The Open University of Israel
Department of Mathematics and Computer Science

Sharing-habits based privacy control in social networks

Thesis submitted as partial fulfillment of the requirements
towards an M.Sc. degree in Computer Science
The Open University of Israel
Computer Science Division

By
Silvie Levy

Prepared under the supervision of
Prof. Ehud Gudes, The Open University Of Israel and Ben Gurion University, Israel
Dr. Nurit Gal-Oz, Sapir College, Sederot, Israel

May 11, 2018

Acknowledgments

I would like to express my gratitude to my thesis advisers, Prof. Ehud Gudes and Dr. Nurit Gal-Oz, for their assistance and support. Ehud and Nurit provided me encouragement, guidance, practical advice and comments with amazing amount of vision, dedication and patience, throughout the whole process of writing this thesis. I learned a great deal from both of them and owe them a huge debt of gratitude. The theoretical problem was enriched by the help of Prof. Zeev Nutov, and his great amount of knowledge. Most of all I must thank my family. My loving husband, Itzik, for his encouragement, faith and support from the very beginning. My children Hila, Efrat, Assaf and Nadav, for their patience and love while I was working on this thesis.

Contents

List of Figures

List of Tables

Abstract

We study users behavior in online social networks (OSN) as a means to preserve privacy. People widely use OSN for a variety of objectives and fields, like updating their profiles and shared media, browsing the internet for social or professional interactions, or reacting to friends shared data. Each OSN has different characteristics, requirements, and vulnerabilities of the private data shared. *Sharing-habits* refers to users' patterns of sharing information. Sharing-habits are implied by the communication between users and their peers. While social networks allow users to have some control over the dissemination of their information, most users are not aware that the private information they share might leak to users with whom they do not wish to share it. Most access control models define access rules in terms of the degree of relationship required to access ones data. These rules are not refined enough to allow for dynamic denial of content from certain peers of the community. In this thesis we address the growing need of social network users to share information with close fiends while hiding it from others. We apply several different well-known strategies from graph-flow theory to an OSN graph with sharing-habits insights, to control the information flow among OSN users. The goal of the method we present is to allow maximum information sharing while enforcing a user's pre-defined privacy criteria. We analyze the user's community within a predefined distance, and enable the user to define the required privacy level for each shared information. The user can define with whom he would like to share the entire shared information, what would be the maximum fraction of data he is willing to share with undesired recipients, and what would be the minimum percentage amount he is willing to avoid from his community acquaintances, in order to achieve maximum privacy level. Our method is evaluated using partial real data from well known social networks and the results are analyzed in terms of correctness and run-time.

Chapter 1

Introduction

Online Social networks (OSN) are websites enabling users to build connections and relationships among each other. The OSN structure represents social relationships between its users. Social networks are widely used by their members for information sharing with the purpose of reaching as many friends and acquaintances as possible. Communication between users hides a lot of private information that can be deduced from the obvious information they share, like their work and home address, places they usually visit, and much more. Users should have control over the dissemination of their information, however they are not fully aware of the possible consequences of their preferences when specifying access rules to their shared data. Most access rules are defined in terms of the degree of relationship required to access ones data and are not refined enough to allow the dynamic denial of content from certain peers of their community, referred to as adversaries. It is the responsibility of OSN administrators to effectively enforce these rules to reduce the risk of information leakage.

For example, consider a chocolatier that owns a small Chocolate boutique shop. The chocolatier has established a customers club as a closed-group in his Facebook account. The chocolatier introduces new tastes, sales opportunities and coupons to his Facebook closed-group. He invents a new taste for the incoming Valentine's day, and makes a special offer to his customers club in advance. The chocolatier would like this information to be shared with as many people as possible, but to remain hidden from his competitors (adversaries). Currently the Chocolatier has limited control over the information shared on his Facebook group, each member of the group, can re-post the shared information, that might reach one of his competitors.

We propose a model for access control that works with minimal user intervention. The model is based on users' patterns of sharing information denoted as *Sharing-habits*. Naturally some users are more likely to share information with others. To minimize the risk of information leakage, the social network is an-

alyzed to determine based on these habits, the probability of information flow through network connections.

In a graph representation of the network, where nodes represent users and edges indicate relationship between users, the challenge is to select the set of edges that should be blocked to prevent leakage of the shared information to unwanted recipients (adversaries).

We review some methods for handling and preserving privacy in social networks, and present our new privacy preserving approach, based on sharing-habits data. Our model combines algorithms that use graph flow methods such as *max-flow-min-cut*, and *contract*. We analyze a user's community within a predefined distance, and enable the user to define the required privacy level for each shared information. The user can define a group of close friends, with whom he would like to share the entire shared information, and what would be the maximum percentage amount of data he is willing to share with undesired recipients (adversaries). The user can also define what would be the minimum percentage amount he is willing to avoid from his community acquaintances, in order to achieve maximum privacy level. Our algorithms define a candidate set of edges that should be blocked in order to reach the required privacy level. The proposed set of edges might not be optimal since by blocking these edges (connections), we might reduce the amount of shared data from the user to his community. We evaluate the privacy level, using the user's predefined threshold levels, and produce the optimal set of edges that should be blocked in order to reduce the amount of shared data leakage.

While the general problem is NP-complete, we show that our algorithm finds the optimal solution in many cases.

We based our study on the ideas described by Maheswaran and Ranjbar [1]. In this paper the authors propose to completely share information within the defined community, and block users that might leak information to adversaries. We relax the limitation defined in their study, and block only edges on the path to the adversaries, instead of blocking all the information from the source user to the users that might leak the information.

We tested our algorithms on a synthetic graph, and on partial data from real networks taken from the Stanford's SNAP database. Since the SNAP database is anonymized, we add random probabilities to the sharing connections (graph edges).

Experimental results show the effectiveness of these algorithms in controlling the flow of information to allow sharing with friends while hiding it from others. To the best of our knowledge, this work is the first to provide dynamic access control model which is based on users dynamic interactions and not on user profiles.

The rest of the thesis is structured as follows: in the next two chapters we

review the background and the related work, in chapter 4 we define the privacy assurance in OSN problem, and in chapter 4.4 we present our algorithms for dealing with this problem. We explain our evaluation method and experimental results in chapter 5 and conclude by summarizing our contribution and discussing directions for future work in chapter 6.
The main contributions of this thesis are:

1. Defining the problem and proposing a new model.
2. Developing algorithms to solve the problem.
3. Testing our algorithms on synthetic data.
4. Testing our algorithms on partial real data.
5. Analyzing the results.

A paper based on this thesis was presented in DBSEC16.
A revised paper based on this thesis was accepted for publication in the Journal of Computer Security (JCS).

Chapter 2

Background and problem domain

Online Social Networks (OSN), are online social services, or websites applications, used by individuals or organizations, to build and maintain the relationships (connections) among each other, while sharing information like pictures, videos, places, and stories, and searching for employment opportunities, business or social events, and more. Many people beside the user's friends and acquaintances are interested in the information posted and maintained by the user on the social network. Unauthorized people are using social networks to gather information that they can use for their needs, either legal usage like business opportunities or advertisement, or illegal usage like identity theft or tracking people. The companies that operate the social networks are also collecting variety of data about their users, both to sell to relevant advertisers and to personalize the services for the users.

Van Dijck [9] reviews the history of social media, providing a critical history of roughly the first decade of connective media, by analyzing five specific platforms to the larger ecosystem and the culture in which it evolved. The transformation from networked communication to "platformed" sociability, and from a participatory culture to a culture of connectivity, changes users' behavior regarding social networks and online services. In December 2011, 82 percent of the world's Internet population over age 15, which is 1.2 billion users, were logged on to a social media site; while in 2007 only 6 percent were logged on to a social media site. With the availability of online services, and not only utilities, more and more people are using the Internet, and social media on a regular basis. Users are now doing more of their everyday activities using online environments, it became a part of their daily routines. Many of the habits that used to be informal and short-lived phenomenon of social life, have recently become infiltrated by social media platforms. People use online services for almost everyday routine; for example, looking up the meaning of a word, checking for movies, restaurants, parking etc. Activities like: talking to friends while exchanging gossip,

and checking on his well-being, sharing holiday pictures, scribbling notes, were commonly shared only with selected friends. Now, through social media, these acts are embedded in the online services, enabling the users to share data with wider friends and acquaintances. Statements that previously expressed unthinkingly, are now released into a public domain where they can reach a great mass of people, having a long-lasting effects. Social media platforms have altered the nature of private and public communication, what was previously held as private, is no more private, when a user uploads data into a social network it is not private as he thinks. The uploaded data is used to track the users' desire. Facebook and other platforms track what people might want by coding relationships between people, things, and ideas into algorithms, this enabling users' daily activities, and habits throughout social networks. For many users social media platforms are used not only for pleasure but also for profitable business. In less than a decade, the norms for online sociability have dramatically changed, and they continue changing. Patterns of behavior that traditionally existed in offline (physical) sociability are increasingly mixed with social and sociotechnical norms created in an online environment. For example, the norms for "sharing" private information and for accepting personalized advertisements in users' social space were very different in 2004, than in 2012. Changes and new features are constantly, and gradually implemented, users are getting habituated to those changes along with the adaptations of norms for privacy. These normalization occurs through various levels of adjustments, including technology features and terms of use.

