v_i's neighbors are low and the local reachability density of vertex v_i is high. The *local closeness factor* is high if its neighbors' *local reachability density* are high and it has a lower *local reachability density.*

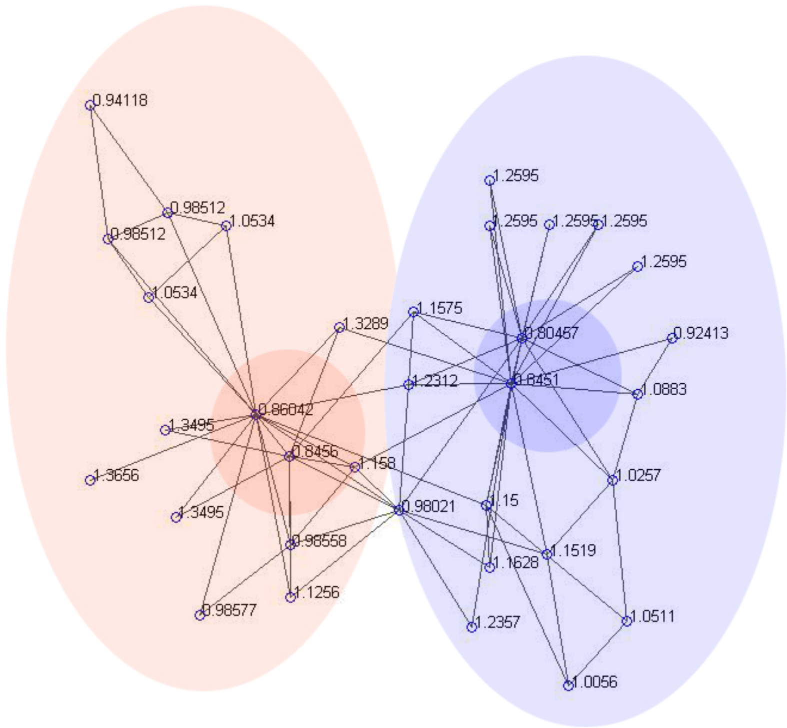


Figure 4.21: Zachary's Karate Club example

Figure 4.21 shows a simple example, a real world graph from Zachary's Karate Club data. It has 34 vertices and 78 edges. Each vertex represents a member of the club, and each edge represents the friendship between the members. The value associated with each vertex is the *local closeness factor* value calculated and plotted next to the corresponding vertex. The members fell into two groups due to a controversy between the administrator and the instructor. The two different colors represent the two communities detected. Vertices in the centre of the communities are, core members of the communities, and are plotted inside the darker shadow. Vertices on the border of the communities are plotted in the lighter shadow. These vertices have relatively higher *local closeness factor* values.

4.3.3.2 Properties of Local Closeness Factor

In this section, we conduct a detailed analysis on the properties of LCF. We aim to show that our definition of LCF captures the spirit of local closeness for community detection. Specifically, we a give lower and upper bound for LCF, and show what LCF value a vertex has in different positions of a community.

4.3.3.3 Bound for LCF

Let V be the set of vertices of a graph G. Let *reach-dist-min* denote the minimum reachability distance of the objects in V, i.e. $\textit{reach-dist-min} = \{rd_k(u) | u \in V\}$. Similarly, let *reach-dist-max* denote the maximum reachability distance of the objects in V. Let ε be defined as:

$$\varepsilon = \frac{\textit{reach-dist-max}}{\textit{reach-dist-min} - 1} \tag{4.5}$$

Then for all vertices $u \in V$, such that for:

1. all the *within-k-step* neighbours v of u are in V, and
2. all the *within-k-step* neighbours w of v are in V,

it holds that $1/(1 + \varepsilon) \leq LCF(u) \leq (1 + \varepsilon)$.

Proof: For all *within-k-step* neighbors v of u, $rd_k(u) \geq \textit{reach-dist-min}$. Then the local reachability density of u, as per definition 3, is $\leq 1/\textit{reach-dist-min}$. On the other hand, $rd_k(u) \leq \textit{reach-dist-max}$. Thus the local reachability density of u is $\geq 1/\textit{reach-dist-max}$.

Let v be a *within-k-step-neighbour* of u. By an argument identical to the one for u above, the local reachability density of v is also between $1/\textit{reach-dist-max}$ and

$1/$*reach-dist-min*.

Thus, by definition 4, we have *reach-dist-min*/*reach-distmax* $\leq LCF(u) \leq$ *reach-dist-max*/*reach-dist-min*. Hence, we establish $1/(1+\varepsilon) \leq LCF(u) \leq (1+\varepsilon)$.

The interpretation of bound is as follows. Let us consider the vertex u that is in the centre of community C. All the *within-k-step* neighbours v of u are in C, and all the *within-k-step* neighbours of v are also in C. For such a central vertex u, the LCF value of u is bounded. If vertices in C are evenly connected to each other, the ε value in the bound can be quite small, thus forcing the LCF of u to be close to 1.

4.3.3.4 LCF for community detection

We originally introduce the local closeness factor to measure how close a vertex is to its local neighbourhood. This measurement directly determines the local closeness of a vertex in a community. The more important a vertex is to a community, the smaller LCF value it has.

To prove this property of LCF, we first give a definition of relative importance to a vertex from the perspective of community detection.

Definition 8. *For any vertex v_c and v_p in the same community, we say v_c is more important than v_p, if v_c has smaller sum of distance to its within-k-step neighbourhood than v_p does.*

Practically, v_c is usually the core member of a community while v_p is the peripheral member. A core member always has smaller LCF value than a peripheral member does. This is the fundamental that ensures our community detection method works correctly and effectively.

Proof: Suppose v_c and v_p are vertices in the same community, v_c is important than v_p. According to definition 8, $lrd(v_c) > lrd(v_p)$, Thus by definition 7, we have

$LCF(v_c) < LCF(v_p)$. ■

4.3.4 Algorithm

We first propose a closeness centrality based algorithm (Algorithm 10). To calculate the closeness centralities, we calculate the distance matrix (line 1), using the Floyd Warshall algorithm [60]. After obtaining a closeness centrality value for each vertex, the algorithm binds each vertex with its immediate neighbour (neighbours) if the neighbour(s) has(have) a closeness centrality value lower than the vertex itself and the rest of its(their) immediate neighbours (Algorithm 11). Otherwise, the vertex is bound with itself. $v.Cset$ contains the bound vertices with which vertex v will possibly be grouped together to form communities.

In the next step, the function checks through each vertex's bound vertices. For vertex v_i, check its bound vertices ($v_i.Cset$). If any of the bound vertices, i.e. v_j, has been bound with other vertices which are only bound with themselves, we bind vertex v_i with those vertices by updating $v_i.Cset$ correspondingly (Algorithm 11.line 6-12). By doing this, the algorithm potentially binds all the vertices that belongs to the same community together, because eventually the vertices that belong to the same community will be bound to the same vertex. Therefore, communities are detected.

Algorithm 4.5: Closeness Centrality Based Community Detection (CC)

Input: graph $G(V, E)$;
Result: Clusters C_i, $i \in (1, 2, ..., |C|)$

1 Compute distance matrix;
2 Compute $cc[v]$, $v \in V$;
3 $\widehat{C}_i = Binding(G, cc, iter)$, $i \in (1, 2, ..., |\widehat{C}|)$;
4 $C_i =$ Refinement$(\widehat{C})$, $i \in (1, 2, ..., |C|)$;
5 **Return** C_i,$i \in (1, 2, ..., |C|)$;

The first two loops of the binding function aim to find the core members of the communities (Algorithm 11.line 1-12). In this procedure, some vertices may be misclassified, especially the vertices that are on the borders of the communities. Therefore the last loop of the binding process conducts a vertex level adjustment. It adjusts the communities to which a vertex belongs if necessary. The function checks through each vertex and changes its membership if the number of connections the vertex has with current communities to which it belongs is less than those it has with the other communities (Algorithm 11.line 13-17). As some vertices leave their current communities and join new communities, other vertices' connections with the communities may be affected. For this reason, this process can continue for several iterations. To prevent deadlocks wherein the vertices repeatedly change memberships among certain communities, the number of iterations is pre-determined and input as a parameter.

The last phase of the CC algorithm is a minor refinement, a community level adjustment (Algorithm 10.line 4). Trivial communities, communities that have more connections with other communities than within themselves or isolated vertices, may be detected. The algorithm goes through each community and checks whether each community has more connections inside the community than with other communities. If not, communities are merged to eliminate the trivial communities (Algorithm 12).

To improve effectiveness and efficiency, we propose the LCF algorithm, an algorithm built upon the local closeness factor values (Algorithm 13). According to the value computed for each vertex, the LCF value, we can tell how closely a vertex is connected to its neighbours. Comparing the *local closeness factor* value of each vertex with those of its neighbours, a vertex is considered to be in the same community as its neighbour(neighbours) whose *local closeness factor* value is the

Algorithm 4.6: The Binding scheme

Input: graph $G(V, E)$, ϕ, *iter*;
/* ϕ *can be lcf or cc values* */;
Result: Clusters $\widehat{C}_i$, $i \in (1, 2, ..., |\widehat{C}|)$

for *each* v **do**
  if $\phi[v_i]$ ¡ $\phi[v]$ **then** `/*` $v_i \in v_{Neighbour}$, ϕ`[`v_i`] >` ϕ`[v] for cc, and all min replaced by max */`
    $v.Cset$ add v_{min} ,where $\phi[v_{min}] = \min(\phi[v_i])$;
  else
    $v.Cset$ add v;
for *each* v **do**
  for *each* $v_j \in v.Cset$ **do**
    if $v_j \notin v_j.Cset$ **then**
      repeat
        $v_t \leftarrow$ each vertex $\in v_j$.Cset;
      until $v_t \in v_t.Cset$;
      update v_j in $v.Cset$;
repeat
  for *each* v **do**
    if *v has more connections to $\overline{C}_i$ than the connections within itself, $1 \leq i \leq |\overline{C}|$;*
    then cluster v into $\overline{C}_i$;
until *finish iterations*;
Return $\widehat{C}_i$,$i \in (1, 2, ..., |\widehat{C}|)$;

Algorithm 4.7: The Refinement scheme

Input: graph $G(V, E)$, $\widehat{C}_i$, $i \in (1, 2, ..., |\widehat{C}|)$;
Result: Clusters C_i, $i \in (1, 2, ..., |C|)$

for *i from 1* **to** $|\widehat{C}|$ **do**
  if *$\widehat{C}_i$ has more connections with $\widehat{C}_j$, $1 \leq j \leq |\widehat{C}|$ and $j \neq i$* **then** `/* check and merge trivial communities */`
    merge $\widehat{C}_i$ and $\widehat{C}_j$
Return C_i,$i \in (1, 2, ..., |C|)$;

minimum among all the neighbours. Vertices on the border of communities may be misclassified, and thus some refinements are made to improve the effectiveness.

The algorithm starts by finding *within-k-step* neighbors. As geodesic distance is different from Euclidean distance in that geodesic distance is presented as an integer, many vertices may have the same geodesic distance with the designated vertex while their connectedness towards the designated vertex is different. For instance, vertex v_j and v_k have the same distance with v_i, as both of them are two steps away from v_i. If v_j has more common neighbours with v_i than v_k does, v_j can be viewed to be closer to v_i than v_k. Pairwise distances are recorded for computation of *reachability distance* and *local closeness factor* value.

Algorithm 4.8: Local Closeness Factor Based Community Detection (LCF)

Input: graph $G(V, E)$, k, $iter$;
Result: Clusters C_i, $i \in (1, 2, ..., |C|)$

1 **for** *each* v **do**
2 | Find neighbours within k steps;
3 Compute $lcf[v]$, $v \in V$;
4 $\widehat{C}_i = Binding(G, lcf, iter)$, $i \in (1, 2, ..., |\widehat{C}|)$;
5 $C_i =$ Refinement($\widehat{C}$), $i \in (1, 2, ..., |C|)$;
6 **Return** C_i,$i \in (1, 2, ..., |C|)$;

After finding *within-k-step* neighbours and calculating the *reachability distance*, *local reachability density* and *local closeness factor* value for each vertex (Algorithm 13.line 1-3), the algorithm triggers the same binding (Algorithm 11) and refinement process (Algorithm 12) as the CC algorithm (Algorithm 13.line 4-5). The binding process (Algorithm 11) binds each vertex with its immediate neighbour (neighbours) if the neighbour(s) has(have) a lower *local closeness factor* value than the vertex itself and the rest of its immediate neighbours. Otherwise, the vertex is bound with itself.

4.3.4.1 Choice of Parameters

In this section, we show how to choose parameters and how parameters affect the effectiveness of our detection method.

- k

In our definition, k specifies the *within-k-step* neighbourhood involved in the calculation of LCF value. In practice, $k = 2$ serves better than $k = 1$, and $k = 2$ does not incur much more computational complexity than $k = 1$ does. $k = 2$ is also more efficient than $k = 3$. Considering both efficiency and effectiveness, we choose k to be 2 in our experiments.

- **Iteration t**

In the last step of the binding function, some vertices migrate, or in other words, change memberships, among communities during each iteration. As the iteration continues, the community memberships gradually become stable.

Figure 4.22 shows the example curve of relations between the number of iterations and the number of migrations. Red curve belongs to a data set of 10,000 vertices, while the blue curve belongs to that of 50,000 vertices. The x-axis stands for the number of iterations, while the y-axis stands for the number of vertex migration occurred in each iteration. The two curves are almost overlapped, which means that the relation between iteration and migration is independent of the size of the data set. For both curves, the number of migrations drop rapidly as iteration continues and starts oscillating after the point wherein *iteration* equals to 4. For $t \in [4, 10]$, we find that there is no big difference in the effectiveness.

Efficiency-wise, it takes 0.5 and 2 seconds respectively for a data set of 10,000 vertices and a data set with 50,000 vertices to finish a 10-iteration micro adjustment.

Based on the effectiveness and complexity of iteration, we choose $t = 4$ as an optimized parameter.

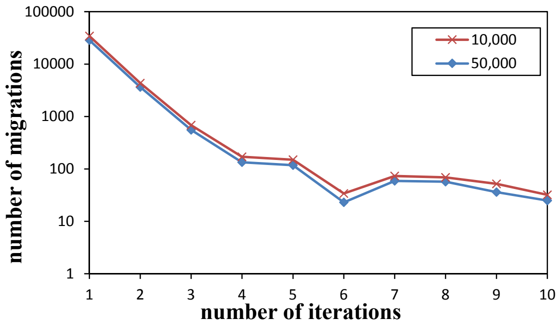


Figure 4.22: The number of vertex migrations during each iteration.

4.3.4.2 Complexity Analysis

The complexity for the CC algorithm is $\mathcal{O}(n^3)$ as the complexity of pair-wise geodesic distance calculation is $\mathcal{O}(n^3)$. n is the number of vertices of the graph. For the LCF algorithm, we set k to 1 for small graphs or 2 for large graphs, as explained in previous subsection, so the complexity for finding neighbours and calculating LCF values is $\mathcal{O}(d \cdot n)$ or $\mathcal{O}(d^2 \cdot n)$ where d is the average degree of the graph. The complexity for calculation of LCF value and binding the vertices is $\mathcal{O}(\beta \cdot n)$ in the worst case, where β should be smaller than the graph diameter in general. The complexity for vertex level adjustment is between $\mathcal{O}(t \cdot e)$ and $\mathcal{O}(t \cdot n \cdot e)$. t is the number of iterations and is chosen to be a small integer as explained earlier. e is the number of edges in the graph. The complexity for the post-processing refinement is $\mathcal{O}(e)$. To sum up, the complexity for LCF algorithm in the worst case can be $\mathcal{O}(t \cdot n \cdot e)$. However the worst case rarely happens. In normal cases, the complexity is approximately $\mathcal{O}(t \cdot e)$ for sparse graphs. We compare the running time in the experiments to show the efficiency in practice.

4.3.5 Experiment

We conduct experiments on both synthetic graphs and real world graphs. The experiments ran on a cluster of nodes of Intel Xeon 2.27 GHz 64 bit processors.

Table 4.3: Description of data sets

Data sets	Vertices	Edges	Description	Communities
DBLP	317,080	1,049,866	Co-Authors	13,477
Amazon	334,863	925,872	Co-Purchase	151,037

The nodes operate Linus 2.6 with gcc version 4.4.6. The algorithm is implemented in C/C++.