There are different types of OSN [22], each has different characteristics, different properties, and different vulnerabilities. Some networks belong to more then one type. OSN types:

1. **Personal networks** are social interaction-centered sites that allow users to create detailed online profiles, connect with other users, and share information with friends. Personal networks emphasize on social relationships such as friendship. An example of a personal network is Facebook.
2. **Status-update networks** are information distribution-centric services, that allow users to quickly and publicly, post short status updates in order to communicate with others quickly. The user can restrict some users from accessing some status updates by using privacy settings. An example of a status-update network is Twitter.
3. **Shared-interest networks** are networks for common interest like hobbies, educational backgrounds, etc. These networks are directed toward specific subset of individuals and contain features from other types of social networks. An example of a shared-interest network is LinkedIn.
4. **Content-sharing networks** are designed as platforms for sharing content, such as videos, music, and photographs, and include interaction features that enable the content-sharing. When these websites include the ability

to create personal profiles, establish contacts and interact with other users through comments, they also become social networks. Examples of content sharing networks are YouTube, Instagram, and Flicker.

5. **Location networks** use the GPS on cellular phones or tablets to broadcast the user's real-time location. Most of these networks are built to interact with other social networks, such that an update made to a location network could post to the user's other social networks. An example of a location network is Google Latitude, and check-in option by Facebook.

People and companies may gather useful information from OSN through data mining. Companies can use the gathered data to improve their sales and profitability, by customizing sale to customers profiles. Facebook uses "Social Ads" program that gives companies access to the millions of profiles in order to tailor their ads to a Facebook user's own interests and hobbies. Facebook also tracks the websites a user uses outside of Facebook through Facebook Beacon, and sells the tracked "social actions" without the actual user information.

Two types of information about a user can be gathered from a social network:

1. **Information that a user may share**: *personal details*, *posts*, *contacts*, *biographical information*, *media*, *interests*, *location*
2. **Information gathered through electronic tracking**

The information that a user shares becomes public in various ways:

- **User's choice**: A user uses the available privacy setting, and posts information as **public**.
- **Default settings**: Some information may be public by default by the social network settings.
- **Social network change of privacy policy**: The social network may change its privacy policy, which may cause a content that was restrictively posted to become visible.
- **Copy and re-post**: An acquaintance with authorization to access the user's shared-information, may copy and re-post some of the shared information, without the user's permission.
- **Third-party applications**: Third-party applications may be granted by the social network itself to access and view information that was posted privately.
- **Bugs and faults**: Unauthorized users can discover faults and bugs and use them to get access to private data.

Most access rules like RBAC [8], are defined in terms of the degree of relationship required to access a user's data like friends, friends of friends, etc. These rules

are not refined enough to allow for dynamic denial of content from certain peers of the user's community. Papacharissi [24] examines self-presentation and social connection in the digital age, behavioral norms, patterns and routines, etc. They show that OSN users have surfing habits. We can extract a lot of information from those surfing habits, like who are the users' close friends, what are their shopping habits, what are their reading habits, what is the frequency of information sharing with each friend, and use it to define a flexible and refined access rule model, that allows for dynamic denial of content from certain peers of the user's community.

Chapter 3

Related Work

3.1 Social networks and Habits

Papacharissi et al. [24], present several aspects of social networks: structure, evolution, and properties, including identities, communities, and the culture on social network sites. The authors conclude that there are emerging patterns of networked sociability that combine newer social habits with old habits, and social routines that existed in the past have grown in new shape in the new social media. Social media platforms have introduced a space where boundaries between private and public space have become fuzzy, which opens up new possibilities for identity formation. Barabasi [4] examines the structure and evolution of social networks. Most networks has a "scale-free" property, which means that it doesn't depend on a specific node (user), and randomly removing nodes, will not destroy the social network. Every network has a few hubs that hold the whole network together, combining communities that are relatively isolated groups of nodes that work independently. Networks are hierarchical, they are built from communities of communities, which are communities that are grouped together into bigger communities. Networks have a purpose, they spread ideas, knowledge, influence, data, etc. Information is passed from a user to friends, who then pass it on to their friends, and so on.

LaRose et al. [27] discuss the social networking addiction and media habits. Some people develop an obsession with some social network sites, known as "Facebook addiction". For many Internet users, social networking has become a media habit, as a form of automatism. People develop an automatic habit of media consumption, for some it turned into a "bad" habit, that might be termed compulsive, problematic, pathological, or addictive, and for others it turned into a "good" habit. A new category of mental illness, called "Internet usage disorder" has been proposed, including a subcategory of email/text messaging. Models

of media behavior can be extended from an understanding of **Internet habits**. Media favorites, which is the preferred media activity within a particular medium, are themselves habits. Some of them became part of people daily ritual. The authors also discuss the Problematic Internet Use (PIU), which is pathological Internet use in relation to symptoms such of impulse control disorders, etc. In order to examine the structure of media habits within socio-cognitive theory, the authors conducted a research on Internet use, resulting with a model of habitual Internet activities. The authors concluded that social networking services are no more problematic, addictive, or habitual than other online activities, despite their widespread popularity. Also, social networking behavior is usually guided by effective self-regulation.

Papacharissi [24] concludes that there are emerging patterns of networked sociability that combine newer social habits with old habits, including social routines of the past, and reflect social tendencies and tensions that take shape on networked planes of social activity. Social media platforms introduced a space where boundaries between private and public space have become fuzzy, which opens up new possibilities for identity formation.

Kim et al. [16], investigate the cultural difference in motivations for using social network sites, between American and Korean college students. American and Korean students showed a similar pattern of daily use of social networks. A notable difference between American students and Korean students was found in the number of connections included in their "friends" list. The number of connections defines the **Sharing-habits**, it shows whether the user tends to share data. The amount of connections indicates the user's willingness of sharing data with friends. American students reported having a larger amount of friends on average. They tend to focus more on entertaining themselves by making new friends through social networks, while Korean students tend to focus more on existing relationships with socially close others from whom they can acquire useful information and social support.
Patterns for using social networks exists, it is similar among different users of the same community, and it is shaped by culture, history, age, etc.

3.2 Privacy preservation

Privacy preservation can be viewed and handled from various aspects. Carmagnola et al. [11] present research about the factors that help user's identification, and information leakage in social networks, based on entity resolution. They conducted a study on the possible factors that make users vulnerable to identi-

fication and personal information leakage. Their study addresses the following questions:

- How effective are the identification techniques based on cross-site checking?
- How can users protect their privacy?
- What are the main risk factors?
- Do users really care, and is it possible to raise users' awareness by showing them the potential possibilities and risks?

The authors used a technique for user identification based on cross-site checking, and connecting user profiles that are not explicitly linked together by linking of user attributes from different profiles. They conducted a study on a group of people that was asked to perform a search using the built prototype, and complete a questionnaire about their experience of using the prototype. Their prototype cross-checked the users' attributes on different profiles, retrieved profiles with compatible attributes and calculated the probability of match between profiles. The results of the study were compared and integrated with a larger set of results automatically extracted by using an identity aggregator. They found that factors that increase the likelihood of users' identification and increase the privacy risks are: the number of social networks used, social networks' features, and especially the amount of profiles abandoned and forgotten by the users. A user profile is more protected in a huge social network like Facebook, since the presence of users with similar attributes, and homonyms makes the identification process more difficult. Users' awareness to the privacy problem can be increased by encouraging them to use people search engines like 123People [30] to monitor their scattered data in OSNs.

Kleinberg and Ligett [14] studied the problem of sharing information with friends while maintaining privacy, and minimizing personal information leakage. Revealing personal information to social networks' friends, generally involves trade-offs between the benefits of sharing information with friends, and the risks that people will propagate the shared information to someone who is not on friendly terms with the information owner, but who is within the owner's community. The authors characterized the existence and computability of stable information-sharing configurations, in which users do not have an incentive to change the set of partners with whom they share information. They describe the social network as a graph where nodes represent users, and an edge between two nodes indicates adversaries that do not wish to share information with each other. Users are unwilling to be members of a group if it contains one of their adversaries, thus the community is partitioned into disjoint "information-sharing groups", corresponding to the groups who are privy to each others' personal information. They partitioned the community by assigning a label to each node according to the

node's set in the partition, and if two nodes are connected by an edge, they must receive different labels, meaning they are adversaries, and cannot be included in the same set. Their aim is to partition the nodes of the graph into disjoint groups such that no group contains nodes with an edge between them. The problem of information sharing is described as the graph coloring problem, were not all colorings are "safe".The authors present a group-based model which is a variant of graph coloring [31]; the coloring problem is augmented with an additional constraint requiring that the partition into color classes be stable, to a certain kind of defection. The model focuses on the partition induced by the sharing of a single type of information, thus different partitions corresponds to sharing different types of information. A conflict-free configuration is an assignment of the nodes into groups such that no two adversaries, with an edge between them, are in the same group. The authors analyzed the stability of solutions for this problem, and the incentive of users to change the set of partners with whom they are willing to share information.