We compare our algorithm to the speaker-listener label propagation algorithm [161] and game theory based algorithm [32], which have been compared to some other algorithms and show better performance [160, 161]. We also compare our algorithm to *BigClam*, a probabilistic model based algorithm [166]. We directly use the original codes provided by authors. We examine the threshold from 0.01 to 0.5, with interval 0.05, and take results with the maximum values of NMI for the *SLPA* algorithm in order to obtain better communities. For the *game-theory* algorithm, we run the algorithm on each graph once with 2,000,000 iterations. Note that our closeness centrality-based algorithm is not evaluated on the set of graphs with increasing size, duo to computational prohibitions.

4.3.5.1 Data sets

We use two real world networks from [165] and a batch of benchmark graphs [90] to evaluate the effectiveness of our method. Table 4.3 shows the information about the two data sets. In DBLP network, an edge exits between two authors (vertices) if they co-authored, and authors are in the same community if they published to the same journal or conference. In the Amazon network, an edge exists between two products (vertices) if they are frequently co-purchased. The products are in the same community if they are in the same product category.

The benchmark graphs are generated with a known community structure including the number of vertices, average degree, number of overlapping vertices, number of memberships of the overlapping vertices, etc. We generate three sets of graphs. Graphs in the first set have 10,000 vertices and five different average degrees varying from 5.8 to 12.6 while the other parameters are the same. We generate five graphs for each average degree. We run the algorithms on all the graphs and take the average of the metric values. Graphs in the second set have a different number of vertices, varying from 10,000 to 50,000. For every number of vertices, we generate five graphs. Graphs in the third set are generated with an increasing number of memberships of the overlapping vertices while the other parameters remain the same. Similarly, for every number of memberships of the overlapping vertices we generate five graphs. We run algorithms on all of them and take the average values of each metric.

4.3.5.2 Metrics

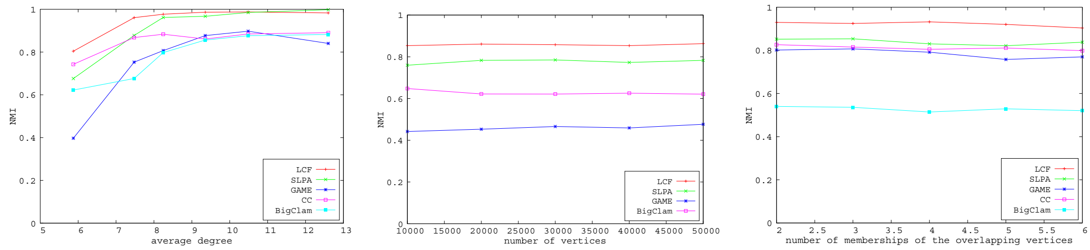
To measure the quality of detected communities, either disjoint communities or overlapping communities, we use the *Normalized Mutual Information (NMI)* [91] and *Omega Index* [39]. We compute the *NMI* and *Omega Index* value based on the communities detected, and the known communities of the input graphs. Therefore we use the benchmarks as the main data sets for the experiments, as the communities of the graphs generated are known.

We also use modularity, conductance, internal density, cut ratio and weighted community clustering to qualify the communities. We compare the size of communities detected with the ground truth as well.

4.3.5.3 Result analysis

Figure 4.23 shows the comparison of the *NMI* values of the detected communities measured on the three sets of graphs. The x-axis shows change in a certain graph property of the graphs. The y-axis shows the NMI value. Each line shows the NMI values of the communities detected by one algorithm. Figure 4.23 (a) shows the *NMI* value changeing with an increasing average degree. The x-axis is the average degree of the graphs. The y-axis is *NMI* values. For each average degree, we have five graphs, so we take the average of the NMI values from each graph. The results show that *LCF* detects communities with the highest *NMI* values on graphs with smaller average degrees. This has been proved further. Figure 4.23 (b) shows that the *NMI* value changes with increasing graph size. These graphs have different sizes but similar average degrees of around 6. The x-axis is the number of vertices. It shows that *LCF* finds communities with the highest values on the graphs of all the sizes tested. Figure 4.23 (c) shows the *NMI* value changes with an increasing number of memberships of the overlapping vertices. The x-axis is the number of memberships of the overlapping vertices. The graphs in this set also have small average degrees of about 7. The figure shows that *LCF* finds communities with the highest *NMI* values.

Figure 4.24 shows a comparison of the *OMEGA* values on the three sets of graphs. Figure 4.24 (a) shows the *OMEGA* value changes with increasing average degrees. The x-axis is the average degree of the graphs. The y-axis is *OMEGA* values. For each average degree, we have five graphs, so we take the average of the OMEGA values. Figure 4.24 shows that *LCF* detects communities with the highest *OMEGA* values on graphs with smaller average degrees. Figure 4.24 (b) shows that the *OMEGA* value changes with increasing graph size. The x-axis is the number of vertices. It shows that *LCF* finds communities with the highest values on the graphs of all the sizes tested. Figure 4.24 (c) shows the *OMEGA* value changes with increasing number of memberships of the overlapping vertices. The x-axis is the number of memberships of the overlapping vertices. Additionally, Figure 4.24

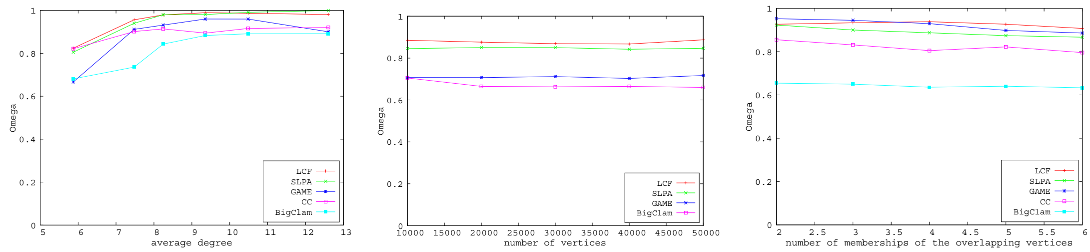


(a) NMI value changes with different graph average degree

(b) NMI value changes with different graph sizes

(c) NMI value changes with different number of memberships of the overlapping vertices

Figure 4.23: Comparison of NMI value on three sets of graphs.



(a) Omega value changes with different graph average degree

(b) Omega value changes with different graph sizes

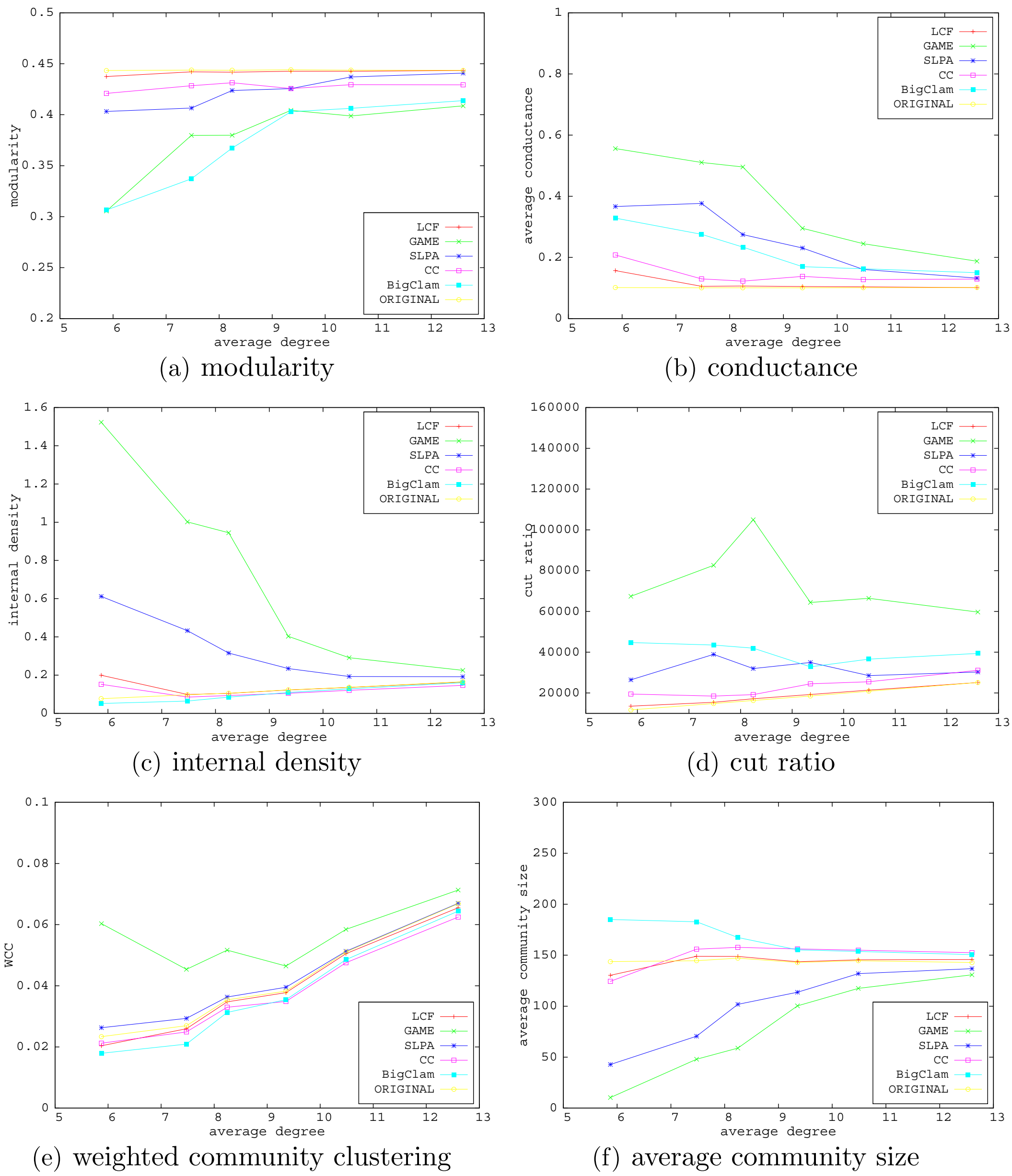
(c) Omega value changes with different number of memberships of overlapping vertices

Figure 4.24: Omega value comparison on three sets of graphs.

(c) shows that communities found by *LCF* have higher *OMEGA* values when the number of memberships of the overlapping vertices is larger, while communities found by *SLPA* have higher *OMEGA* values when the number of memberships of the overlapping vertices is smaller.

The above results show that the *NMI* values for *LCF* are higher than the other algorithms on sparse graphs, which further indicates that the communities found by *FCL* match the actual communities best. The results of *OMEGA* show that the *OMEGA* values for *LCF* are highest, and hence, most truthful communities detected, in the case of sparse graphs and the case of more memberships of the overlapping vertices.

Figure 4.25 shows the measurements of community quality based on the selected



(a) modularity
(b) conductance
(c) internal density
(d) cut ratio
(e) weighted community clustering
(f) average community size

Figure 4.25: Comparison of modularity, conductance, internal density, cut ratio, weighted community clustering, average community size on graphs with different average degree

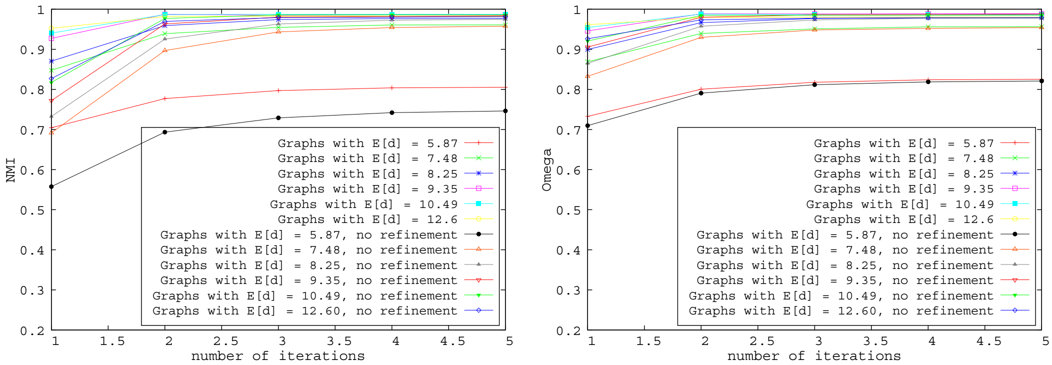


Figure 4.26: Effects of the number of iterations and effects of refinement

metrics. In each sub-figure, the x-axis is the average degree. The y-axis is the value of a metric. Figure 4.25 (a) shows the comparison of modularity. Figure 4.25 (b) shows the comparison of conductance. Figure 4.25 (c) shows the comparison of internal density. Figure 4.25 (d) shows the comparison of cut ratio. Figure 4.25 (e) shows the comparison of weighted community clustering. Figure 4.25 (f) shows the comparison of average community size. We compare the values for the communities detected by the algorithms directly, with the values of the ground truth (which are labelled "Original" in the figures). The figures show that for all the metrics measured, *LCF* produces the values that are closest to the ground truth. This further indicates the high quality of the communities discovered by *LCF* algorithm in sparse graphs.

Figure 4.26 shows how the number of iterations affects the detected communities and corresponding values of NMI and Omega on the graphs with different average degrees, and how the refinement affects them. The x-axis is the number of iterations, and the y-axis is the NMI or Omega value. Each line plots the average values from five graphs with the similar average degrees. The figures show that as the number of iterations increases, the values of NMI and Omega increase, and the differences between the cases with refinement and without refinement decrease.

The comparison on the graphs with a different number of memberships of the

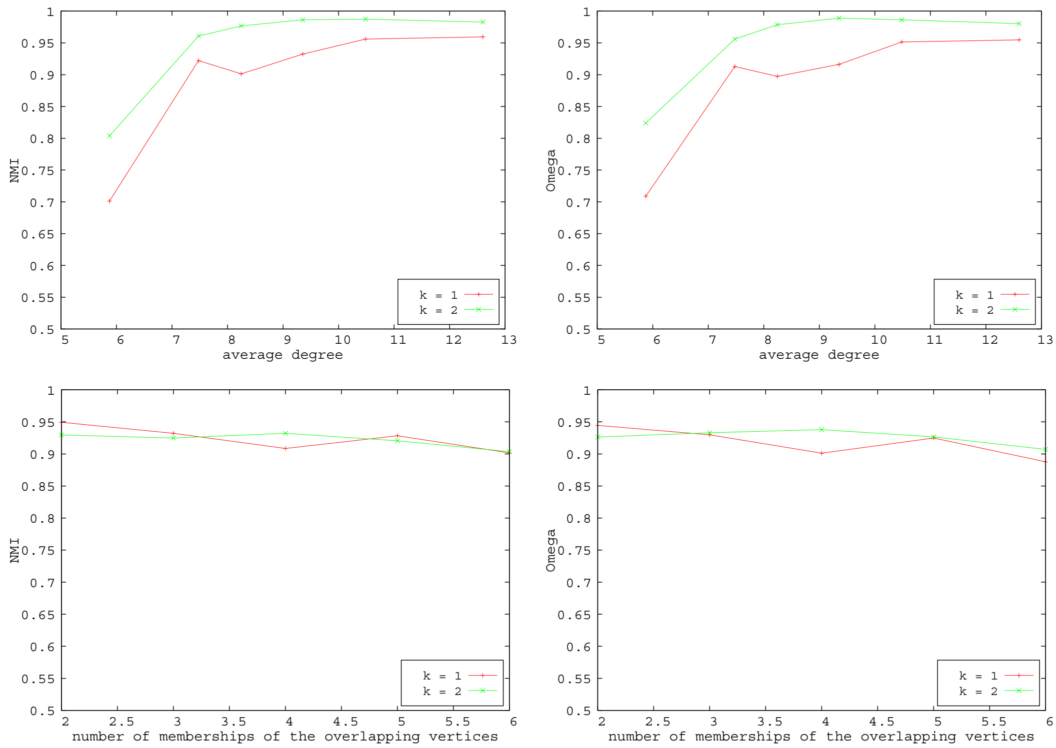


Figure 4.27: Changing values of NMI and Omega with parameter k varying.

overlapping vertices shows similar results, but the increment and decrement are larger, and with the number of iterations increasing, the differences between the cases with refinement and without refinement shrink by a large amount. *NMI* values tend to be stable after the 3rd iteration, and thus setting iteration to 3 or 4 is a good choice, taking both efficiency and effectiveness into consideration. We adopt 4 iterations in the experiments.

Figure 4.27 shows how the choice of parameter k affects the values of NMI and Omega on the graphs with different average degrees and graphs with different numbers of memberships of the overlapping vertices. The x-axis is the average degree or the number memberships of the overlapping vertices. The y-axis is NMI or Omega value. From these two figures, we can see that for the these graphs, setting k to 2 generates highest *OMEGA* and *NMI* value in most cases.

Figure 4.28 shows a comparison of the running time for each algorithm to complete each task of finding communities on the three sets of graphs. The y-axis shows the running time recorded in seconds. Figure 4.28 (a) focuses on the efficiency comparison between the CC algorithm and the other algorithms. We can see that for the same task, the CC algorithm takes a much longer time than the others, and thus, it is not efficient. This demonstrates one of the reasons why we propose the LCF algorithm. Figure 4.28 (b) (c) and (d) show how the LCF algorithm performs the faster than the other algorithms in general. For graphs with 50,000 vertices, *LCF* is capable of finding communities in about 60 seconds and for graphs with 10,000, it takes only a few seconds.

Figure 4.29 shows the comparisons for the real world networks. Figure 4.29 (a) and (b) show a comparison based on NMI values on the DBLP network and Amazon network, separately. The detected communities are compared to all the communities based on the ground truth, or the most important 5000 communities given by [165]. NMI for the *game-theory* algorithm is missing on the Amazon data, due to its scalability. Figure 4.29 (c) shows the running time. The results on these two data sets show that the higher the NMI value, the more time it takes. In terms of both running time and NMI values, LCF performs best in the case of top important communities on the Amazon network. In the other cases, LCF shows a stable performance compared to the other algorithms.

4.3.6 Summary

In this section, we try to propose an algorithm for overlapping community detection in sparse graphs, that leverages closeness centrality. However, using closeness centrality for community detection seems to be inefficient. Therefore, we look for a local notion of closeness centrality to lessen the the problem. We adapt the idea of local density to graph and define the local closeness factor. It is computed by

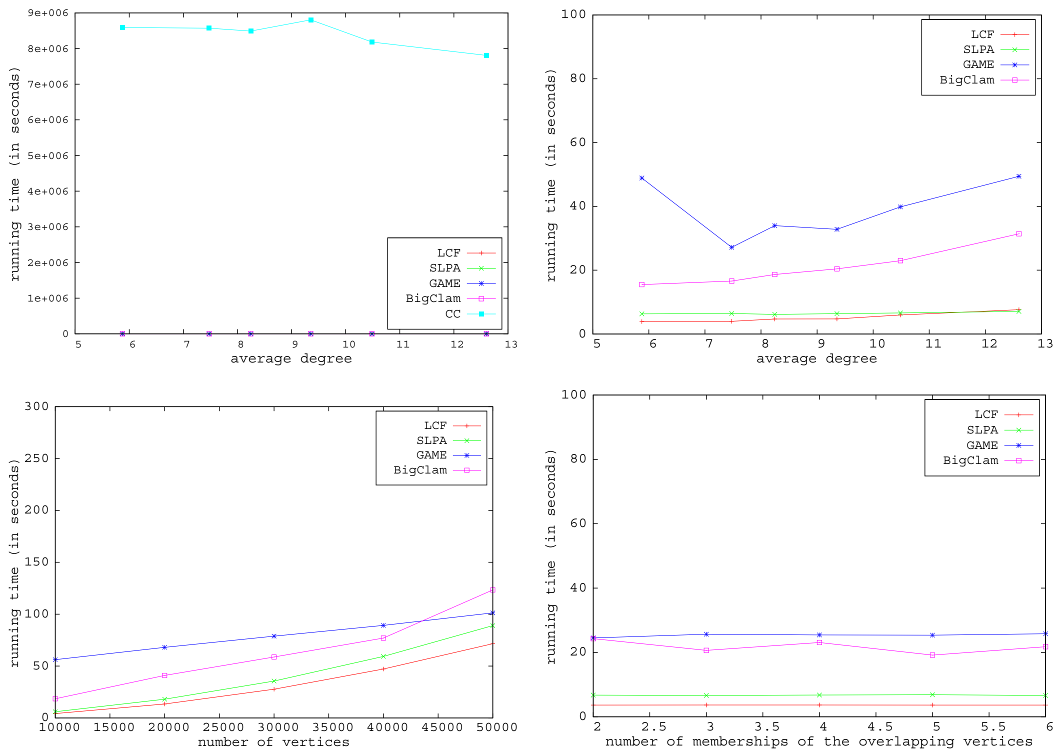
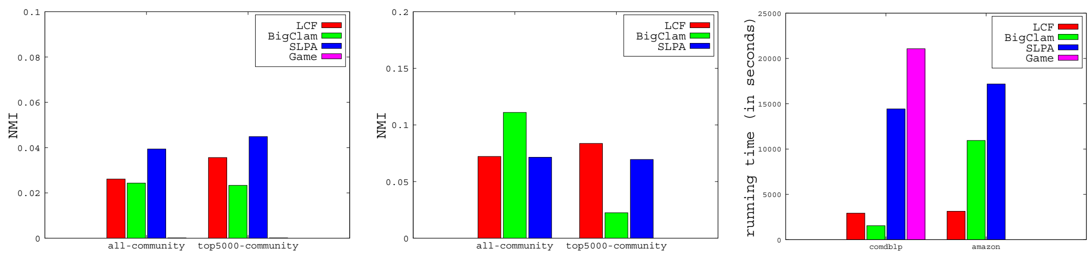


Figure 4.28: Comparison of running time on on three sets of graphs.



(a) DBLP (b) Amazon (c) Running time

Figure 4.29: Comparison of NMI value and running time on real world networks.

computing distance, reachability distance and the density of each vertex within its restricted neighbourhood. We devise our algorithm based on the local closeness factor. We conduct experiments on real world graphs and benchmark graphs with known communities. We show that, on the generated graphs, our solution not only outperforms the algorithm using closeness centrality, but also outperforms *SLPA*, *BigClam* and *game-theory* algorithms in sparse graphs, in terms of effectiveness. *LCF* finds communities closer to the ground truth in large sparse graphs. Efficiency-wise, *LCF* is faster than both algorithms. For a graph with 10,000 vertices, it only takes a few seconds; and about 60 seconds for a graph with 50,000 vertices. Therefore, *LCF* is effective and efficient. On the real world graphs, *LCF* shows competitive performance combining effectiveness and efficiency.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, motivated by the protection of interpersonal trust [97] and institutional trust and the facilitation of building weak ties and bridging social capital [55], we model social networks as simple graphs and study social network analysis on both privacy and utility problems from the graph perspectives. Specifically, we studied graph anonymization problems and community detection problems. We reviewed the concepts and the existing algorithms for each problem, and then separately studied the fundamental problems. We proposed algorithms to solve the problems found. Through a comprehensive experimental study on both the real world network and synthetic data sets, the proposed solutions were shown to be efficient and effective.

In Chapter 3, we discussed the fundamental problems that we found in graph anonymization. We proposed approaches to solve the problems. The main results are revisited as follows.

- We proposed an *FKDA* algorithm to overcome the shortcomings of the KDA [102] algorithm to achieve *k-anonymity* of the graph. *FKDA* anonymizes a graph by simultaneously adding edges and anonymizing its degree sequence in groups of vertices, and thus is significantly more efficient than KDA and more effective than KDA on large graphs. The comparative empirical performance evaluation on three real world graphs verified these results.

- We proposed a *Similar Reachability Graph* algorithm (*SRG*) for revealing a subgraph of connections in a user's neighborhood. We aimed to offer an informative view of the network for users while resisting certain structural attacks. (*SRG*) guarantees the graph reachability-related utility while it perturbs the graphs to certain extent to defend against attacks.

We do not claim that our solutions are a panacea for graph anonymization in general, given the generality of background knowledge potentially available to adversaries. However, they are effective and efficient solutions for the protection of privacy in the presence of certain background knowledge such as vertex degrees.

In Chapter 4, we discussed the fundamental problems that we found in community detection. We proposed novel algorithms for the related problems. The main results are revisited as follows.

- We proposed the *FR-KM* algorithm based on the Force-directed graph drawing method, which was inspired by the idea of dimension reduction. The algorithm projects the graph onto Euclidean space, and clusters the vertices according to their Euclidean distance. Real world case studies and empirical comparison with the state-of-the-art algorithms confirmed that our algorithm is efficient and reasonably effective for finding communities in the networks.

- We proposed the *Fast Community Detection Algorithm (FCD)* and its parallel version to detect communities in the networks that are modeled as simple graphs. The algorithm is vertex centric. It initiates each vertex to independently seek the community in its neighbourhood. The empirical experiment results state that the algorithms find communities with comparable quality, and are the fastest in general, compared to the *InfoMap*, *WalkTrap* and *GN* algorithms.

- We proposed the *Local Closeness Factor Algorithm (LCF)* for community detection in sparse graphs that leverages a local notion of closeness centrality to lessen the the problem. We adapted the idea of local density to graph and define the local closeness factor. It is computed by computing distance, reachability distance and the density of each vertex within its restricted neighbourhood. The empirical experiment results state the effectiveness and efficiency of the *LCF* algorithm on large sparse graphs.

In conclusion, by leveraging graph features and structural properties, we can design effective and efficient methods to better understand the connectedness of the social networks, and thereafter benefit benign users.

5.2 Future Work

Social networks are temporal and dynamic in essence. Evolving networks have a high potential for capturing natural and social phenomena over time. Examining the structural changes (e.g. evolving communities) over time provides insights into structural evolution patterns, factors causing the changes, and ultimately predict the future structure of the network.

Chakrabarti et al. [30], one of the pioneers, studied the evolutionary clustering on attributed data, and proposed a framework incorporating the *temporal smoothness* in the clustering process. Backstrom et al. [8] studied community evolution in social networks with known communities. At a personal level, they investigate the relationship between individuals' decisions to join communities and network structures. At a global level, they looked into the evolution of community membership and content. Sun et al. [140] in 2007 proposed one of the first approaches towards community detection in dynamic graphs. Thereafter, density-based methods [86, 58, 29], modularity methods [49, 74, 116, 144] and generative-model-based methods [167, 101] have been proposed. Some authors adapted their methods from static scenarios to dynamic scenarios. For example, [121] [159] were extended from Label Propagation algorithms [162, 127, 51] are extended from [47], and [34] was extended from spectral graph clustering.

However the lack of the proper benchmarks or thorough and empirical comparisons of the existing methods brings difficulties to users when choosing the appropriate methods for applications.

We would like to work on these specific problems in the future: generation of dynamic synthetic benchmarks incorporating ground truth clusterings, dynamic community detection that capture both the current community structure and evolution patterns, evaluation of dynamic community detection algorithms as well as future structure prediction.

On the other hand, while this thesis has presented solutions on simple graph models, we would like to extend the work to richer models that contain more than just structural information, for example, weighted graphs or attributed graphs.

Bibliography

[1] Email-urv. `http://deim.urv.cat/aarenas/data/welcome.htm`.

[2] R. Agrawal, S. Rajagopalan, R. Srikant, and Y. Xu. Mining newsgroups using networks arising from social behavior. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 529–535. ACM, 2003.

[3] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multi-scale complexity in networks. *NATURE*, 466:761, 2010.

[4] A.Narayanan and V.Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy*, 2009.

[5] A.P.Dempster, N.M.Laird, and D.B.Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical society, Series B*, 1977.

[6] J. A. Aslam, E. Pelekhov, and D. Rus. The star clustering algorithm for static and dynamic information organization. *J. Graph Algorithms Appl.*, 8:95–129, 2004.

[7] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou R3579X?: Anonymized social networks, hidden patterns, and structural steganography. In *WWW*, pages 181–190. ACM, 2007.

[8] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 44–54, New York, NY, USA, 2006. ACM.

[9] A.-L. Barabási and R. Albert. Emergence of Scaling in Random Networks. *Science*, 286:509–512, 1999.

[10] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1998.

[11] J. Baumes, M. K. Goldberg, M. S. Krishnamoorthy, M. Magdon-Ismail, and N. Preston. Finding communities by clustering a graph into overlapping subgraphs. In *IADIS AC*, pages 97–104, 2005.

[12] J. Baumes, M. K. Goldberg, and M. Magdon-Ismail. Efficient identification of overlapping communities. In *ISI*, pages 27–36, 2005.

[13] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Kluwer Academic Publishers, 1981.

[14] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. S. and. Class-based graph anonymization for social network data. *PVLDB*, 2(1), 2009.

[15] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. Privacy in dynamic social networks. In *WWW*, pages 1059–1060. ACM, 2010.

[16] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[17] B. Boden, S. Günnemann, and T. Seidl. Tracing clusters in evolving graphs with node attributes. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 2331–2334. ACM, 2012.

[18] P. Boldi and S. Vigna. Axioms for centrality. *CoRR*, 2013.

[19] B. Bollobas. *Modern Graph Theory*. Springer, 1998.

[20] F. Bonchi, A. Gionis, and T. Tassa. Identity obfuscation in graphs through the information theoretic lens. In *ICDE*, 2011.

[21] D. M. Boyd and N. B. Ellison. Social network sites: Definition, history, and scholarship. *J. of Computer-Mediated Communication*, 13:210–230, 2008.

[22] U. Brandes and C. Pich. Eigensolver methods for progressive multidimensional scaling of large data. In *Proceedings of the 14th International Con-*

ference on Graph Drawing, GD'06, pages 42–53, Berlin, Heidelberg, 2007. Springer-Verlag.

[23] U. Brandes and C. Pich. An experimental study on distance-based graph drawing. In *Graph Drawing*, pages 218–229, 2008.

[24] M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM Sigmod International Conference on Management of Data*. ACM, 2000.

[25] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international conference on WWW*, 1998.

[26] J. J. Brown and P. H. Reingen. Social Ties and Word-of-Mouth Referral Behavior. *The Journal of Consumer Research*, 14(3):350–362, Dec. 1987.

[27] A. Campan and T. M. Truta. A clustering approach for data and structural anonymity in social networks. In *PinKDD*, 2008.

[28] A. Campan and T. M. Truta. Data and structural k-anonymity in social networks. In *PinKDD*, pages 33–54, 2008.

[29] R. Cazabet, F. Amblard, and C. Hanachi. Detection of overlapping communities in dynamical social networks. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 309–314, Aug 2010.

[30] D. Chakrabarti, R. Kumar, and A. Tomkins. Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 554–560, New York, NY, USA, 2006. ACM.

[31] B.-C. Chen, D. Kifer, K. LeFevre, and A. Machanavajjhala. Privacy-preserving data publishing. *Found. Trends databases*, 2(1-2):1–167, January 2009.

[32] W. Chen, Z. Liu, X. Sun, and Y. Wang. A game-theoretic framework to identify overlapping communities in social networks. *Data Min. Knowl. Discov.*, 2010.

[33] J. Cheng, A. W.-C. Fu, and J. Liu. K-isomorphism: privacy-preserving network publication against structural attacks. In *SIGMOD*, 2010.

[34] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 153–162, New York, NY, USA, 2007. ACM.

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

[36] A. Clauset. Finding local community structure in networks. *PHYS.REV.E*, 72:026132, 2005.

[37] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *PHYS.REV.E*, 70:066111, 2004.

[38] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Reviews*, 2007.

[39] L. M. Collins and C. W. Dent. Omega: A general formulation of the rand index of cluster recovery suitable for non-disjoint solutions. *Multivariate Behavioral Research*, 23(2):231–242, 1988.

[40] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. *PVLDB*, 19(1), 2010.

[41] M. Coscia, G. Rossetti, F. Giannotti, and D. Pedreschi. Demon: a local-first discovery method for overlapping communities. *CoRR*, 2012.

[42] CUDA-Zone. http://www.nvidia.com/object/what_is_cuda_new.html.

[43] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang. Online search of overlapping communities. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 277–288. ACM, 2013.

[44] A. Cuzzocrea and F. Folino. Community evolution detection in time-evolving information networks. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, EDBT '13. ACM, 2013.

[45] C. C.Yang and X. Tang. Social networks integration and privacy preservation using subgraph generalization. In *KDD-WS*, 2009.

[46] S. Das, Ö. Egecioglu, and A. E. Abbadi. Anonymizing weighted social network graphs. In *ICDE*, 2010.

[47] I. Derényi, G. Palla, and T. Vicsek. Clique percolation in random networks. *Physical Review Letters*, 2005.

[48] K. Dimitrios. Greek construction firms formation and topological analysis of a collaboration network. *International Research Journal of Finance and Economics*, 2010.

[49] T. Dinh, I. Shin, N. Thai, M. Thai, and T. Znati. A general approach for modules identification in evolving networks. In *Dynamics of Information Systems*, volume 40 of *Springer Optimization and Its Applications*. Springer New York, 2010.