Social networks are of interest to researchers from many disciplines, the data of interest might contain sensitive information, thus, it cannot be released as is. There is a need for anonymization of the data prior to publication, without depraving the data in value. Data anonymization trades off with utility, the released data is anonymized, but still holds enough utility, and preserves privacy to some accepted degree [25]. In [32], Tassa and Cohen, handle the information release problem by manipulation of the released data. They present algorithms to compute an anonymization of the released data to a level of k-anonymity, the algorithms can be used in sequential and distributed environments, while maintaining high utility of the anonymized data. In the basic form social networks are modeled by a graph where nodes correspond to entities, and edges denote relations between the entities. There are three types of methods for privacy reserving with k-anonymity:

1. Edge additions or deletions in a deterministic procedure.
   It is assumed that the adversary has a background knowledge regarding some property of its target node, so the graph is modified such that it becomes k-anonymous with respect to the assumed property.
2. Add noise to the data.
   In order to prevent adversaries from identifying their target in the network, or inferring the existence of links between nodes, perform random additions, deletions or switching of edges.
3. Clustering.
   The graph is not altered, nodes are clustered together into super-nodes of size at least k, where k is the required anonymity parameter, and the graph data is published in that coarse resolution.

The authors present sequential and distributed clustering algorithms for anonymizing social networks, which produce anonymization with better utility than those achieved by existing algorithms.

Record linkage is the process of identifying which records in two or more databases correspond to the same entity. Data quality activities like data pre-processing and data integration, are important aspects of record linkage. Record linkage is also known as data matching or entity resolution. Record linkage has several aspects and challenges, beside privacy and confidentiality, it includes scalability to large databases, accurate matching and classification. The privacy and confidentiality challenge arises because the linkage process use commonly personal identifying data of individuals, like names, addresses and dates of birth. Vatsalan et al. [6] conducted a survey of 'privacy-preserving record linkage' (PPRL) techniques, with an overview of techniques that allow the linking of databases between organizations while at the same time preserving the privacy of these data. In this paper they present taxonomy of PPRL which characterize the known PPRL techniques along 15 dimensions, highlight shortcomings of current techniques avenues for future research. The PPRL taxonomy is divided to 5 categories: Privacy aspect, linkage techniques, theoretical analysis, evaluation, and practical aspects. Each is divided to 3 different categories, thus providing 15 dimensions. The authors review each dimension, and describe the required additional work needed. For example, regarding privacy aspects, PPRL on multiple databases is required, most work in PPRL thus far has concentrated on linking data from two database owners only. PPRL for malicious adversaries is required, most solutions proposed so far assume the Honest-but-curious behavior (HBC) adversary model [34, 29].In the HBC model, parties are curious in that they try to find out as much as they can about the other party's inputs while following the protocol.

3.3 Privacy and Information flow

Ranjbar and Maheswaran [1], describe the social network as a graph where nodes represent users, and an edge between two nodes indicates that those two users are friends that wish to share information. They present algorithms for defining communities among users, where the information is shared among users within the community, and algorithms for defining a set of users that should be blocked in order to prevent the shared information from reaching undesirable recipients (adversaries), and leaking outside the community. In online social networks, communities are subsets of users connected to each other; the community members have common interests and high levels of mutual trust, it can be described by

a connected graph, where each user is a node in the graph, and an edge connecting two nodes indicates a relationship between two users. A community is defined by the authors from the view point of an individual user. *myCommunity* is defined as the largest sub-graph of users who are likely to receive and hold the information without leaking.

The way information spreads on the web while using social networks is determined to a large extent by human decisions. They identify two main challenges in defining new access control techniques and controlling confidentiality of information on OSNs: the first one is enforcing usage conditions. Since sharing information in social network is not governed by precise usage policies, it is hard for a user to track the release information. The second is that information sharing in social networks is not automatically coupled with the level or the direction of interactions, and there is no simple way to avoid information sharing with undesirable friends. They provide algorithms that use the Monte Carlo method to compute an initial release set (user's community), which is the set of users that are likely to receive data from that user. By defining a sub-community for each user, the algorithm prevents information from reaching users outside the initial release set.

This set is called $\alpha - myCommunity$, where α is a communication intensity threshold, for a given communication. The authors provide an algorithm that for an acceptable threshold of information leakage, defined by α, compute the set of friends who might leak the shared information to an adversary; those friends should be blocked, and the user should stop sharing information with them.

Our study is based on the ideas described in Ranjbar and Maheswaran [1]'s paper; while they only share information within the defined community, and block users that might leak information to adversaries, we relax the limitation defined in their study, and block only edges on the path to the adversaries, instead of blocking all the information from the source user to the users that might leak the information. In a dynamic network there is the problem of enforcing the flow decisions.

Jaehong and Ravi [23] present the ORIGIN CONTROL (ORCON) access control model where every piece of information is associated with its creator forever. Originator Control is an access control policy, where recipients must gain approval for re-dissemination of disseminated digital object from its originator. Originator control policies address the problem of usage control (UCON) and digital rights management (DRM). Current commercial DRM are mainly based on payment, and thus they do not include enforcement of access control policies. The authors extend the traditional originator control solutions to enforce access control policies outside of a closed system environment where a central control authority is not available.

Usage control concept is originally based on virtual machines and control sets

(the license). Electronic information is available freely, the access to the information is controlled by using the control set that represents the approved access rights or bypassed by the virtual machine.

3.4 Network Flows

Network flows [26] is a problem domain that is common to several research fields, including applied mathematics, computer science, engineering, management, and operations research. A flow network or a transportation network in graph theory, is a directed graph where each edge has a capacity and receives a flow. The amount of flow on an edge cannot exceed the edge capacity, and the amount of flow into a node must equal the amount of flow out of it, unless the node is a source which has only outgoing flow, or sink which has only incoming flow. A network can be used to model traffic in a road system, fluids in pipes, currents in an electrical circuit, or anything in which something travels through a network of nodes.
In a social network described as a directed graph, where users are the nodes of the graphs, and a relationship between two users is an edge connecting their respective nodes, we refer to the probability of two users sharing data, as the flow passing from one user to the other along the connecting edge. An S-T cut in a graph is a partition of the graph into two non-empty disjoint sets S, and T. The cut-set contains all the nodes connected by edges crossing from S to T. The minimum S-T cut problem is to minimize the S-T cut in a graph, that is, to determine S and T such that the capacity of the S-T cut is minimal.
The max-flow min-cut theorem states that the maximum value of an S-T flow is equal to the minimum capacity over all S-T cuts. The meaning is that the maximum amount of flow passing from the source to the sink is equal to the total weight of the edges in the minimum cut.

There are various implementations of the Max-flow-min-cut algorithm, each with different complexity; let $|V|$ be the number of vertices in a graph, $|E|$ is the number of edges in the graph, U is the maximum edge capacity, and F is the maximum flow value, the min-cut complexity depends on the max-flow-min-cut implementation, and is varied from $O(|V|^2 \cdot U)$ [7] to $O(|E| \cdot |V|^{2/3} \cdot \log(\frac{|V|^2}{|E|}) \cdot \log U)$ [3]. The Ford-Fulkerson [19] finds the maximum flow by sending the minimum of the residual capacities on the path to the sink (end node) as long as there is an open path through the residual graph. It is based on the concept of residual network. The basic idea behind the method is iterative improvement: start with a zero flow from source (start node) to sink (end node), and as long as there is a path from the source to the sink with available capacity on all edges in the

path, send flow along one of these paths. Then find another path, and so on. A path with available capacity is called an augmenting path. By adding the flow of the augmenting path to the flow already established in the graph, the maximum flow will be reached when no more augmenting paths with flow can be found in the graph.If all weights are rational the algorithm is guaranteed to terminate and find the maximum flow, otherwise it is possible that the algorithm will not converge to the maximum value.
Edmonds-Karp [13] algorithm is a specialization of Ford-Fulkerson algorithm that uses breadth-first search to find augmenting paths, with guaranteed termination and a run-time independent of the maximum flow value. It computes the maximum flow in a flow network in $O(V \cdot E^2)$ time
Dinic's [10] blocking flow algorithm builds in each phase a layered graph with breadth-first search on the residual graph. The maximum flow in a layered graph can be calculated in $O(|V|\,|E|)$ time, and the maximum number of the phases is $|V| - 1$. In networks with unit capacities, Dinic's algorithm terminates in $O(\min |V|^{2/3}, |E|^{1/2}\}\,|E|)$ time. Another version of Dinic's algorithm uses dynamic trees data structure to speed up the maximum flow computation in the layered graph to $O(|E| \log(|V|))$.
The push-relabel algorithm by Karzanov [18] maintains a preflow, which is a flow function with the possibility of excess in the vertices. The algorithm runs while there is a vertex with positive excess that is an active vertex in the graph. The push operation increases the flow on a residual edge, and a height function on the vertices controls which residual edges can be pushed. The height function is changed with a relabel operation. The proper definitions of these operations guarantee that the resulting flow function is a maximum flow.
Push-relabel algorithm with FIFO vertex selection rule by Goldberg and Tarjan [2] is a variant to the push-relabel algorithm which always selects the most recently active vertex, and performs push operations until the excess is positive or there are admissible residual edges from this vertex.
Another variant of the push-relabel algorithm is the push-relabel algorithm with dynamic trees, which builds limited size trees on the residual graph regarding to height function. These trees provide multilevel push operations.
KRT (King, Rao, Tarjan)'s algorithm [33] ia a randomized algorithm that efficiently play a certain combinatorial game that arises during the computation along with a strategy that yields a deterministic algorithm for computing the maximum flow.
Karger and Stein [17, 5] present a randomized strongly polynomial sequential algorithm for finding the minimum cuts in weighted undirected graph, which runs in $O(E \cdot V^2)$, and when parallelized runs in $O(log^2 V)$. The Contraction algorithm contracts two vertices of the graph into a new multi-vertex, removing