[50] N. Du, B. Wu, X. Pei, B. Wang, and L. Xu. Community detection in large-scale social networks. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, WebKDD/SNA-KDD '07, pages 16–25. ACM, 2007.

[51] D. Duan, Y. Li, R. Li, and Z. Lu. Incremental k-clique clustering in dynamic social networks. *Artif. Intell. Rev.*, pages 129–147, 2012.

[52] P. A. Eades. A heuristic for graph drawing. In *Congressus Numerantium*, volume 42, 1984.

[53] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

[54] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. Natl. Acad. Sci. U.S.A.*, 1998.

[55] N. B. Ellison, C. Steinfield, and C. Lampe. The benefits of facebook "friends": Social capital and college students' use of online social network sites. *J. of Computer-Mediated Communication*, 12:1143–1168, 2007.

[56] B. Etling, J. Kelly, R. Faris, and P. John. Mapping the arabic blogosphere: Politics, culture, and dissent. *Berkman Center Research Publication*, 2009.

[57] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, 8:128–140, 1741.

[58] T. Falkowski, A. Barth, and M. Spiliopoulou. Dengraph: A density-based community detection algorithm. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, WI '07, pages 112–115, Washington, DC, USA, 2007. IEEE Computer Society.

[59] Flickr. http://www.flickr.com/.

[60] R. W. Floyd. Algorithm 97: Shortest path. *Comm. of the ACM*, 5(6):345, 1962.

[61] S. Fortunato and C. Castellano. Community structure in graphs. *Physics Report*, 2010.

[62] S. Fortunato and A. Lancichinetti. Community detection algorithms: a comparative analysis: invited presentation, extended abstract. In *VALUETOOLS '09*. ICST, 2009.

[63] L. C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1978.

[64] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, 1991.

[65] F. Fukuyama. *Trust: The Social Virtues and the Creation of Prosperity*. Free Press, New York, first edition, 1995.

[66] E. R. Gansner, E. Koutsofios, S. C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Trans. Softw. Eng.*, 19(3):214–230, 1993.

[67] I. F. Gergely Palla, Imre Derenyi and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, June 2005.

[68] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis. Fast data anonymization with low information loss. In *VLDB*, pages 758–769. VLDB Endowment, 2007.

[69] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 2002.

[70] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, New York, NY, USA, 2012. ACM.

[71] S. Goel, R. Muhamad, and D. Watts. Social search in "small-world" experiments. In *WWW*, pages 701–710. ACM, 2009.

[72] M. K. Goldberg, S. Kelley, M. Magdon-Ismail, K. Mertsalov, and A. Wallace. Finding overlapping communities in social networks. In *SocialCom/PASSAT*, pages 104–113, 2010.

[73] M. Gomez Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 1019–1028, New York, NY, USA, 2010. ACM.

[74] R. Görke, P. Maillard, A. Schumm, C. Staudt, and D. Wagner. Dynamic graph clustering combining modularity and smoothness. *J. Exp. Algorithmics*, 18, 2013.

[75] M. Granovetter. The strength of weak ties: A network theory revisited. *Sociological Theory*, 1:201–233, 1983.

[76] S. Gregory. An algorithm to find overlapping community structure in networks. In *PKDD 2007*, pages 91–102. Springer-Verlag, 2007.

[77] S. Gregory. A fast algorithm to find overlapping communities in networks. In *ECML PKDD '08*, pages 408–423. Springer-Verlag, 2008.

[78] S. Günnemann, B. Boden, and T. Seidl. Db-csc: A density-based approach for subspace clustering in graphs with feature vectors. In *ECML/PKDD (1)*, pages 565–580, 2011.

[79] S. Gürses and B. Berendt. The Social Web and Privacy: Practice, Reciprocity and Conflicts in Social Networks. In F. Bonchi and E. Ferrari, editors, *Privacy-Aware Knowledge Discovery: Novel Applications and New Techiques*. Chapman and Hall/CRC, 2010.

[80] S. Hachul and M. Jinger. Drawing large graphs with a potential-field-based multilevel algorithm. In *Graph Drawing*, Lecture Notes in Computer Science, pages 285–295. Springer, 2004.

[81] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis. Resisting structural re-identification in anonymized social networks. *PVLDB*, 1(1):102–114, 2008.

[82] P. W. Holland and S. Leinhardt. Transitivity in Structural Models of Small Groups. *Small Group Research*, 2(2):107–124, 1971.

[83] Y. F. Hu. Efficient and high quality force-directed graph drawing. *The Mathematica Journal*, 10:37–71, 2005.

[84] D. Jin, B. Yang, C. Baquero, D. Liu, D. He, and J. Liu. A Markov random walk under constraint for discovering overlapping communities in complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011, 2011.

[85] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Inf. Process. Lett.*, 31(1):7–15, 1989.

[86] M.-S. Kim and J. Han. A particle-and-density based evolutionary clustering method for dynamic networks. *Proc. VLDB Endow.*, 2(1):622–633, Aug. 2009.

[87] S. G. Kobourov. Force-directed drawing algorithms. *Handbook of Graph Drawing and Visualization*, 2013.

[88] A. Korolova, R. Motwani, S. U. Nabar, and Y. Xu. Link privacy in social networks. In *CIKM*, 2008.

[89] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.*, 11(3):243–271, 2005.

[90] A. Lancichinetti and S. Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E*, 80(1):016118, 2009.

[91] A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11, 2009.

[92] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78, 2008.

[93] M. Leng, Y. Yao, J. Cheng, L. Weiming, and X. Chen. Active semi-supervised community detection algorithm with label propagation. In *DASFAA*, 2013.

[94] J. Leskovec and E. Horvitz. Planetary-scale views on a large instant-messaging network. In *WWW*. ACM, 2008.

[95] J. Leskovec, D. Huttenlocher, and J. Kleinberg. Predicting positive and negative links in online social networks. In *Proceedings of the 19th international conference on World Wide Web*. ACM, 2010.

[96] J. Leskovec, J. M. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *TKDD*, 1(1), 2007.

[97] D. Z. Levin and R. Cross. The strength of weak ties you can trust: The mediating role of trust in effective knowledge transfer. *Management Science*, 50(11):1477–1490, 2004.

[98] N. Li, T. Li, and S. Venkatasubramanian. t-closeness:privacy beyond k-anonymity and l-diversity. In *IEEE 23rd International Conference on Data Engineering*, 2007.

[99] X.-L. Li, A. Tan, P. S. Yu, and S.-K. Ng. Ecode: event-based community detection from social networks. In *Proceedings of the 16th international conference on Database systems for advanced applications - Volume Part I*, DASFAA'11, pages 22–37. Springer-Verlag, 2011.

[100] Y. Li and H. Shen. Anonymizing graphs against weight-based attacks. In *ICDM Workshops*, 2010.

[101] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng. Analyzing communities and their evolutions in dynamic social networks. *ACM Trans. Knowl. Discov. Data*, 3(2):8:1–8:31, Apr. 2009.

[102] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD Conference*, pages 93–106, 2008.

[103] L. Liu, J. Wang, J. Liu, and J. Zhang. Privacy preservation in social networks with sensitive edge weights. In *SIAM International Conference on Data Mining*, 2009.

[104] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[105] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

[106] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. ℓ-diversity: privacy beyond k-anonymity. In *ICDE*, 2006.

[107] K. Macropol, T. Can, and A. K. Singh. Rrw: repeated random walks on genome-scale protein networks for local cluster discovery. *BMC Bioinformatics*, 2009.

[108] P. Massa and P. Avesani. Trust metrics in recommender systems. In *Computing with Social Trust*. Springer London, 2009.

[109] M.Hay, G.Miklau, D.Jensen, P.Weis, and S.Srivastava. Anonymizing social networks. Technical report, Computer Science Department, University of Massachusetts Amherst, 2007.

[110] S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

[111] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, 2007.

[112] MPI. http://socialnetworks.mpi-sws.org/.

[113] Networkx. http://networkx.lanl.gov/.

[114] M. Newman and M.Girvan. Finding and evaluating community structure in networks. *PHYS.REV.E*, 69:026113, 2004.

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

[116] N. P. Nguyen, T. N. Dinh, Y. Xuan, and M. T. Thai. Adaptive algorithms for detecting community structure in dynamic social networks. In *INFOCOM*, pages 2282–2290, 2011.

[117] V. Nicosia, G. Mangioni.V.Carchiolo, and M. Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. *Journal of statistical Mechanics: Theory And Experiment*, 2009.

[118] A. Noack. An energy model for visual graph clustering. In *Graph Drawing*, pages 425–436, 2003.

[119] K. Okamoto, W. Chen, and X. Li. Ranking of closeness centrality for large-scale social networks. In *Proceedings of the 2Nd Annual International Workshop on Frontiers in Algorithmics*, FAW '08, pages 186–195, Berlin, Heidelberg, 2008. Springer-Verlag.

[120] C. R. Palmer, P. B. Gibbons, and C. Faloutsos. ANF: a fast and scalable tool for data mining in massive graphs. In *KDD*. ACM, 2002.

[121] S. Pang, C. Chen, and T. Wei. A realtime community detection algorithm: incremental label propagation. In *First International Conference on Future Information Networks ICFIN 2009*, pages 313–317, Oct 2009.

[122] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos. Community detection in social media. *Data Min. Knowl. Discov.*, 24(3), May 2012.

[123] P. Pons and M. Latapy. Computing communities in large networks using random walks. In *ISCIS*, 2005.

[124] A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey. Shaping communities out of triangles. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12. ACM, 2012.

[125] A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey. Shaping communities out of triangles. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12. ACM, 2012.

[126] G.-J. Qi, C. C. Aggarwal, and T. S. Huang. Community detection with edge content in social media networks. In *ICDE*, pages 534–545, 2012.

[127] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3), Sept. 2007.

[128] P. K. Reddy, M. Kitsuregawa, P. Sreekanth, and S. S. Rao. A graph based approach to extract a neighborhood customer community for collaborative filtering. In *DNIS*, DNIS '02, pages 188–200. Springer-Verlag, 2002.

[129] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences of the United States of America*, 105, 2008.

[130] Y. Ruan, D. Fuhry, and S. Parthasarathy. Efficient community detection in large networks using content and links. *CoRR*, 2012.

[131] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, pages 59–66, 1998.

[132] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *Intl Journal of Computer Vision*, 40(2):99–121, 2000.

[133] G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, December 1966.

[134] I. Safro, D. Ron, and A. Brandt. Multilevel algorithms for linear ordering problems. *J. Exp. Algorithmics*, 13:4:1.4–4:1.20, 2009.

[135] J. P. Scott. *Social Network Analysis: A Handbook*. SAGE Publications, Jan. 2000.

[136] SNAP. http://snap.stanford.edu/data.

[137] Y. Song, P. Karras, Q. Xiao, and S. Bressan. Sensitive label privacy protection on social network data. Technical report, 2012.

[138] G. Su, A. Kuchinsky, J. H. Morris, D. J. States, and F. Meng. Glay: community structure analysis of biological networks. *Bioinformatics*, 26(24):3135–3137, 2010.

[139] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man & Cybernetics*, pages 109–125, 1981.

[140] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu. Graphscope: Parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 687–696, New York, NY, USA, 2007. ACM.

[141] H. Sundaram, Y.-R. Lin, M. D. Choudhury, and A. Kelliher. Understanding community dynamics in online social networks: A multidisciplinary review. *IEEE Signal Process. Mag.*, 29(2):33–40, 2012.

[142] L. Sweeney. *K*-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5), 2002.

[143] C.-H. Tai, P. S. Yu, D.-N. Yang, and M.-S. Chen. Privacy-preserving social network publication against friendship attacks. In *SIGKDD*, 2011.

[144] M. Takaffoli, R. Rabbany, and O. R. Zaïane. Incremental local community identification in dynamic social networks. In *ASONAM*, pages 90–94, 2013.

[145] A. L. Traud, E. D. Kelsic, P. J. Mucha, and M. A. Porter. Comparing community structure to characteristics in online collegiate social networks. *SIAM Review*, 53(3):526–543, 2011.

[146] TrustLet. http://www.trustlet.org/.

[147] W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, pages 743–767, 1963.

[148] J. R. Tyler, D. M. Wilkinson, and B. A. Huberman. Email as spectroscopy: automated discovery of community structure within organizations. In *Communities and technologies*, pages 81–96. Kluwer, B.V., 2003.

[149] S. M. van Dongen. *Graph clustering by flow simulation*. PhD thesis, University of Utrecht, 2000.

[150] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *WOSN*, 2009.

[151] C. Walshaw. A multilevel algorithm for force-directed graph drawing. In *Proceedings of the 8th International Symposium on Graph Drawing*, GD '00, pages 171–182, London, UK, UK, 2001. Springer-Verlag.

[152] J. H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, 1963.

[153] S. Wasserman, K. Faust, and D. Iacobucci. *Social Network Analysis : Methods and Applications* . Cambridge University Press, 1994.

[154] D. J. Watts. *Six Degrees: The Science of a Connected Age*. Norton, 2003.

[155] D. J. Watts and S. H. Strogatz. Collective dynamics of'small-world'networks. *Nature*, 393(6684):409–10, 1998.

[156] R. S. Weiss and E. Jacobson. A method for the analysis of the structure of complex organizations. *American Sociological Review*, 20(6), Dec. 1955.

[157] B. Wellman. *Networks In The Global Village: Life In Contemporary Communities*. Westview Press, 1999.

[158] W. Wu, Y. Xiao, W. Wang, Z. He, and Z. Wang. K-symmetry model for identity anonymization in social networks. In *EDBT*, 2010.

[159] J. Xie, M. Chen, and B. K. Szymanski. Labelrankt: Incremental community detection in dynamic networks via label propagation. *CoRR*, 2013.

[160] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Comput. Surv.*, 45, 2013.

[161] J. Xie and B. K. Szymanski. Towards linear time overlapping community detection in social networks. In *PAKDD'12*. Springer-Verlag, 2012.

[162] J. Xie and B. K. Szymanski. Labelrank: A stabilized label propagation algorithm for community detection in networks. In *NSW*, 2013.

[163] N. B. Yahia, N. Bellamine, and H. B. Ghesala. Combined use of community detection and particle swarm optimization to support decision making. *Journal of computing*, pages 157–163–43, 2012.

[164] B. Yan and S. Gregory. Detecting communities in networks by merging cliques. *CoRR*, 2012.

[165] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12. ACM, 2012.

[166] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM*, 2013.

[167] T. Yang, Y. Chi, S. Zhu, Y. Gong, and R. Jin. A bayesian approach toward finding communities and their evolutions in dynamic social networks. In *SIAM Conference on Data Mining (SDM)*, 2009.

[168] X. Ying, K. Pan, X. Wu, and L. Guo. Comparisons of randomization and k-degree anonymization schemes for privacy preserving social network publishing. In *SNA-KDD*, 2009.

[169] X. Ying and X. Wu. Randomizing social networks: a spectrum perserving approach. In *SDM*, 2008.

[170] X. Ying and X. Wu. On link privacy in randomizing social networks. In *PAKDD*, 2009.

[171] M. Yuan, L. Chen, and P. S. Yu. Personalized privacy protection in social networks. *PVLDB*, 4(2), 2010.

[172] L. Zhang and W. Zhang. Edge anonymity in social network graphs. In *CSE*. IEEE Computer Society, 2009.

[173] S. Zhang, R. S. Wang, and X. S. Zhang. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A: Statistical Mechanics and its Applications*, 374(1):483–490, 2007.

[174] E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *PinKDD*, 2007.

[175] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *ICDE*. IEEE Computer Society, 2008.

[176] B. Zhou and J. Pei. The k-anonymity and ℓ-diversity approaches for privacy preservation in social networks against neighborhood attacks. *Knowledge and Information Systems*, 28(1), 2010.

[177] L. Zou, L. Chen, , and M. T. Özsu. K-automorphism: a general framework for privacy-preserving network publication. *PVLDB*, 2(1), 2009.

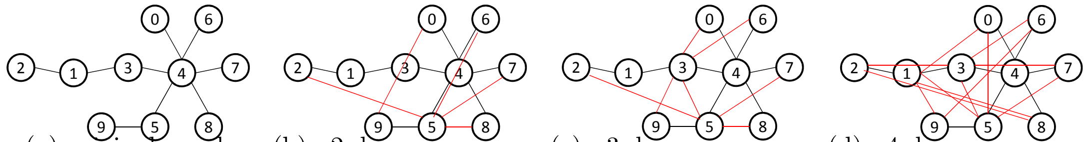
Appendix A

On the Privacy and Utility of Anonymized Social Networks

We try to empirically quantify the trade-off for the k-degree anonymity algorithm [102] and k-automorphism algorithm [177]. The k-degree anonymity algorithm transforms the original graph into one in which, at least, k vertices have the same degree. The transformed graph is k-degree anonymous. The k-automorphism algorithm is a state-of-the-art algorithm that protects against most structural attacks. The algorithm transforms the original graph into one in which, at least, k subgraphs are structurally identical. The transformed graph is k-automorphic.