the edges between them, and replacing the edges to other vertices with edges from the new multi-vetex to the other vertices. Each iteration randomly selects an edge and contract the two vertices connected by the selected edge into a new multi-vertex, until the graph contains only two multi-vertices which are two sets of vertices that were contracted into two multi-vertices. These sets are the cut. If we repeat the contract process enough times the algorithm will find the minimum cut.

We use both algorithms, Edmonds-Karp algorithm [13], and Karger and Stein [17, 5] to find a set of edges that are candidates for removal from the graph, to disable information flow from the source to the sink, which is the source's adversary.

Chapter 4

The OSN Sharing-habits based privacy assurance Problem

Using social networks to share information with friends and acquaintances might lead to leakage of the shared information to adversaries. To minimize the probability of information leakage, the social network is analyzed to determine based on users' habits, the probability of information flow through network connections. In a graph representation of the network, where edges indicate relationship between users, the challenge is to select the set of edges that should be blocked to prevent leakage of the shared information to unwanted recipients (adversaries). In this section we define the general problem of privacy assurance in OSN and our proposed method that uses information from users sharing-habits.

4.1 Problem definition

Let $G = (V, E)$ be a directed graph that describes a social network, where V is the set of network's users, and E is the set of directed and weighted edges representing the users' information flow relationships. An edge $(u_i, u_j) \in E$ exists only if u_i shares information with u_j.
Ego is an individual focal node, it is the specific user from which we consider the information flow. A network has as many egos as it has nodes, *ego-community* is the collection of ego and all nodes to whom ego has a connection at some path length.
The distance between two vertices, $dist_G(u_i, u_j)$ is the length of the shortest path from u_i to u_j in G.
The δ-community of a user, represented by the ego vertex u_i is the sub-graph $G_\delta(u_i)$=($V_\delta(u_i)$,$E_\delta(i)$), where for each $v_i \in V_\delta(u_i)$, $v_i \neq u_i$, $dist_G(u_i, v_i) \leq \delta$. Let $V_\delta(u_i)$ be the set of nodes that consists the ego-node u_i and all the nodes

u_j such that $dist_G(u_i, u_j) \leq \delta$, $E_\delta(i)$ is the set of edges on the paths from the ego-node u_i to the nodes within $V_\delta(u_i)$ where $dist_G(u_i, v_i) \leq \delta$. The size of the community is determined by the distance from the ego node and also by the density of the graph.
Graph density is the ratio between the number of edges and the number of nodes in the graph: $Density = \frac{|E|}{|V|\cdot(|V|-1)}$

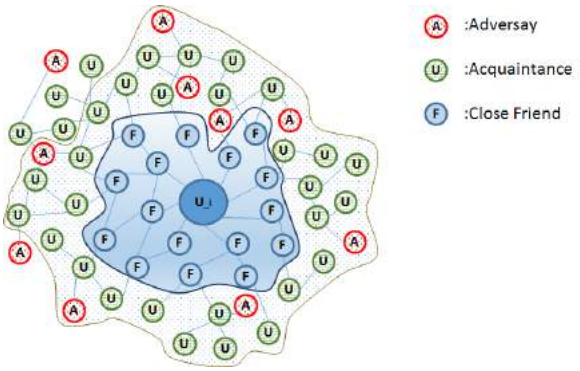

Figure 4.1: $u_i's$ Community Graph

Figure 4.1 describes an *ego-community* graph for the *ego* node u_i. The dotted area surrounds $u_i's$ δ-community graph where $\delta = 4$, i.e., all acquaintances within distance ≤ 4. The blue area surrounds all $u_i's$ friends within distance ≤ 2 denoted $u_i's$ β-community where $\beta = 2$.
As shown by the figure the δ-community of friends is usually much larger than the β-community of close friends.

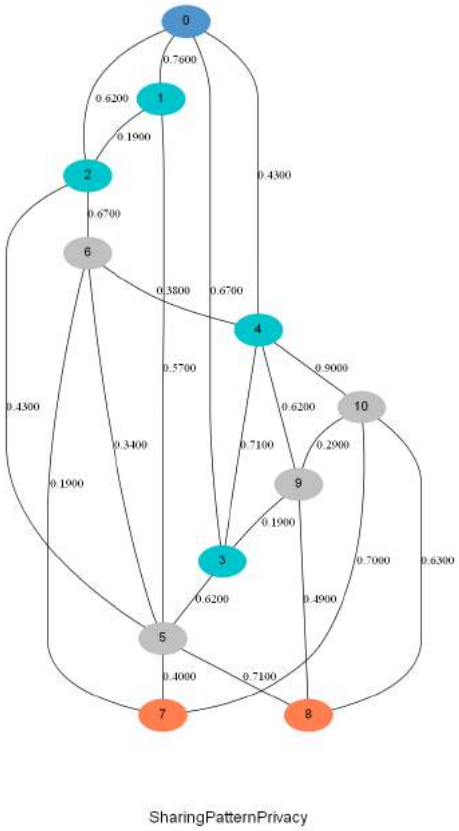

Figure 4.2: The Community Graph of node 0 with close friends of distance 1

Figure 4.2 describes a community graph, where the ego-user is node 0 colored with blue, nodes $1, 2, 3, 4$ are his close friends of distance 1, colored with turquoise, and nodes $7, 8$ are his adversaries colored with coral.
The capacity of an edge is a mapping $c : E \to \Re^+$, denoted by $c_{u_i u_j}$.
It represents the maximum amount of flow that can pass through an edge.
A flow is a mapping $f : E \to \Re^+$, denoted by $f(u_i, u_j)$, subject to the following two constraints:

- **Capacity Constraint:** $\forall (u_i, u_j) \in E : f_{u_i u_j} \leq c_{u_i u_j}$.
- **Conservation of Flows:** $\forall u_k \in V \backslash \{u_i u_j\} : \sum_{\{u:(u,u_k)\in E\}} f_{u u_k} = \sum_{\{u':(u_k,u')\in E\}} f_{u_k u'}$. The total incoming flow to V equals the total outgoing flow from V, there is no generation of additional flow.

The value of flow is defined by $|f| = \sum_{u_j \in V} f_{u_i u_j}$,where u_i is the source of G. It represents the amount of flow passing from the source vertex u_i, to the sink vertex u_j. To compute the flow, we use the log of the edges' probabilities on a

path between u_i and u_j: $|f| = \sum_{u_i,u_j \in V} \log p_{i,j}$, where u_i is the ego node.
Given a set $T \subseteq V$ of vertices let $f_G(u_i, T)$ denote the maximum flow value from u_i over T.
We use the following definitions as defined by Ranjbar et al. [1]:
p_i is the probability that user u_i is willing to share the information with some of his friends.

$$p_i = \begin{cases} (outflow/inflow) & (outflow < inflow), \\ 1 & (outflow \geq inflow). \end{cases} \tag{4.1}$$

In a period of time:

- Outflow is the number of sharing interactions from u_i to his friends.
- Inflow is the number of sharing interactions from $u_i's$ friends to u_i.

The likelihood of u_i sharing information with u_j along the edge (u_i, u_j) is represented by the weight on the edge $w_{i,j}$. This weight is derived from the relationship between u_i and u_j, it is a fixed number indicating the willingness of u_i to share information with u_j. It may be set by the user and usually it does not change.
The probability of flow between two neighbor users, u_i and u_j is denoted as p_{ij}, and calculated by $p_{i,j} = p_i \times w_{i,j}$. ($w_{i,j}$ may represent apriori probability, while p_{ij} is the actual probability, based on the number of interactions.)
We assume that the user behavior is consistent; user u_i shares all the data with user u_j with probability $p_{i,j}$. This probability can change with time, but it does not depend on the content of the shared information.
For example, if in figure 5.1, the number of sharing interactions from user 4 to his friends is 23, (13 to user 3 and 10 to user 5), and the number of sharing interactions from $4's$ friends to 4 is 30, (9 from user 7 and 21 from user u_0), p_i will be $23/30 = 0.77$. If $w_{4,5}$ is 0.8 and $w_{4,3}$ is 0.92, the resulting $p_{4,5}$ is 0.62 and $p_{4,3}$ is 0.71.
The Probability of Information Flow (PIF), is the maximum probability of information flow throughout the entire paths between u_i and u_j.
A path probability flow between u_i and u_j is the flow of the edge with the minimum $p_{i,j}$. It is denoted as $PATH_{i,j}$. The PIF is the maximum among of all paths between u_i and u_j of $PATH_{i,j}$. The function f which denotes flow is computed by the PIF.
To prevent information flow from one user to another we search for the minimal set of edges that when removed from the community graph, or blocked, disables the flow. We denote this set of blocked edges as B. Note that after edges are removed, the PIF and therefore f should be recomputed.

4.1.1 Research Question

Our aim is to enable a user to share information with as many friends and acquaintances as possible, while preventing information leakage to adversaries within the user's community. Ranjbar et al. [1] describe a method for sharing information within the source user defined community, while blocking users (friends and acquaintances) that might leak information to adversaries. We relax the limitation due to blocking friends, and instead of blocking all the information from the source user u_i to the users that might leak the information, we block only edges on the path from u_i to her adversaries. For a source u_i, we use the following criteria to define and evaluate the resulting u_i ego-community graph:

1. **Minimum Friends Information Flow** : the minimum information flow from u_i to every user within her community must preserve a certain percentage of the original information flow to every user denoted by α.
   Let $G_\delta(u_i) = (V_\delta(u_i), E_\delta(u_i))$ be the δ-community of $u_i, u_j \in V(u_i)$

$$f(u_i, u_j) \geq \alpha \cdot f_{original}(u_i, u_j) \tag{4.2}$$

2. **Close Friends Distance** : Close friends are defined by their distance from u_i. $G_\beta(u_i) = (V_\beta(u_i), E_\beta(u_i))$ is the β-community of $u_i, u_j \in V(u_i)$, $\beta < \delta$. This criteria reflects the requirement that all the users within $u_i's$ β-community must receive the entire information from u_i, and cannot be blocked.
   Let B be the set of blocked edges, than

$$B \subset \{(u_s, u_t) | d_{G_\delta}(u_i, u_s) \geq \beta, u_s, u_t, u_i \in V_{G_\delta}(u_i)\} \tag{4.3}$$

   We assume that there are no adversaries within $u_i's$ β-community.

3. **Maximum Adversaries Information Flow** : the maximum information flow from u_i to each of her adversaries cannot be more than γ from the original information flow to each adversary. $u_{adv} \in$ {set of adversaries}

$$f(u_i, u_{adv}) \leq \gamma \cdot f_{original}(u_i, u_{adv}) \tag{4.4}$$

4. **Fuzziness** : the blocked edges list should not be predicted.

For example the threshold parameters can be: $\alpha = 0.9$, $\beta = 2$, and $\gamma = 0.1$. The problem goal is to remove the least number of edges such that the three inequalities 2,3,4 are satisfied. A detailed example for this process is given in Section 5.1.

4.2 Cuts in graphs

A cut in a graph is a set of edges between two subsets of a graph, one containing u_i, and the other containing $u_i's$ adversaries, such that when removed, prevents information flow from one subset to the other.
A naive algorithm for solving the problem would be an algorithm that finds any cut between the adversaries' set and $u_i's$ community, and defines this cut as the blocked edges list. Algorithm 1 is a naive algorithm for blocked users.

Algorithm 1 Naive algorithm for blocked users

Input

$G = (V, E)$ a directed graph that describes the social network.

u_i the ego user.

δ the community distance.

$AdversariesList$: the list of $u_i's$ adversaries.

Output

B:the set of blocked edges.

1: set $B = \emptyset$
2: **for all** $u_j \in V$ and $(u_j \notin AdversariesList)$ and $(dist_G(u_i, u_j) \leq \delta)$ **do**
3: insert u_j to $V_\delta(u_i)$
4: **for all** $u_j \in AdversariesList$ **do**
5: insert u_j to $V_\delta(adversaries)$
6: Find a cut between the community graph, $V_\delta(u_i)$ and the adversaries $V_\delta(adversaries)$.
7: $C(V_\delta(u_i), V_\delta(adversaries)) = \{(i,j) \| i \in V_\delta(u_i)$ and $j \in V_\delta(adversaries)\}$
8: **for all** $e_{ij} \in C(V_\delta(u_i), V_\delta(adversaries))$ **do**
9: insert e_{ij} to B
10: **return** B

The naive algorithm serves as a basic framework to our proposed solution. However, it is not suitable for our problem, since it doesn't address (1) Minimum Friends Information Flow and (2) Close Friends Distance criteria of our problem. Condition (1) requires maximum information flow from u_i to all members in $u_i's$ community, the naive algorithm finds any cut between the two subsets, without verifying maximum information flow to $u_i's$ community.
Condition (2) requires minimum distance to closed friends, the naive algorithm finds any cut between the two subset, without verifying that edges to close friend aren't removed. While the naive algorithm is not sufficient to our problem, it is important for understanding the theoretical problem defined here.

4.3 Problem complexity

Here we analyze the *Sharing-habits based Privacy Assurance in OSN problem* complexity. We show that a *"Minimum-Distance with Maximum-Flow and Minimum Leakage"* problem, which is a simple subset of the *"Sharing-habits based Privacy Assurance problem"* (defined in subsection 4.1 is NP-Complete, and so does the general *Sharing-habits based Privacy Assurance in OSN problem*. In section 4.4 we present a practical approximating solution for the general problem.

4.3.1 Maximum-Flow Minimum-Leakage problem definition

Let $G = (V + u_i, E)$ be a directed graph with an ego node u_i, which is the source user, and edge capacities/lengths $\{c_e : e \in E\}$.
Given a set $T \subseteq V$ of vertices, let $f_G(u_i, T)$ denote the maximum (u_i, T)-flow value under capacities c_e.
$T \subseteq V$ is the set of $u_i's$ friends.
$S \subseteq V$ is the set of $u_i's$ adversaries.

Objective:
Given a threshold k, find a subgraph $H = (V, E_H)$ of G with $f_H(u_i, S)$ minimum such that $f_H(u_i, T) \geq k$

A particular case of *Maximum-Flow Minimum-Leakage Problem* is the *Minimum-Distance Maximum-Flow Minimum-Leakage Problem* , which is a simple subset of our general problem, were the adversaries are in the boundaries of $u_i's$ δ-community : $dist_G(u_i, u_j) \leq dist_G(u_i, s)$ for all $u_i, u_j \in T, s \in S$.

In the next subsection we present the proof for the claim that the Minimum-Distance Maximum-Flow Minimum-Leakage problem is NP-complete and so does the simple problem, and furthermore does the general problem.

4.3.2 Minimum-Distance Maximum-Flow Minimum-Leakage problem is NP-complete : proof

Here we show that the the Minimum-Distance Maximum-Flow Minimum-Leakage problem is NP-complete and so does the simple problem and furthermore does the general problem.

We start by showing that the Maximum-Flow Minimum-Leakage Problem is a Hitting-Set-hard.

The Hitting-Set Problem
Instance: Collection C of subsets of a set S, positive integer K.
Objective: Does S contain a *hitting-set* for C of size K or less, that is, a subset $S' \subseteq S$ with $|S'| \leq K$ and such that S' contains at least one element from each subset in C.

A ***p*-approximation algorithm** for a minimization problem runs in polynomial time and produces a feasible solution of value at most *p* times the value of an optimal solution; if such an algorithm exists then we will say that the problem admits **approximation ratio *p***. If such an algorithm is unlikely to exist (e.g., if existence of such an algorithm implies P=NP) then we will say that the problem has **approximation threshold *p***.
Given a subset A of nodes of a graph J, let $\Gamma_J(A)$ denote the set of neighbors of A in J.
The following known problem is NP-hard, and moreover, is known to have a logarithmic approximation threshold.

The Graph Hitting-Set Problem
*Instance:*A directed bipartite graph $J = (A + B, E_J)$ with $\Gamma_J(A) = B$ such that every edge in E_J has tail in A and head in B.
Objective: Find a minimum size node subset $A' \subseteq A$ such that $\Gamma_J(A') = B$.

Garey and Johnson showed in [20] that the Hitting-Set is NP-HARD by showing that the hitting-set problem is reduced to the vertex-cover problem.

The Vertex-cover problem
Consider a bipartite graph, with vertices on the left representing sets, vertices on the right representing the universe of elements, and edges representing the inclusion of elements in sets. The objective is to find a minimum cardinality subset of left-vertices which covers all right-vertices.

In the hitting-set problem, the objective is to cover the left-vertices using a minimum subset of the right vertices; by interchanging the two sets of vertices we convert the bipartite graph problem into the hitting-set problem.