A.1 k-degree anonymity

The k-degree-anonymity approach [102] aims to prevent attacks involving adversary knowledge of degrees. Before the graph is released, the original one is transformed so that an adversary cannot identify the vertices based on her knowledge of the degree of the published graph. It is transformed by adding edges, in such a way that each vertex has the same degree as at least $k-1$ other vertices. In other



(a) original graph (b) 2-degree anonymous graph (c) 3-degree anonymous graph (d) 4-degree anonymous graph

Figure A.1: k-degree anonymous graphs, for various values of k

words, a vertex cannot be identified with a probability higher than $1/k$ based on degree. The transformed graph is k-degree-anonymous.

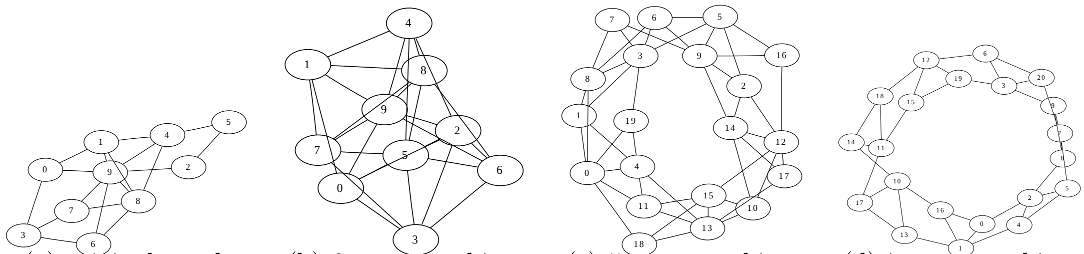
The k-degree-anonymity algorithm is described in Section 3.1. Figure A.1a shows a graph with degree sequence [1,2,1,2,6,2,1,1,1,1]. Figure A.1b shows its 2-degree anonymous graph with degree sequence [2,2,2,2,6,6,2,2,2,2]. Figure A.1c shows its 3-degree anonymous graph with degree sequence [2,2,2,6,6,6,2,2,2,2], and Figure A.1d shows its 4-degree anonymous graph with degree sequence [3,6,3,6,6,6,6,3,3,3].

A.2 k-automorphism

This approach [177] aims to prevent structural attacks, namely attacks involving adversary knowledge such as degree, neighbors, shortest-distances from hubs and so on [177]. Before it is published, the original graph is transformed so that an adversary cannot identify vertices based on her knowledge of the structure of the published graph. It is transformed by adding and removing edges (and possibly vertices), in a way that each vertex is structurally undistinguishable from at least $k-1$ other vertices. In other words, a vertex cannot be identified with probability higher than $1/k$ based on the graph structure. The algorithm transforms the original graph into one in which every subgraph is structurally identical to $k-1$ other subgraphs. The transformed graph is k-automorphic.

The k-automorphism algorithm starts from a naive anonymized graph. It partitions the naive anonymized graph into blocks, and then it makes groups of at least k blocks. For each group, all blocks in the group are made isomorphic by alignment. Consequently, each vertex in one block has a symmetric vertex in, at least, each of the $k-1$ other blocks in the same group. Every vertex has, at least, $k-1$ symmetric vertices in total. Dummy vertices may be introduced in this step. After obtaining isomorphic subgraphs in each group, the edges across blocks are considered. If there exists an edge between v_1 in block i and v_2 in block j in a group, then edges are inserted to make sure that all of v_1's the symmetric vertices in other blocks have edges with the corresponding vertices, the symmetric vertices of v_2. Finally, the anonymized graph is the graph obtained after alignment and edge-copy on all groups.

To preserve utility and minimize information loss, the algorithm adopts a greedy method, together with the notion of *frequent sub-graph* [89] in the first stage of the algorithm, graph partition, and block grouping. In each iteration, frequent sub-graphs with minimum support k are extracted from the whole graph and then expanded hop-by-hop in parallel, unless the overall grouping cost increases. Because of the expansion of blocks, there may be fewer edges crossing over blocks, less edges are inserted in the edge-copy stage, but these larger size of blocks also means higher costs in graph alignment. Therefore, whether an overall optimal anonymization cost is achieved is part of the criterion for block expansion. The final resulting blocks are clustered into one group.



(a) original graph (b) 2-automorphism (c) 5-automorphism (d) 7-automorphism

Figure A.2: k-automorphic graphs, for various values of k

Figure A.2b shows a 2-automorphic graph with vertices 0, 1, 2, 5, 6 which have been partitioned into one block, and vertices 3, 4, 7, 8, and 9 in another block. After graph alignment, vertices 3 and 1 are symmetric, and so are vertices 6 and 4, 7 and 0, 9 and 5, and 8 and 2. No additional vertices are added in this case. Only edges are inserted. But in the case of 5-automorphism (Figure A.2c), dummy vertices are added. When modifying the graph to create a 5-automorphic graph, the original vertices 0, 1, 4 have been put in one block, and vertices 3, 6, 8 one block, and vertices 2, 5, 9 in one block. After anonymization, together with the added vertices, vertices 1, 2, 6, 10, 11 become symmetric, vertices 0, 3, 9, 12, 13 become symmetric, vertices 4, 5, 8, 14, 15 become symmetric, and the remaining vertices become symmetric. Throughout this work, the whole graph is one group.

A.3 Utility Metrics

We are concerned in this work with the structural properties of the social network. We therefore consider the following utility metrics: diameter, radius, density, degree centrality, closeness centrality, betweenness centrality, eigenvector centrality, clustering coefficient, mean geodesic distance, algebraic connectivity, earth mover's distance and edit distance. These metrics quantify various structural aspects and properties of the graph, such as connectivity and centrality. These features are typically used by analysts, for instance, studying the influence, power, authority, engagement and communication, as they quantify the size and shape of social circles.

A.4 Data Sets

The main motivation of this work is to quantify the trade-off between privacy and utility for real social networks. We study snapshots of six social media: Facebook, Epinions, Wikipedia, Orkut, Enron and URV.

The **Facebook** data set contains user-to-user links from Facebook New Orleans networks. It was collected by Viswa-nath et al. [150] from December 29th, 2008 to January 3rd, 2009. We obtained it from *MPI* [112]. The graph is undirected. It has 90,269 vertices and 3,646,662 edges. Each vertex represents a user. An edge exists if two users are friends. It was crawled using a breadth-first search: visiting all the friends of one starting single user and iteratively visiting their friends. Due to the privacy policy, only those users who made their profiles visible to the network could be visited and crawled. We randomly sampled ten graphs from the original network data, with 6,339 vertices and an average of 34,539 edges in each graph.

The **Epinions** data set contains user-to-user (who-trust-whom) links from the Epinions network. It was collected by Epinions staff P. Massa. We obtained it from the *trustlet* website [146][108]. The whole graph is directed and edge-labeled. It has about 132,000 vertices and 487,372 edges. Each vertex represents a user. An edge corresponds to a trust/distrust statement from one user to another user, since users of the Epinions website, a product review website, can comment on products as well as other users' comments and make statements about whether they trust others. An edge from vertex i to vertex j labeled with value 1 means a trust statement was made by vertex i stating his/her appreciation about the content or the behavior of the other user. Distrust statement means the opposite situation. While we measured the utility metrics on the samples of the whole data set, we also divided the data set into two parts, and measured the utility metrics on them separately: one with trust statements only and the other one with distrust statements only.

The **Wikipedia** data set contains user-to-user (who-vote-whom) links from the Wikipedia network. It was collected by Leskove et al. [95] in January, 2008. We obtained it from the SNAP website [136]. The graph is directed. It has 7,115 vertices and 103,689 edges. Each vertex represents a user. An edge is created from a user to a candidate if a user votes for Wikipedia admin candidates. As the original data set contains several components, we use the largest one among them which has 7,066 vertices and 100,736 edges.

The **Orkut** data set contains user-to-user links from Orkut network. It was collected by Mislove et al. [111] from October 3rd to November 11th, 2006. We obtained it from *MPI* [112]. The graph is undirected. It has 3,072,441 vertices and 223,534,301 edges. Each vertex represents a user. An edge is created between two users if they list each other as friends. The data set was crawled with a breadth-first search. The crawling was conducted using HTML screen-scraping technique. Similarly we extract ten sample graphs with 9,217 vertices and, on average, 19,550 edges in each graph.

The remaining two data sets are networks of email exchanges: each vertex corresponds to an email address, and edges correspond to messages between email addresses. The **Email-Enron** data set contains user-to-user (address-to-address) links. It was made public by the Federal Energy Regulatory Commission during its investigations. We obtained it from [136]. The graph is undirected. It has 36,692 vertices and 367,662 edges. Each vertex represents an email address. An edge exists between vertex i and vertex j, if address i sends at least one email message to address j. We excerpt ten sample graphs with 10,108 vertices and on average 180,811 edges in each graph.

The **Email-URV** data set contains user-to-user (address-to-address) links from

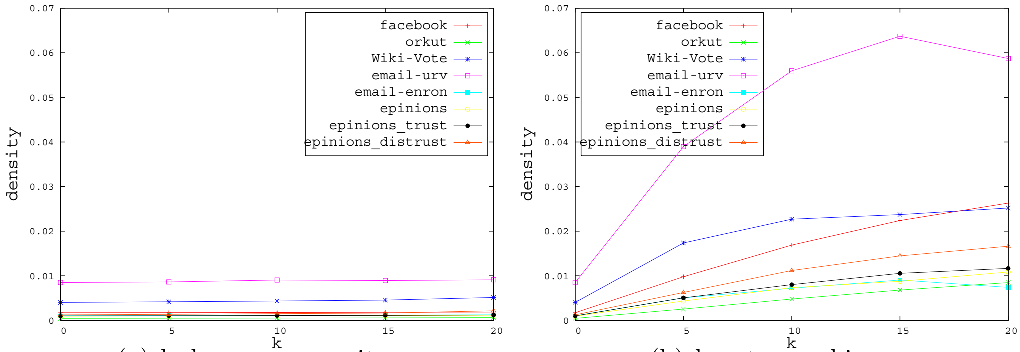
Table A.1: Description of data sets

	NO.of vertices	NO.of edges	type
Facebook	6,339	34,539	undirected
Wikipedia	7,066	100,736	directed
Orkut	9,217	19,550	undirected
Epinions	13,182	83,147	undirected
Epinions-trust	11,446	66,464	directed
Epinions-distrust	4,334	11,748	directed
Email-Enron	10,108	180,811	undirected
Email-URV	1,133	5530	undirected

the network of e-mail interchanges among faculty and graduate students at Rovira i Virgili University of Tarragona, Spain. It was collected by Guimer et al. [1]. We obtained it from Alex Arenas Website [1]. The graph is undirected. It has 1,133 vertices and 10,902 edges. Each vertex represents an email address. An edge exists between two vertices if there is an email communication between them. As with the Orkut data set, the graph is one connected component.

A.5 Experiments

The experiments are conducted on an Intel Core, 2 Quad CPU, 2.83GHz machine with 4GB main memory running Windows 7 OS. The programs for the metrics are implemented in two programming languages, C and Python. We calculate betweenness centrality and algebraic connectivity in python, utilizing functions in *Networkx* package [113]. All the other metrics are calculated in C, some of which, such as eigenvector and closeness centrality, with the help of the functions provided by the snap network analysis library [136]. For the ten sample graphs we randomly extract from each original network (Facebook, Epinions, Orkut, Email-Enron), as we mentioned before, we evaluate the metrics on all graphs and get the average value so that the results are less affected by the randomness of sampling. We run



(a) k-degree anonymity (b) k-automorphism

Figure A.3: Denstiy

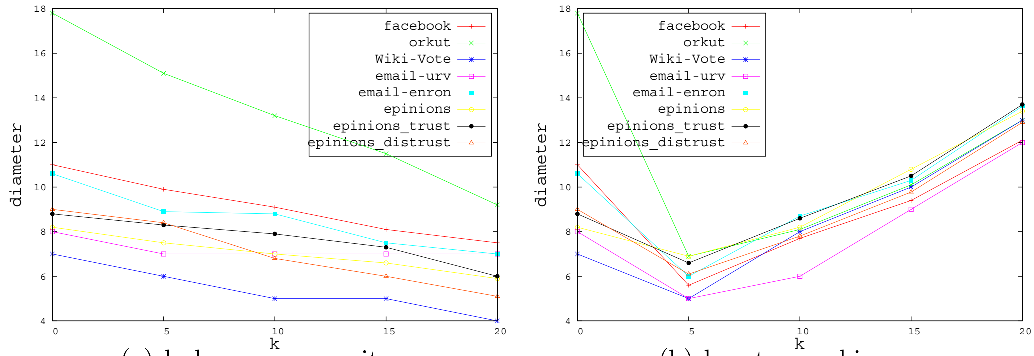
the anonymization algorithm on all the samples, then conduct the measurements and take the average values. We treat all the graphs as undirected graphs for the adoption of the k-automorphism algorithm.

Figure A.3 shows the density of each graph before and after anonymization for varying values of k. The x-axis shows the value of k, and the y-axis shows the density. When k equals 0, y-axis shows the densities of the original graphs.

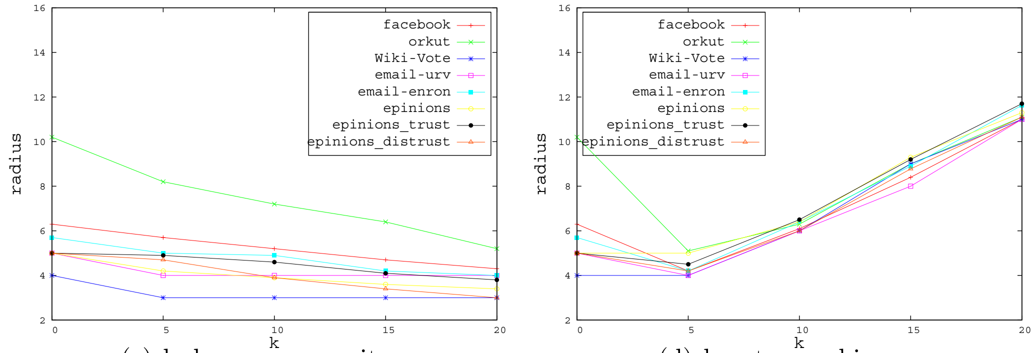
In view of different data sets, the email-urv graphs show especially high densities. In view of the same data set, modified graphs have greater densities than the original graphs, and a larger k does not always correspond to a larger density. In view of different algorithms, the k-degree anonymity algorithm increases density much less than the k-automorphism algorithm does.

Figure A.4 shows the diameter and radius of each graph before and after anonymization for varying values of k. The x-axis shows the value of k, and the y-axis shows the diameter and radius. Figure A.4(a)(c) show the diameter and radius of the original graphs and the k-degree anonymous graphs. Figure A.4(b)(d) show the diameter and radius of the original graphs and k-automorphic graphs.

We can see that modified graphs corresponding to relatively smaller values of k have both smaller diameter and radius than the original graphs do. As k increases, the differences among the diameters of the modified graphs decrease, similar for



(a) k-degree anonymity (b) k-automorphism

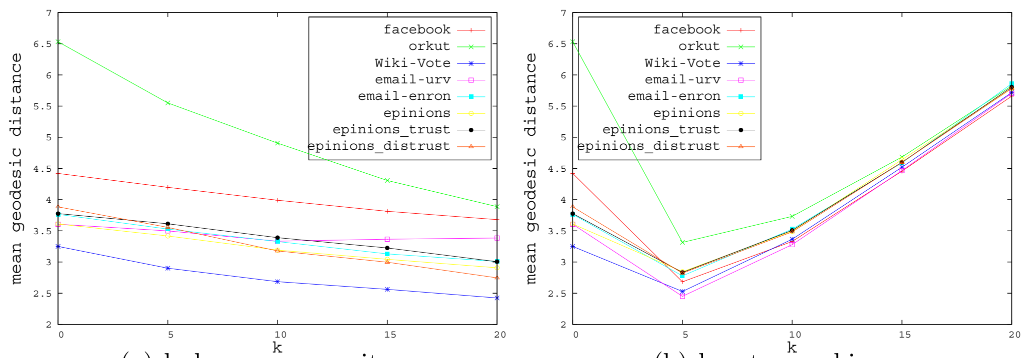


(c) k-degree anonymity (d) k-automorphism

Figure A.4: Graph diameter and radius

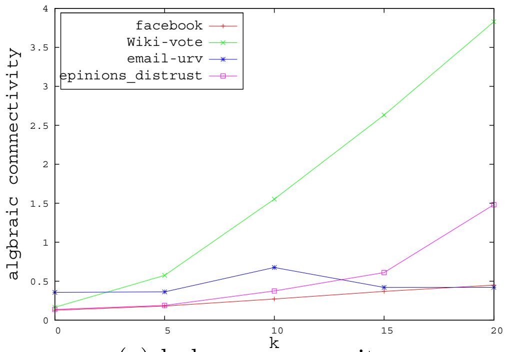
radius. In the figure, all the modified graphs have diameter around 6 when k equals 5, while large differences exist among the diameters of the original graphs.

Figure A.5 shows the mean geodesic distance of each graph before and after anonymization for varying values of k. The x-axis shows the value of k. The y-axis shows the mean geodesic distance. Subfigure(a) shows the values of the original graphs and the k-degree anonymous graphs. Subfigure(b) shows the values of the

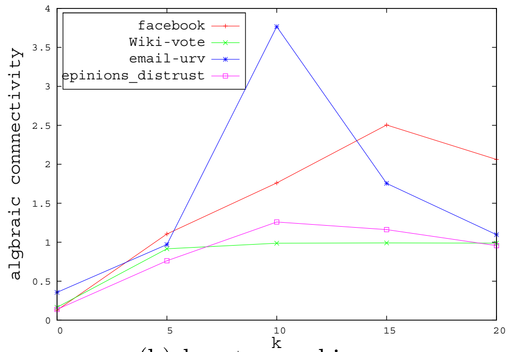


(a) k-degree anonymity (b) k-automorphism

Figure A.5: Mean geodesic distance



(a) k-degree anonymity



(b) k-automorphism

Figure A.6: Algebraic connectivity

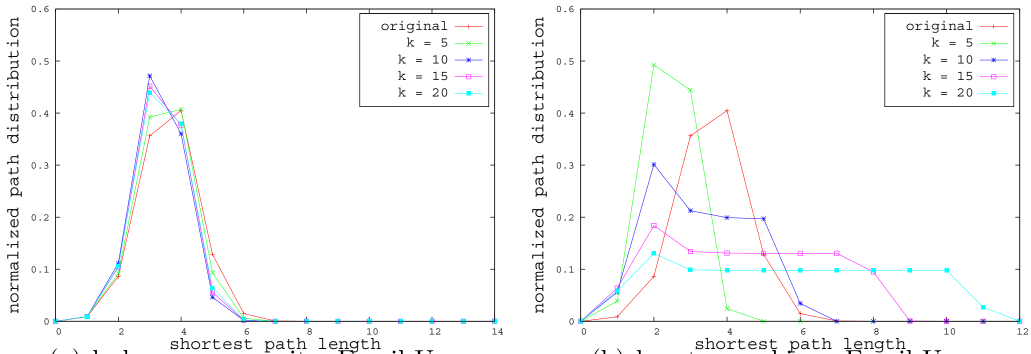
original graph and the k-automorphic graphs.

We can see that as k increases, the differences among mean geodesic distances of the modified graphs for all the data sets decrease. This figure also shows that the mean geodesic distance is smaller for a smaller value of k, similar to diameter and radius.

Figure A.6 shows the algebraic connectivity of each graph before and after anonymization for varying values of k. The x-axis shows the value of k, and the y-axis shows the algebraic connectivity. Subfigure(a) shows the values of the original graphs and the k-degree anonymous ones. Subfigure(b) shows the values of the original graph and the k-automorphic ones.

Due to the limitation of machine memory size, we are unable to calculate algebraic connectivity for all the data sets. Since the calculation of algebraic connectivity is based on the adjacency matrix, for very large graphs, the memory is not large enough to afford having the whole graph represented in an adjacency matrix. Nevertheless, from the available result, we see that as k increases, algebraic connectivity increases in general.

Figure A.7 shows the geodesic distributions for the Email-Urv graph, both before and after anonymization. The x-axis shows the value of geodesic distance, and the y-axis shows the ratio of the number of certain geodesic distance to the number



(a) k-degree anonymity. Email-Urv (b) k-automorphism. Email-Urv

Figure A.7: Geodesic distribution

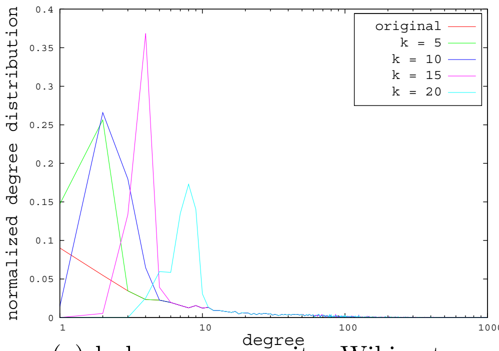
of all geodesic distances. Subfigure(a) shows the distributions of the original graphs and the k-degree anonymous graphs. Subfigure(b) shows the distributions of the original graph and the k-automorphic graphs.

From Figure A.7(b) we can see that as k increases after a certain value, the distributions of the geodesic distance become uniform in larger ranges, while the changes of the distribution made by the k-degree anonymity algorithm is much less, as we can see from Figure A.7(a).

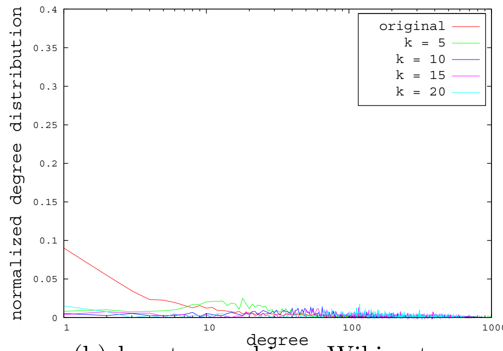
Figure A.8 shows the degree distributions for the Email-Urv and Wiki graph both before and after anonymization. The x-axis shows the vertex degree, and the y-axis shows the ratio of the number of vertices with a certain degree to the total number of vertices of the graph. Subfigure(a) shows the distributions of the original graphs and the k-degree anonymous graphs. Subfigure(b) shows the distributions of the original graph and the k-automorphic graphs.

We can see that degree distribution of the anonymized graph significantly differs from the degree distribution of the original graph.

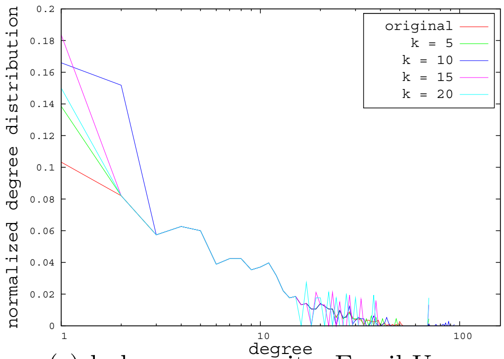
Figure A.9 shows the global clustering coefficient and average local clustering coefficient over all the vertices of each graph, before and after anonymization, for varying values of k. The x-axis shows the value of k, and the y-axis shows the clustering coefficient. Figure A.4(a)(c) show the metrics values of the original



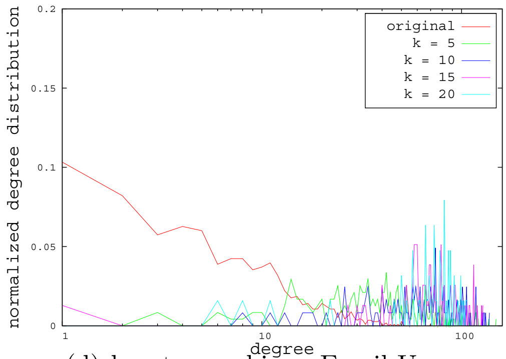
(a) k-degree anonymity. Wiki-vote



(b) k-automorphism. Wiki-vote

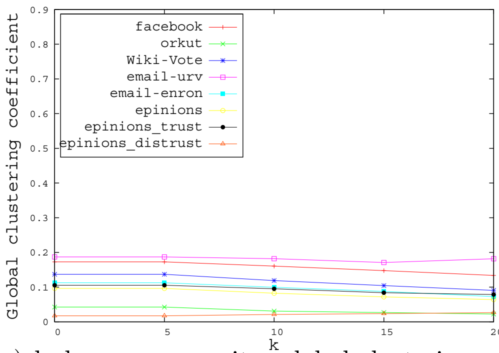


(c) k-degree anonymity. Email-Urv

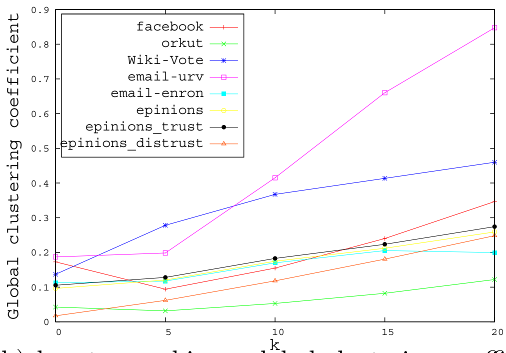


(d) k-automorphism. Email-Urv

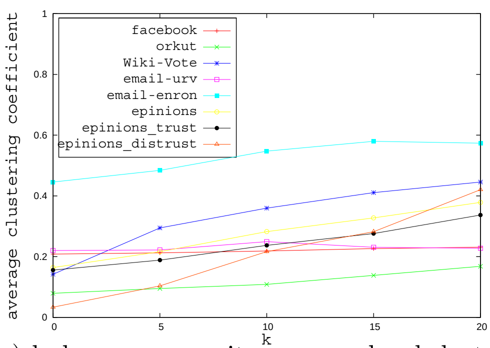
Figure A.8: Degree distribution



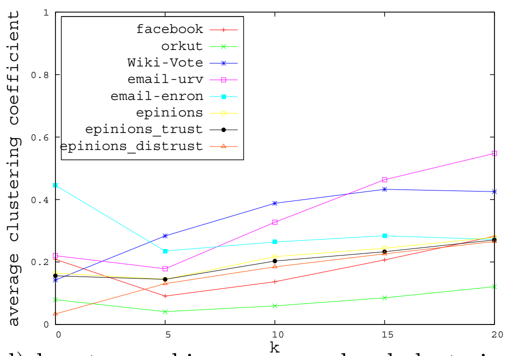
(a) k-degree anonymity. global clustering coefficient



(b) k-automorphism. global clustering coefficient



(c) k-degree anonymity. average local clustering coefficient



(d) k-automorphism. average local clustering coefficient

Figure A.9: Clustering Coefficient

graphs and the k-degree anonymous graphs. Figure A.4(b)(d) show the metrics values of the original graphs and k-automorphic graphs.

We can see that the modifications caused by k-automorphism algorithm is significant in some cases.

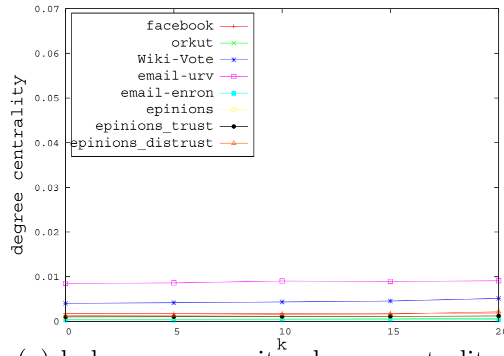
Figure A.10 shows the four centrality measures of each graph, before and after anonymization, for varying values of k. The x-axis shows the value of k, and the y-axis shows the centralities. We can see that as k increases, the degree centrality of modified graph increases significantly (Figure A.10a). Closeness centrality (Figure A.10d) and betweenness centrality (Figure A.10f) have a similar trend as the diameter, radius and mean geodesic distance changed during the anonymization.

For each vertex, we consider the corresponding value in the eigenvector of the adjacency matrix of the graph for the original graph and, then, for the modified graph. Figure A.11 shows the number of vertices that were in the top 10% with the highest value in the original graph and that remain in the top 10% in the modified graph. The x-axis shows the value of k, and the y-axis shows the ratio of the number of remaining vertices to the total number of vertices of each graph.

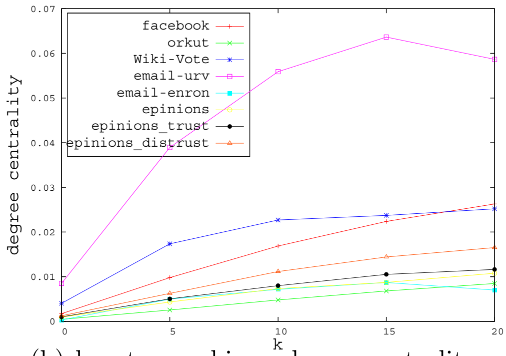
For all the k-automorphic graphs, at least 30% of the top 10% important vertices fall out of the range, while for some cases of the k-degree anonymous graphs, more than 30% of the vertices also fall out of the range.

Figure A.12 shows the *EMD* between the degree distributions of each graph, before and after anonymization, for varying values of k. The x-axis shows the value of k, and the y-axis shows *EMD*. We can see that as k increases, *EMD* increases in general. This suggests that the difference of degree distributions between graphs before and after anonymization increases.

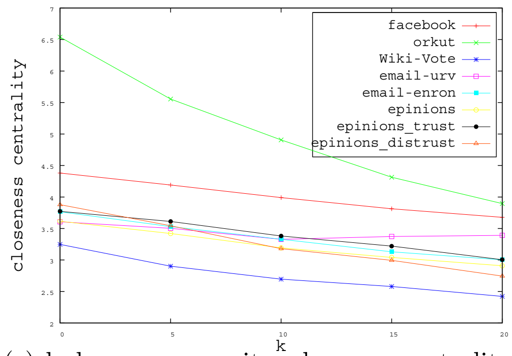
Figure A.13 shows the edit distance for each data set, and for varying values of k. The x-axis shows the value of k. The y-axis shows the edit distance. We can



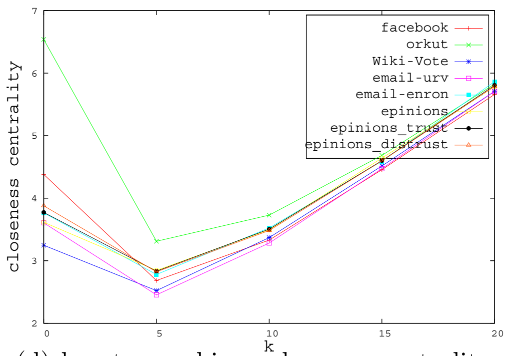
(a) k-degree anonymity. degree centrality



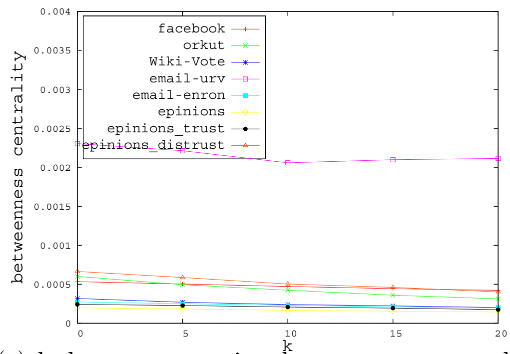
(b) k-automorphism. degree centrality



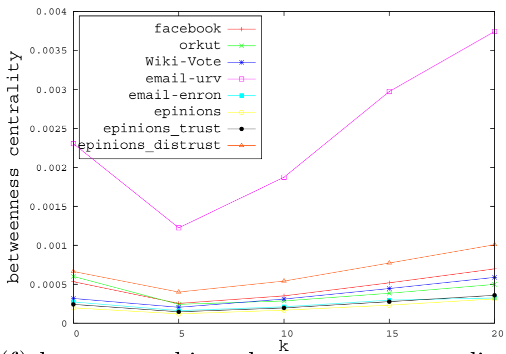
(c) k-degree anonymity. closeness centrality



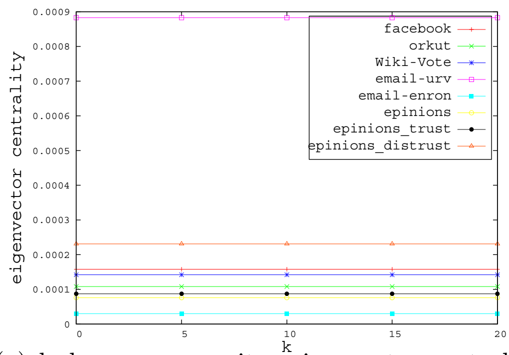
(d) k-automorphism. closeness centrality