Let $n = |V|$, we prove that **Minimum-Distance Maximum-Flow Minimum-Leakage Problem** is **Hitting-Set-hard** to approximate, and in particular obtain the result showed by Feige [12].

Given an instance of Hitting-Set with $|A| = |B|$, we construct an instance of Maximum-Flow Minimum-Leakage Problem as follows (see Figure 4.3):

- Assign to each edge in E_J a capacity of 1.
- Add a new node u_i and a set of edges $E_{u_i} = \{u_i v : v \in A\}$ of capacity $|B|$ each.
- Add a new node u_j and a set of edges $E_{u_j} = \{v u_j : v \in B\}$ of capacity 1 each.
- Set $S = A$ and $T = \{u_j\}$.
- Set $k = |B|$.

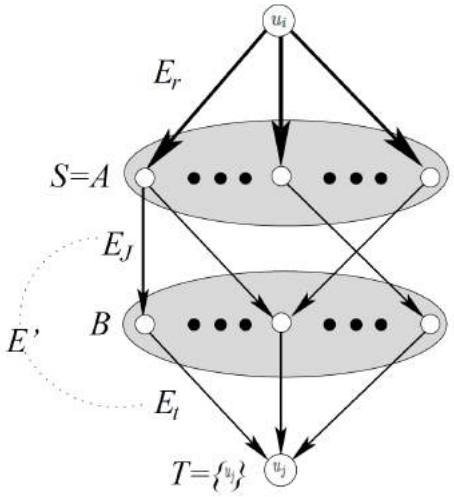

Figure 4.3: Illustration to the Maximum-Flow Minimum-Leakage Problem

Let $E' = E_J \cup E_t$ be the set of edges of capacity 1. Note that for any subgraph H of the obtained Maximum-Flow Minimum-Leakage Problem instance we have:

- $f_{H+E'}(u_i, S) = f_H(u_i, S)$ (adding E' to H does not change the (u_i, S)-flow).
- $f_{H+E'}(u_i, u_j) \geq f_H(u_i, u_j)$ (adding E' to H cannot decrease the (u_i, u_j)-flow).

Feige [12] showed that **Hitting-Set** cannot be approximated within $(1-o(1))ln\,|B|$

unless NP has quasi-polynomial time algorithms, while Raz and Safra [28] established a lower bound of $C \cdot ln\,|B|$, where C is a constant, under the weaker assumption that $P \neq NP$. This is so even when $|A| = |B|$.

4.3.2.1 Theorem 1

Minimum-Distance Maximum-Flow Minimum-Leakage Problem cannot be approximated within:

- $(1 - o(1))$ln n, unless NP has quasi-polynomial time algorithms.
- $C\cdot$ ln n for some constant C, unless $P \neq NP$.

Furthermore, this is so even if $|T| = 1$ *and* $k = \lfloor n/2 \rfloor$.

4.3.2.2 Proof of Theorem 1

Now let H be a feasible solution to the obtained Maximum-Flow Minimum-Leakage Problem instance, namely, $f_H(u_i, u_j) \geq |B|$. Then $f_{H+E'}(u_i, u_j) \geq f_H(u_i, u_j) \geq |B|$, and thus $H + E'$ is also a feasible solution. Moreover, $f_{H+E'}(u_i, S) = f_H(u_i, S)$ and thus the solutions H and $H + E'$ have the same value $f_{H+E'}(u_i, S) = f_H(u_i, S)$.
Let us call a solution H to the obtained Maximum-Flow Minimum-Leakage Problem instance proper if it contains E'. From the above discussion we see that for the obtained Maximum-Flow Minimum-Leakage Problem instance, we can restrict ourselves to proper solutions, without changing the problem. Furthermore, it is not hard to see that we have a bijective correspondence between proper feasible solutions to the obtained Maximum-Flow Minimum-Leakage Problem instance and the original Hitting-Set instance, as follows.

4.3.2.2.1 Corollary 1 .

H is a proper feasible solution to the obtained Maximum-Flow Minimum-Leakage Problem instance if and only if the set of nodes $A' = \Gamma_H(u_i)$ is a feasible solution to the Hitting-Set instance. Furthermore, if H is a proper feasible solution to the obtained Maximum-Flow Minimum-Leakage instance then $f_H(u_i, S) = |B| \cdot |\Gamma_H(u_i)|$; namely, the maximum flow value from u_i to S in H equals $|B|$ times the size of the corresponding solution $A' = \Gamma_H(u_i)$ to the Hitting-Set instance.
This implies that the optimal solution value of the obtained Maximum-Flow Minimum-Leakage Problem instance equals $|B|$ times the optimal solution value of the original Hitting-Set instance. Observing that in our construction $n =$

$|A| + |B| + 1 = 2|B| + 1$ and $k = |B|$, and using the results of [12] and [28], we get that Theorem 1 is valid for Maximum-Flow Minimum-Leakage Problem. To get a similar result for Minimum-Distance Maximum-Flow Minimum-Leakage Problem we slightly modify the construction by adding an edge of capacity 0 from u_i to u_j. This gives $dist_G(u_i, u_j) = 0$, hence $dist_G(u_i, u_j) \leq dist_G(u_i, v)$ for every $v \in S$. Corollary 1 remains valid for this modification. This concludes the proof of Theorem 1, and our claim that the Maximum-Flow Minimum-Leakage Problem is a Hitting-Set-hard.

Since the Maximum-Flow Minimum-Leakage Problem, which is a subset of the simple problem is already NP-hard, so does the general problem: the Sharing-habits based Privacy Assurance in OSN problem.

It is obvious that since the simple problem is already NP-hard, so is the general problem.

In the next section we present several approximations algorithms to the general problem.

4.4 Solution algorithms

As we showed in 4.3 the problem presented here is NP-Complete, here we propose a model for finding the best set of edges that should be blocked to allow maximum information sharing with the community of the information source and minimum information leakage. In section 5.4 we analyze the complexity of the proposed model, and show that it is $O(|V|^2+|V|\cdot|E|+|E|^2)$ when using paralleled contract for finding the initial candidates-set for edges to be blocked, or $O(|V|\cdot|E|^2+|E|^2)$ when using min-cut for finding the initial candidates-set for edges to be blocked.

Our model consists of two major steps: the first is the initialization step in which we create a multi-graph with a super-vertex s_1 containing $u_i's$ β-community, this step is described in subsection 4.4.1. In the second step we present two methods for identifying candidate sets of edges to be blocked as described in subsection 4.4.2.

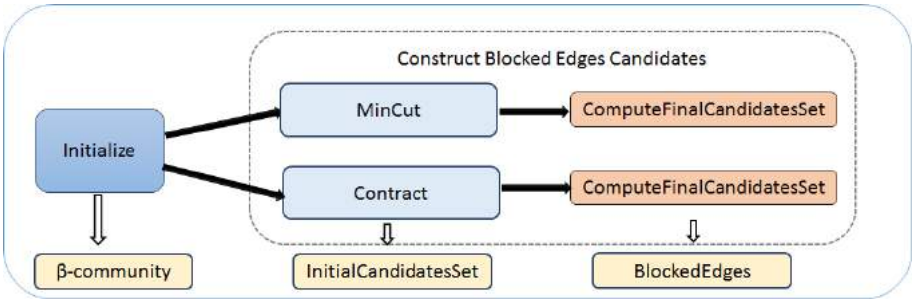

Figure 4.4: Construct Blocked Edges main building blocks

Figure 4.4 describes the main building blocks of the algorithm for defining the edges to be removed from $u_i's$ δ-community in order to prevent information leakage to $u_i's$ adversaries.

Algorithm 2 is a skeleton outlines these steps: It starts with the initialization step (lines 1-8), next it calls the procedure that finds candidates-sets of edges to be blocked (line 11), by using min-cut algorithm, contract algorithm, or both, and then it calls the procedure that examines the proposed set against the required privacy criteria (line 12).

Algorithm 2 Construct blocked edges

Input

u_i: the ego vertex.

$G_\delta(u_i) = (V_\delta(u_i), E_\delta(u_i))$: $u_i's$ δ-community graph.

α, γ: Flow thresholds.

β: β-community distance.

$AdversariesList$: the list of $u_i's$ adversaries.

Output

B:a set with edges to be blocked.

1: $MultiSet$ **function Initialize**$(u_i, G_\delta(u_i), \beta, AdversariesList)$
2: set $s_1 = \{u_i\}$
3: **for all** $(u_j \in G_\delta(u_i))$ and $(dist_{G_\delta}(u_i, u_j) \leq \beta)$ **do**
4: **if** $(u_j \notin AdversariesList)$ **then**
5: insert u_j to s_1
6: **else**
7: **return** $\emptyset$
8: **return** s_1
{————Main————————————}
9: s_1 =**Initialize**$(u_i, G_\delta(u_i), \beta, AdversariesList)$
10: **if** $(s_1 \neq \emptyset)$ **then**
11: $InitialCandidatesSet$ = **ConstructBlockedEdgesCandidates**$(u_i, G_\delta(u_i), s_1, AdversariesList, \alpha, \gamma)$
12: B = **SelectBestBlockedEdges**$(InitialCandidatesSet)$
13: **return** B

Next we describe in detail each one of these building blocks.