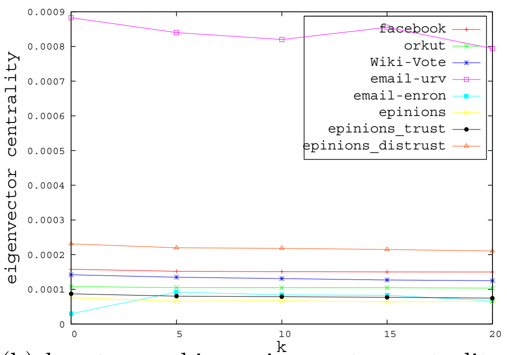
(e) k-degree anonymity. betweenness centrality



(f) k-automorphism. betweenness centrality

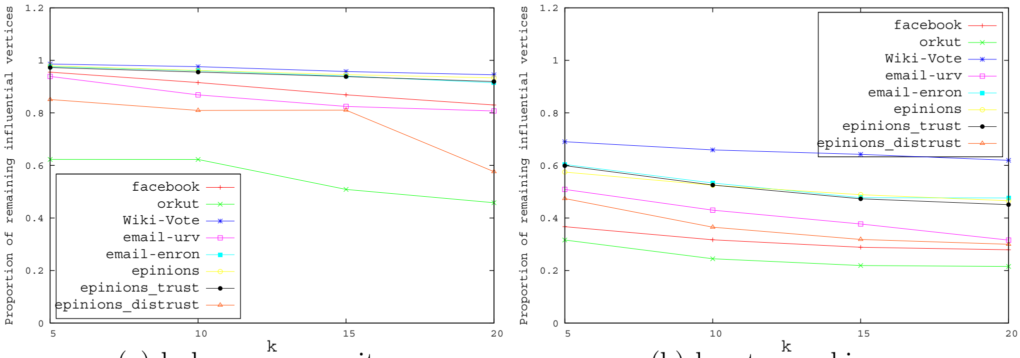


(g) k-degree anonymity. eigenvector centrality



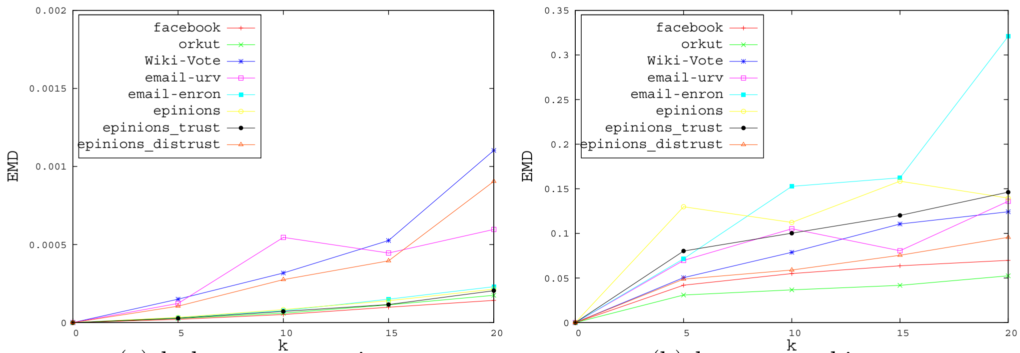
(h) k-automorphism. eigenvector centrality

Figure A.10: Centrality metrics



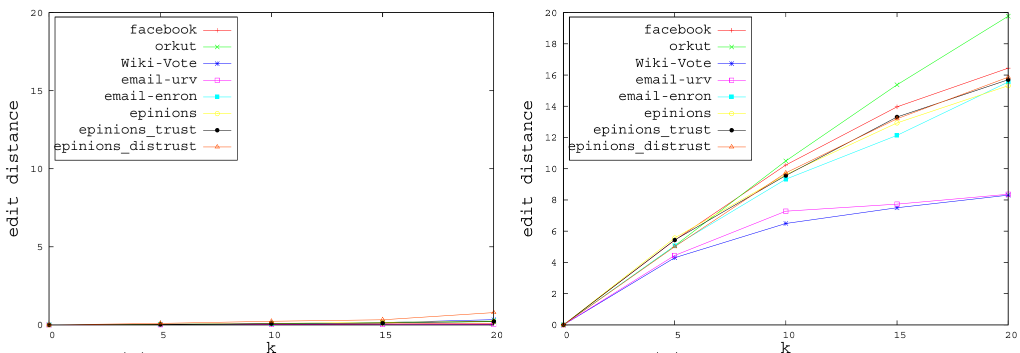
(a) k-degree anonymity (b) k-automorphism

Figure A.11: Remaining proportion of influential vertices



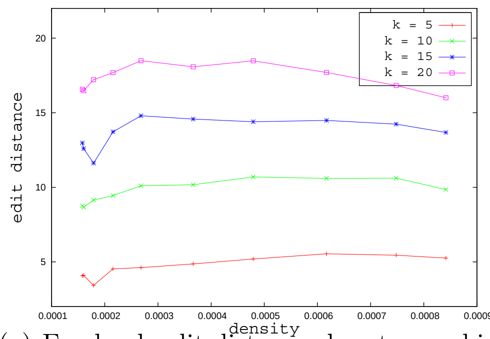
(a) k-degree anonymity (b) k-automorphism

Figure A.12: Earth mover's distance

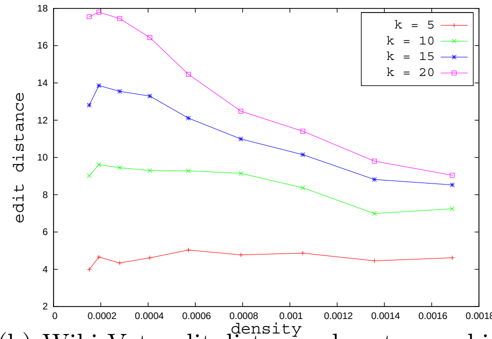


(a) k-degree anonymity (b) k-automorphism

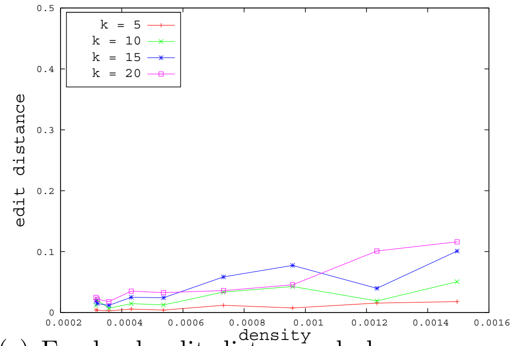
Figure A.13: Edit distance



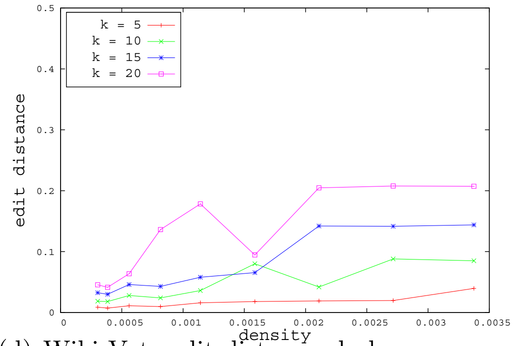
(a) Facebook edit distance. k-automorphism



(b) Wiki-Vote edit distance. k-automorphism



(c) Facebook edit distance. k-degree anonymity



(d) Wiki-Vote edit distance. k-degree anonymity

Figure A.14: Edit distance vs density

see that as k increases, the edit distance increases.

To verify the effect of graph density, we sample the graphs with different densities from the Facebook and Wiki-vote data set. We anonymize each sample and measure the edit distance. Figure A.14 shows the edit distances for graphs with various densities. The x-axis shows the density. The y-axis shows the edit distance. We can see that, as k increases, the edit distance increases in most cases.

This comprehensive set of experiments on graphs from real social networks demonstrate that utility metrics are impacted by k-degree anonymity and k-automorphism anonymization. Especially for k-automorphism anonymization, we can see significant impact. With increasing values of k, the density of anonymized graph increases in general, and is up to six times that of the original graph in the worst case, for all the data sets considered. Diameter, radius, and mean geodesic distance can be two to three times smaller or larger in the worst cases, though they remain unchanged in some cases. Degree centrality continuously increases. Clustering coefficient, closeness and betweenness centrality remain the same in the best case, but the clustering coefficient increases significantly in some cases. The number of top 10% influential vertices remaining, evaluated based on eigenvector centrality, decreases. The highest proportion is about 70%, while the lowest approaches 20%.

Privacy is guaranteed by the design of k-degree anonymity for degree attacks and k-automorphism for most structural attacks. However, this is achieved at a high cost. Significant modifications are induced by the graph perturbation.

A.6 Summary

We empirically quantify the trade-off between utility and privacy for the k-degree anonymity and k-automorphism graph anonymization algorithm. We measure and compare several utility metrics for a series of real graphs from various social media before and after their anonymization under varying settings.

The study shows that anonymization is not anodyne. It protects privacy by significantly modifying a graph before its publication at the expense of utility. Furthermore, although the general trend of the effect on some metrics, for instance those measuring graph density, can be foreseen, some effects seem to remain unpredictable.

Appendix B

GPU-based Parallel Particle Swarm Optimization Methods for Graph Drawing

Effective graph drawing is needed for presentation and qualitative analysis. It is generally agreed that an effective graph drawing should have the following aesthetic characteristics [10]:

1. Minimal edge crossing,
2. Vertices are evenly distributed in the space,
3. Connected vertices are close to each other, and
4. Symmetry may exist in the graph.

In 1963, Tutte [147] proposed an algorithm to draw planar graphs by fixing selected nodes on a face and placing the rest of the nodes at the barycenters of their neighbors. In 1981, Sugiyama et al. [139] proposed a method for the hierarchical drawing of directed graphs. Eades [52] proposed a force-directed algorithm in 1984.

Kamada and Kawai [85] proposed the spring embedding algorithm in 1989. In 1991, Fruchterman and Reingold improved the force-directed method [64]. These algorithms were further improved, and some of them are applied to large graphs [66, 22, 23, 80, 83, 151, 134] later.

Within the graph drawing literature, the name "force-directed algorithm" has often been used to refer to spring-electrical models [52, 85, 64]. Graphs drawn by these algorithms are aesthetically pleasing, symmetric, and tend to have minimal cross edges. In general, force-directed methods define an objective function which maps each graph layout into a real number representing the energy of the layout. This function is defined in such a way that low energies correspond to layouts in which adjacent nodes are near some pre-specified distance from each other, and in which non-adjacent nodes are well-spaced [52]. A layout for a graph is then calculated by finding a (often local) minimum of the objective function.

We introduce the *Particle Swarm Optimization* algorithm to solve the graph drawing problem. Inspired by the paradigm of birds flocking, Kennedy and Eberhart proposed Particle Swarm Optimization (PSO) in [53]. It is a global optimization method, where the system is initialized with a swarm of random particles and the algorithm searches for optima by updating generations. the PSO algorithm is widely used in many combinatorial optimization problems [163] for its simple implementation and fast speed. The layout of a graph drawn by the force-directed algorithm is obtained by finding an optimal solution of the objective function. That is to say, the layout problem can be converted to an optimization problem.

We propose a new method called *PSOGD* for drawing undirected graphs using PSO and a force-directed algorithm [85]. The graph is initialized with a swarm of random particles which store the position information of all vertices in the graph. All particles automatically update their position and velocity in order to find the optimal layout until the algorithm terminates.

Particle Swarm Optimization is a global optimization method based on swarm, where the system is initialized with a swarm of random particles and the algorithm searches for optima by updating generations. Suppose that the dimension of search space is D, and the number of the swarm size is N. The position vector and velocity vector of particle i can be represented as follows:

$$X_i = (x_{i1}, x_{i2}, ..., x_{ij}, ..., x_{iD}) \tag{B.1}$$

$$V_i = (v_{i1}, v_{i2}, ..., v_{ij}, ..., v_{iD}) \tag{B.2}$$

The memory position vector of the particle i previously visited and the global memory position vector of the swarm found so far are denoted P_i and P_g:

$$P_i = (p_{i1}, p_{i2}, ..., p_{iD}) \tag{B.3}$$

$$P_g = (g_1, g_2, ..., g_D) \tag{B.4}$$

The fitness $f(P_i)$ of each particle can be evaluated by putting its position into a designated objective function. The particle's velocity and its new position are updated as formula B.5 and B.6.

$$v_{ij}^{t+1} = \omega \cdot v_{ij}^t + c_1 \cdot r_1 \cdot (p_{ij}^t - x_{ij}^t) + c_2 \cdot r_2 \cdot (g_{ij}^t - x_{ij}^t) \tag{B.5}$$

$$x_{ij}^{t+1} = x_{ij}^t + v_{ij}^{t+1} \tag{B.6}$$

Here, $j \in \{1, 2, ..., D\}$, $i \in 1, 2, ..., N$. The superscript t denotes the iteration number; ω is the initial weight; r_1 and r_2 are two random values in the range of 0 and 1; c_1 and c_2 are the cognitive and social scaling parameters which are positive constants. The steps of PSO algorithm is described as following.

The steps of PSO algorithm is described as following.

Step1: Initialization of PSO.

Step2: Update V and X by formula B.5 and B.6.

Step3: Calculate fitness of each particle.

Step4: Update P_i and P_g.

Step5: If terminal condition is met, end the algorithm; else go to step2.

There are two key techniques in the algorithm. One is the encoding schema of the swarm. The other is the definition of the fitness function. In a simple graph $G = (V, E)$, x_i denotes the current position of vertex i in d-dimension Euclidean space. Generally the value of d is 2 or 3. The objective of graph drawing is to find the positions of all vertices so that the drawing can give a good layout of the graph

PSO algorithm has powerful global search capability. The first step of PSO is to determine the encoding schema of the swarm. An efficient encoding schema enables the problem simple and intuitive. Considering of the objective of graph drawing, each particle of the swarm corresponds to the position information of all vertices. Each particle is encoded as a n-dimensional vector X as following.

$$x = (x_1, x_2, ..., x_i, ..., x_n) \tag{B.7}$$

Here, $n = |V|$ and x_i denotes the position of vertex i in d-dimension Euclidean space.

A particle with such an encoding schema corresponds to a kind of layout of graph drawing. It is evident that finding the best particle of the swarm is to find the optimal layout of graph drawing. Each particle of the swarm updates the velocity and position vector in the process of evolution by formula B.5 and B.6.

The selection of fitness function is critical to the PSO algorithm. An effective fitness function can make the particles of the swarm find the optimal solution. In the PSOGD algorithm, a particle corresponds to a kind of layout of graph drawing. The objective of graph drawing is finding a good visual representation of the connectivity information between vertices. There are different styles of representation, suitable to different types of graphs or different purposes of presentation. However, no uniform criteria can be used to evaluate the performance of different representation. Here, inspired by the spring-electrical model, we use the idea of the force-directed algorithm in [87, 118] to define the fitness of the swarm. The attractive force is defined as formula B.8.

$$f_a(i,j) = \frac{||x_i - x_j||^3}{3k} \tag{B.8}$$

The repulsive force is defined as formula B.9.

$$f_r(i,j) = k^2 \cdot ln(||x_i - x_j||) \tag{B.9}$$

Here, $k = C\sqrt{\frac{area}{number\_of\_vertices}}$. $area$ is the windows size for display the graph. C is a constant.

The objective function of the particle is defined as formula B.10.

$$f(X) = \sum_{(i,j)\in E} f_a(i,j) + \sum_{(i,j)\in E} f_r(i,j) \tag{B.10}$$

The fitness function maps a position vector into a real number representing the energy of the layout. It means that low energies correspond to layouts in which adjacent nodes are near some pre-specified distance from each other, and in which non-adjacent nodes are well-spaced. That is to say, the particle with the minimal fitness value is the global optimal particle. The layout of a graph is obtained by searching for the minimum of the fitness function. The objective of the PSOGD algorithm is to search for the fitness that is as small as possible.

The pseudo code of the algorithm is described in Algorithm 14

Algorithm B.1: Graph Drawing by Particle Swarm Optimization

Input: G(V,E), the number of iteration T;
Result: the layout of the graph

set parameter(swarm size m, initial weight ω);
particle by the formula B.8-B.10;
select P_g;
for t = *1 to* T **do**
    for i = *1 to* m **do**
        update V and X by formula B.5, B.6;
        for *j=1 to —V—* **do**
            for *k=1 to —V—* **do**
                calculate $f_a(j,k)$ by formula B.8;
                if $(j,k) \in E$ **then**
                    calculate $f_r(j,k)$ by formula B.9;
            calculate $f(P_i^t)$ by formula B.10;
            if $f(P_i^t)$ ¡ $f(P_g)$ **then**
                P_g=P_i^t;
Return optimal layout;

In the experiments, we compare PSOGD algorithm with F-R [64] algorithm.

Table B.1: Parameters of the algorithm

ω	m	c_1	c_2	t	W	L	C
0.72	10	2.02	2.02	200	1.0	1.0	0.75

We choose 10 artificial graphs and 3 real network graphs, and divide them into four groups. Group 1 consists of 4 general graphs: g1, g2, g3 and g4. Group 2 consists of 5 symmetric graphs: g5, g6, g7, g8 and g9. Group 3 only consists of one graph g10. Group 4 consists of 3 real network graphs. Different groups of graphs are intended to test the performance of the algorithms from different aspects. The

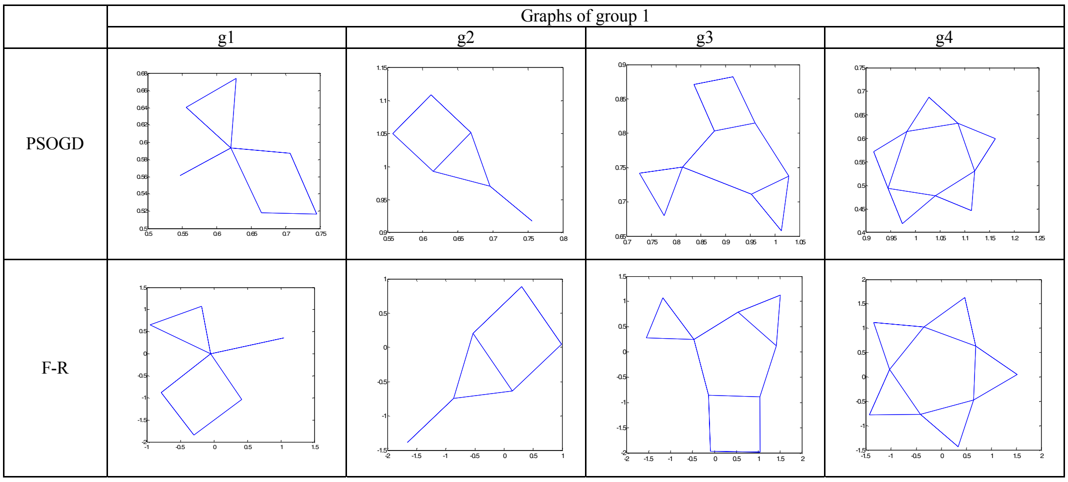


Figure B.1: Drawing of group 1 by PSOGD and F-R

parameters of the algorithm are set in Table B.1. Here, m is the number of particles in the swarm, and t is the number of iterations. The meaning of other parameters is described in the paper. The information of group 4 is listed in Table B.2.

Table B.2: Information of Group 4

	Karate club (g11)	Dolphin network (g12)	American football (g13)
number of vertices	34	62	115
number of edges	78	105	613

Figure B.1 shows the results of the PSOGD and F-R algorithms on group 1. The layouts of these graphs drawn by the two algorithms are similar. Figure B.2 is the drawings of PSOGD and F-R on group 2. It can be seen that the drawings by two algorithms all look nice, symmetric, and have fewer crossing edges except for graph g6. The drawing by PSOGD is better than the one by F-R, which has a crossing edge. The results in Figure B.1 and B.2 illustrate the effectiveness of PSOGD algorithm.

Figure B.3 shows the results of the PSOGD and F-R algorithms on group 3. Five drawings all represent the relationship of vertices and edges in graph g10, but the

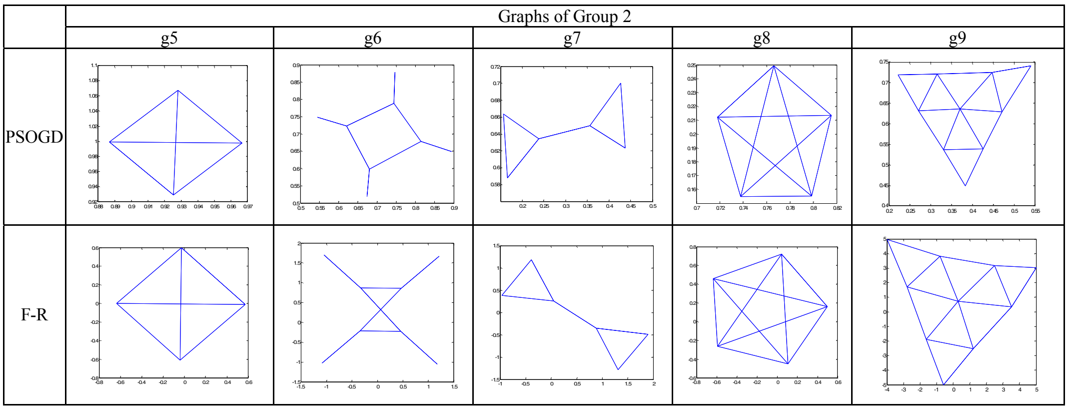


Figure B.2: Drawing of group 2 by PSOGD and F-R

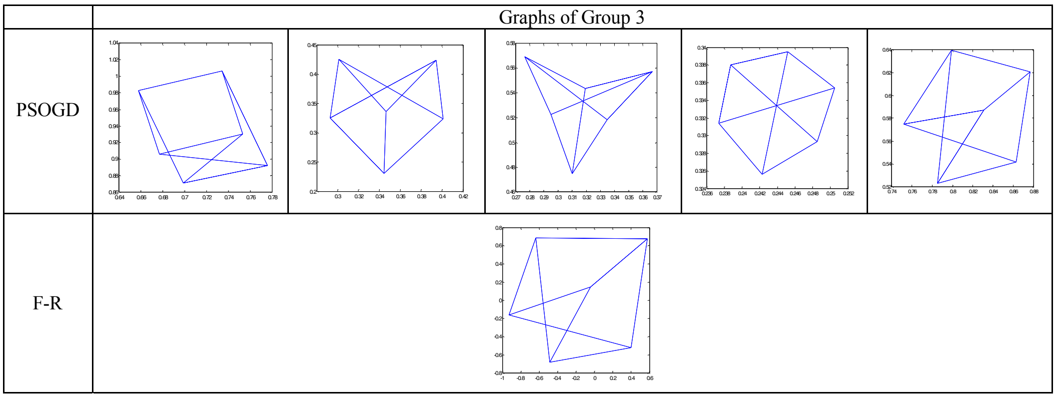


Figure B.3: Drawing of group 3 by PSOGD and F-R

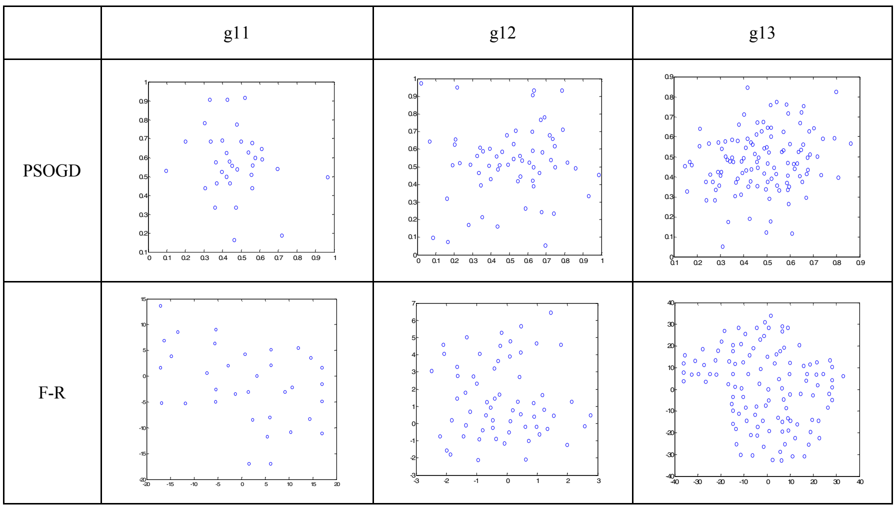


Figure B.4: Drawing of group 4 by PSOGD and F-R

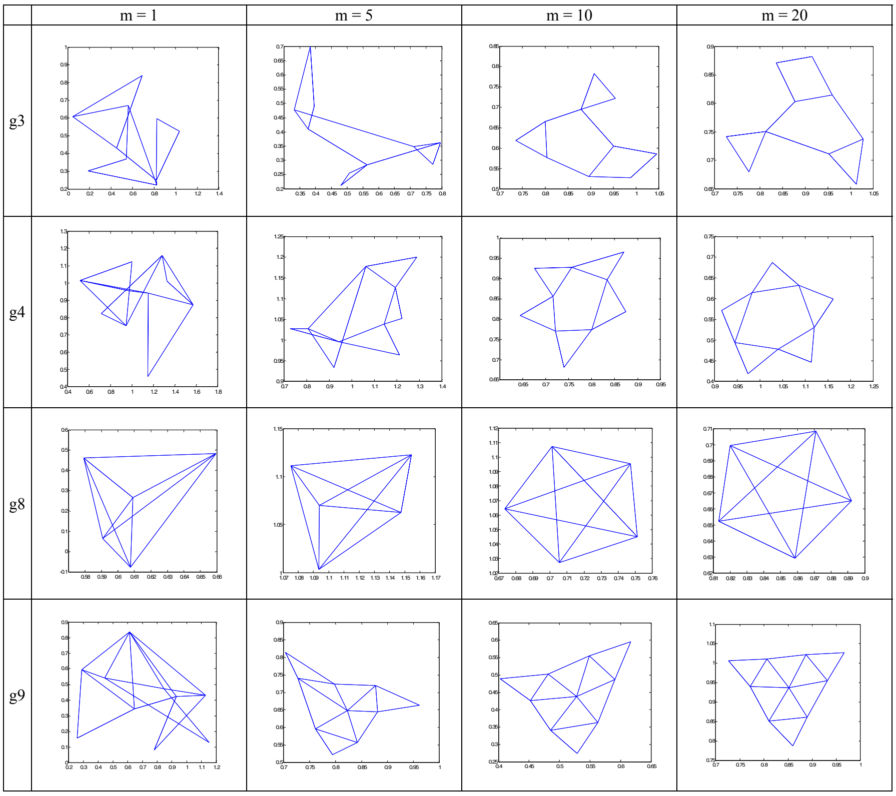
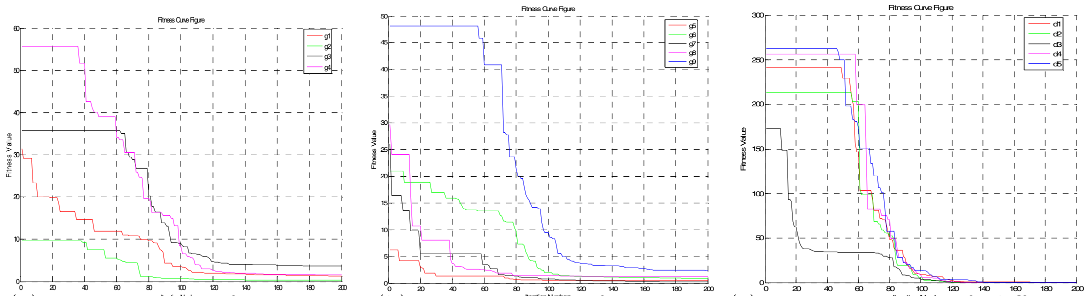


Figure B.5: Evolutionary drawing with varying number of particles by PSOGD

PSOGD algorithm gets 5 different kinds of layouts. Such results are caused by the randomness of the PSOGD algorithm. PSO is a stochastic optimization method, and different initial solutions will get different results. The different drawings show the diversity of the layouts obtained by PSOGD algorithm.

Figure B.4 is the results of of the PSOGD algorithm and F-R algorithm on group 4. To make the position of vertices clear, we only give the corresponding drawings with vertices. The vertices in the drawings by both algorithms look distributed evenly.

We draw the evolutionary drawings with a varying number of particles on graph g3, g4, g8 and g9 by PSOGD algorithm. Figure B.5 shows that when m is 1, the drawings of four graphs are not good. They show better layouts with the increasing number of particles. When m is 10, four graphs all appear the nice layouts. When



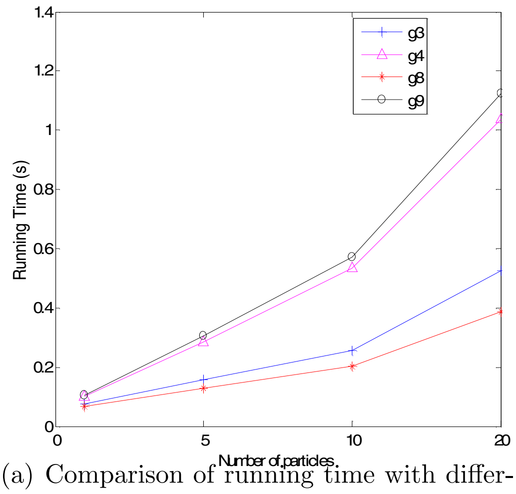
(a) Fitness curve of group 1 (b) Fitness curve of group 2 (c) Fitness curve of different layouts of group 3

Figure B.6: Fitness curve

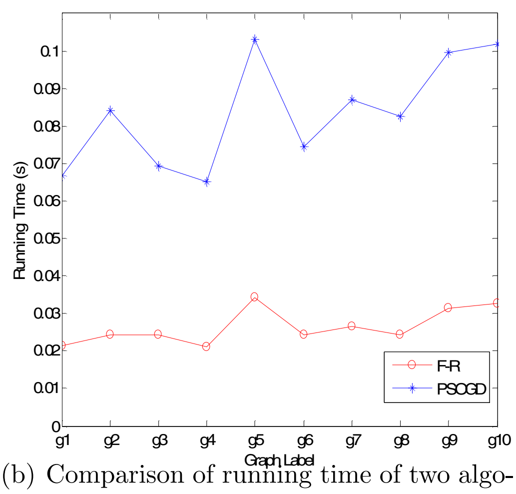
an increases to 20, they all get the best layouts.

Figure B.6 compares the change of the fitness value during the optimization process of the PSOGD algorithm on the 3 groups of graphs. Figure B.6(a) is the fitness curve of group 1. Figure B.6(b) shows the fitness curve of group 2. Figure B.6(c) shows the fitness curve of five different layouts of group 3. From these fitness curves in Figure B.6, it can be seen that the PSOGD algorithm can converge after a number of iterations. We do not give the number of iterations. It depends on many elements, such as size of the graph, the swarm size, etc. The number of the iterations in Figure B.6 is only an experimental result. In the F-R algorithm, there is no detailed explanation on the termination conditions.

Based on the results in Figure B.7(a), we also compared the running time of the PSOGD algorithm on graphs g3, g4, g8 and g9 with the varying number of particles from 1 to 20. Figure B.7(b) shows the result. It is obvious that the running time gets longer with the increasing number of particles. Such a result can be analyzed from the time complexity of the PSOGD algorithms. In each iteration, the time complexity of the PSOGD algorithm is $\mathcal{O}(m|V|^2)$. That is, the time complexity is proportional to the number of particles. Therefore, we get the result in Figure B.7. It can be seen from the evolutionary drawings of graph g3 and g8 in Figure B.5 that there are no larger improvements on the drawings with the increasing m from



(a) Comparison of running time with different number of particles



(b) Comparison of running time of two algorithm on 10 graphs

Figure B.7: Comparison of running time

10 to 20, but the running time has increased a lot. Therefore, simply increasing the number of particles is cannot improve the effectiveness of the algorithm for some graphs.

Figure B.7(b) compares the running time of PSOGD and F-R algorithms on 10 different graphs. The number of the iterations is 200 and the number of particles is 20. In each iteration, the time complexity of F-R algorithm is $\mathcal{O}(|V|^2 + |E|)$. It can be concluded that the time complexity of PSOGD is higher than that of F-R algorithm. The result in Figure B.7(b) confirms this conclusion. Nevertheless, the layouts of some graphs obtained by PSOGD are better than those obtained by F-R.

To sum up, we propose PSOGD, a force-directed algorithm computing the equilibrium using particle swarm optimization. The algorithm is simple, as well as its implementation. We empirically and comparatively evaluated the performance of the PSOGD algorithm. The results confirmed the symmetry and diversity of the graph layout and showed that the graphs drawn by PSOGD algorithm are generally aesthetic.