4.4.1 Initialization

The δ-community of $u_i's$ consists of all users u_j connected to u_i with a path with distance $\leq \delta$. The β parameter defines the size of the community of close friends. Therefore, a β-community of u_i would be a sub-graph contained in δ-community were $\beta \leq \delta$, as demonstrated in figure 4.1. The privacy criteria that is defined in sub-section 4.1.1 requires that the entire information shared by u_i is shared with $u_i's$ close friends (2). To comply with this requirement, the Initialization step creates a multi-graph with one super-vertex s_1 containing u_i and her close friends with distance $\leq \beta$. This step ensures that no edge on a path between u_i and her close friends will be blocked since they all belong to the same super vertex, s_1 (see figure 4.5).

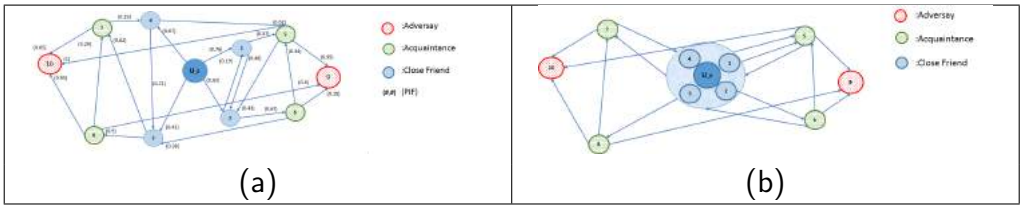

Figure 4.5: u'_0s δ-community graph: (a) u'_0s comuunity (b) after initialization

Figure 4.5(a) describes a δ-community graph for u_0, δ=3, with 10 members, 4 are close friends with distances=1 (blue vertices), 4 acquaintances (green vertices) and 2 adversaries (red vertices). Figure 4.5(b) describes the graph after initialization. The initialization process is depicted in steps 1-8 of the algorithm.

4.4.2 Construct Blocked Edges Candidates

A candidate-set of blocked edges is a cut between two sets of vertices, one set containing u_i, u'_is β-community, and some vertices from of u'_is δ-community. The other set containing the remaining part of u'_is δ-community, and u'_is adversaries.

The candidate-set is evaluated against the privacy criteria we have defined in section 4 and is described later in section 4.4.3. We use the following two methods for finding the initial candidate-set of edges to block:

1. *Min-Cut*: based on Ford-Fulkerson [19] Max-flow-min-cut algorithm, we find the minimum cut between the super-vertex s_1 and each of u'_is adversaries.
2. *Contract*: based on Karger et al. [5] contract algorithm, we find any cut between the super-vertex s_1 and each of u'_is adversaries.

Each candidate set is later evaluated to achieve maximum information flow to community members with minimum information flow to adversaries.

4.4.2.1 Block edges by Min-Cut

Algorithm 3 implements the Sharing-habits privacy assurance based on the max-flow min-cut method by Ford and Fulkerson [19], and then tests for privacy criteria compliance:

1. Find a minimum cut between super-vertex s_1 and u'_is adversaries [19].
2. Check if the cut complies with the required privacy criteria as defined in sub-section 4.1.1 and select the final candidates-set. This process is described in section 4.4.3.

Algorithm 3 Block edges by Min-Cut

Input

u_i: the ego vertex.

$G_\delta(u_i) = (V_\delta(u_i), E_\delta(u_i))$: $u_i's$ δ-community graph, after the initialization step.

α, γ: Flow threshold.

$AdversariesList$: the list of $u_i's$ adversaries.

Output

B:a set with edges to be blocked.

1: set $B = \emptyset$
2: $InitialBlockedEdges$= FindMinCut(u_i,$G_\delta(u_i)$,$AdversariesList$)
3: **if** ($InitialBlockedEdges \neq \emptyset$) **then**
4: B=ComputeFinalCandidatesSet(u_i,$G_\delta(u_i)$,$AdversariesList$,$InitialBlockedEdges$,α, γ)
5: **return** B

4.4.2.2 Block edges by Contract

The minimum cut between the beta community of user u_i, $G_\beta(u_i)$, and $u_i's$ adversaries, found by BlockEdgesByMinCut algorithm, might not be the optimal solution for our problem, since the edges in this cut may not satisfy the privacy criteria. We therefore apply the contract algorithm, to find a variety of other cuts possibly complying with this criteria.

Algorithm 4 implements the Sharing-habits privacy assurance based on the contract method by Karger and Stein [17, 5].

In each iteration, the contract algorithm finds a different cut between the super-vertex containing $G_\beta(u_i)$ and the super-vertex containing $u_i's$ adversaries. The contract algorithm repeatedly contract vertices to super-vertices until it gets two super-vertices connected by a set of edges that defines a cut between the two sets of vertices contained in each super-vertex.

It is important to note three important features of the contract algorithm:

1. The contract algorithm may be called many times until the resulting cut complies with the required privacy criteria, as defined in section 4.1.1.
2. When repeated enough times the contract will find the min-cut.
3. Fuzziness is inherent in the contract algorithm, where in each iteration we randomly select an edge and contract the two vertices connected by the selected edge into a new multi-vertex, thus each time we run the contract we will get a different cut which defines the initial candidates-set of edges to be blocked.

Algorithm 4 is composed of the following main steps:

1. Find a cut between super-vertex $G_\beta(u_i)$ and $u_i's$ adversaries.

2. Check if the cut complies with the required privacy criteria as defined in sub-section 4.1.1 and select the final candidates-set. This process is described in 4.4.3.

Algorithm 4 Block edges by Contract

Input

u_i : the source node.

$G_\delta(u_i) = (V_\delta(u_i), E_\delta(u_i))$: $u_i's$ δ-community graph, after the initialization step.

α, γ: Flow thresholds.

$AdversariesList$: the list of $u_i's$ adversaries.

Output

B:the set with the blocked edges.

1: set $B = \emptyset$
2: $InitialBlockedEdges$= ContractFindCut(u_i,$G_\delta(u_i)$, $AdversariesList$)
3: **if** ($InitialBlockedEdges \neq \emptyset$) **then**
4: B=ComputeFinalCandidatesSet(u_i,$G_\delta(u_i)$,$AdversariesList$,$InitialBlockedEdges$,α, γ)
5: **return** B

Algorithm 5 is called by algorithm 4 to find a cut between two vertices by randomly selecting an edge and contracting the two vertices connected by the selected edge into one super-vertex.

Figures 4.6- 4.8 describe a simple community graph and some steps of one run of the contract algorithm.

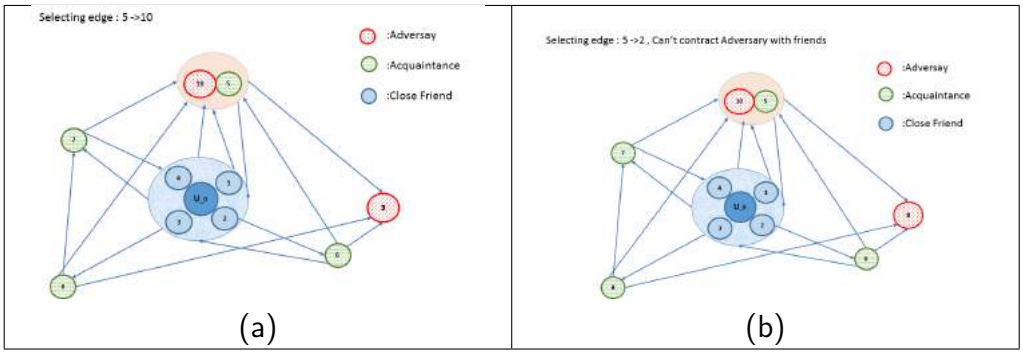

Figure 4.6: Contract: (a) Edge (5,10) was randomly selected, (b) Edge (5, 2) cannot be selected, can not contract a super-vertex containing u_0 with a super-vertex containing $u_0's$ adversary.

Algorithm 5 ContractFindCut

Find a cut in a graph by repeatedly contracting vertices into two super vertices

Input

$G_\delta(u_i) = (V_\delta(u_i), E_\delta(u_i))$: δ-community multi-graph, after the initialization step.

u_i: the source.

$AdversariesList$: the list of $u_i's$ adversaries.

Output

$CutSet$: the resulting cut.

set $CutSet = \emptyset$
repeat
  if (all edges (u, v) are tested) **then**
    return $CutSet$
  else
    choose an edge (u, v) uniformly at random from $E \setminus testededges$
    if (u and v do not contain each others' adversaries) **then**
      contract the vertices u and v to a super vertex w
      keep parallel edges, remove self loops
until (G has only two super-vertices)
set $CutSet =$ the edges between the two vertices
return $CutSet$

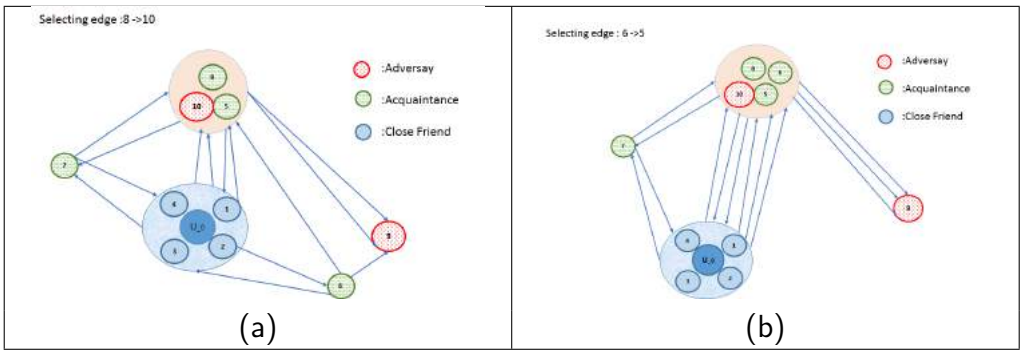

Figure 4.7: Contract: (a) Edge $(8, 10)$ was randomly selected (b) Edge $(6, 5)$ was randomly selected

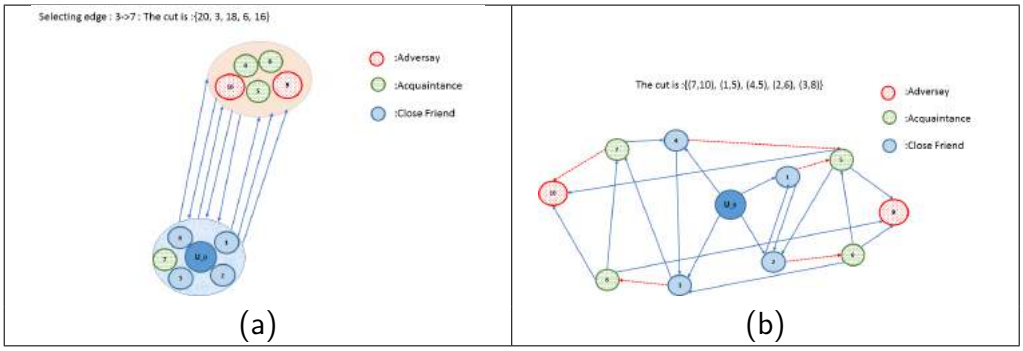

Figure 4.8: Contract: (a) Edge $(3, 7)$ was randomly selected (b) The obtained cut from one run of Contract algorithm

4.4.3 Compute Final Candidates Set

After selecting the initial candidates-set of edges to be blocked, each method uses algorithm 6 for selecting the final candidates-set of edges that should be removed from $u_i's$ δ-community graph. In the first step of the algorithm, we check if by removing the initial-candidates-set of edges from $u_i's$ δ-community graph, the remaining δ-community graph for user u_i complies with the required privacy criteria. If it does not comply with the required privacy criteria, we try to remove edges from the initial blocked candidates-set and insert them back into $u_i's$ δ-community graph, until the remaining community graph complies with the required criteria, or until we tested the entire edges in the initial candidate-set, and could not find a set of edges to be blocked.

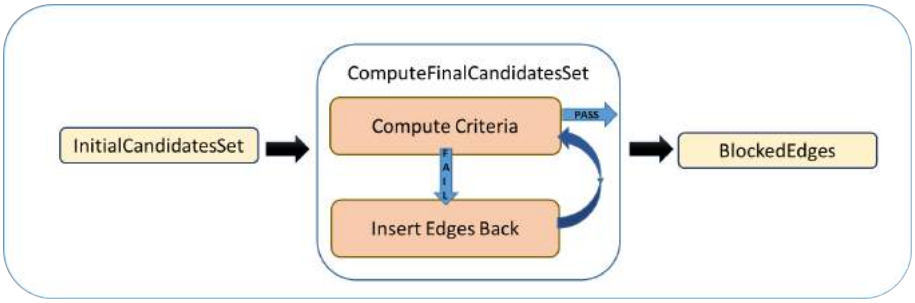

Figure 4.9: Compute Final Candidates Set

Figure 4.9 describes the flow of computing the final-candidates-set. We keep checking if the community graph without the edges from the initial-candidates-set complies with the required privacy criteria and inserting blocked-edges back to the graph, until the remaining community graph complies with the required criteria, or until we tested the entire edges in the initial candidate-set.

The contract algorithm may provide at each run different cuts with different sets of edges to be blocked, the optimal set of edges to be blocked may vary in terms of the number of blocked edges, the amount of information flow to acquaintances and the amount of information leakage to adversaries. The order of selecting the edges to be blocked may provide different acceptable solutions to our problem. Fuzziness is inherent in the contract algorithm, where in each iteration we randomly select an edge and contract the two vertices connected by the selected edge into a new multi-vertex, thus each time we run the contract we will get a different cut which defines the initial candidates-set of edges to be blocked. This is important in order that the adversary will not suspect a blocking which is fixed over a long period of time.
We propose three methods for selecting and removing an edge from the initial candidates-set, and insert the selected edge back to δ-community graph:

1. Randomize: select an edge randomly.
2. Maximum PIF: select the edge with the maximum probability of information flow.
3. Minimum PIF: select the edge with the minimum probability of information flow.

The motivation for the first choice is to add fuzziness to our algorithm. The motivation for the second choice is to reduce maximum flow to adversaries. The motivation for the third choice is to increase flow to friends. It is hard to find a balanced set since the general problem is NP-complete.
Algorithm 6 implements the three methods and algorithm 7 tests the criteria.

Algorithm 6 Compute final candidates-set

Input

u_i : the source.

G_{u_i}: $u_i's$ δ-community multi-graph, after the initialization step.

$AdversariesList$: the list of $u_i's$ adversaries.

$InitialBlockedEdges$: the list of edges to be blocked.

α, γ: Flow thresholds.

$EdgeMethod$: the method for selecting the next edge for unblocking.

Output

$BlockedEdges$: the final set of edges to be blocked.

while ((**ComputeCriteria**$(u_i, G_{u_i}, AdversariesList, InitialBlockedEdges, \alpha, \gamma) \neq TRUE$) and ($InitialBlockedEdges \neq \emptyset$)) **do**
  switch ($EdgeMethod$**)**
  case Random:
    $e \leftarrow$ select random $(u, v) \in E_{u_i}$
  case MaxPIF:
    $e \leftarrow \arg\max_{(u,v) \in E_{u_i}} PIF$
  case MinPIF:
    $e \leftarrow \arg\min_{(u,v) \in E_{u_i}} PIF$
  end switch
  InitialBlockedEdges = InitialBlockedEdges $\setminus (u, v)$
$BlockedEdges = InitialBlockedEdges$
return $BlockedEdges$

This test verifies that the flow to each community member, a friend or an acquaintance, is at least $\alpha\times$ the original flow to that member before blocking the set of edges and the flow to each adversary is no more than $\gamma\times$ the original flow to that acquaintance before blocking the set of edges. In terms of flow value, the set of blocked edges by min-cut algorithm is better than the set blocked by contract algorithm, although the contract when run enough times will eventually find the min-cut set of edges.

Algorithm 7 Compute the required criteria

Input

u_i : the source.

G_{u_i}: $u_i's$ δ-community multi-graph, after the initialization step.

$AdversariesList$: the list of $u_i's$ adversaries.

$BlockedEdges$: the list of edges to be removed.

α, γ: Flow thresholds.

Output

$ComplyCriteria$: indicator whether the community graph without the blocking-set complies with the required privacy criteria.

1: **for all** $v \in G_{u_i}$ **do**
2: **if** $(f(u_i, v) < \alpha \cdot f_{original}(u_i, v))$ **then**
3: **return** $FALSE$
4: **for all** $a_{adv} \in AdversariesList$ **do**
5: **if** $(f(u_i, u_{adv}) > \gamma \cdot f_{original}(u_i, u_{adv}))$ **then**
6: **return** $FALSE$
7: **return** $TRUE$

Chapter 5

Experimental Evaluation

In this chapter we describe the evaluation method we use for the proposed algorithm, and the results we obtained using partial real data from SNAP Datasets [15]. We first demonstrate our methods and the difference between them using a toy community.

5.1 Demonstration on a synthetic community

We demonstrate our algorithms on a small graph representing a community that is based on the example given in [31], containing 11 vertices, and 23 edges with the following community parameters: community distance $\delta = 3$, close friends distance $\beta = 1$, and 2 adversaries, graph density is 0.209. The algorithms are tested with different probabilities of information flow from source user U_0 to the community members.

Figure 5.1 describes the synthetic community graph with high probability of information flow on the edges to adversaries. This situation simulates a collision, for which it is hard to find a cut other than the one that trivially contains the edges to adversaries. In these cases a cut will be found only for privacy criteria that allows very low levels of information flow to u'_0s community (set by α), and/or very high levels of leakage of information flow to u'_0s adversaries (set by γ).

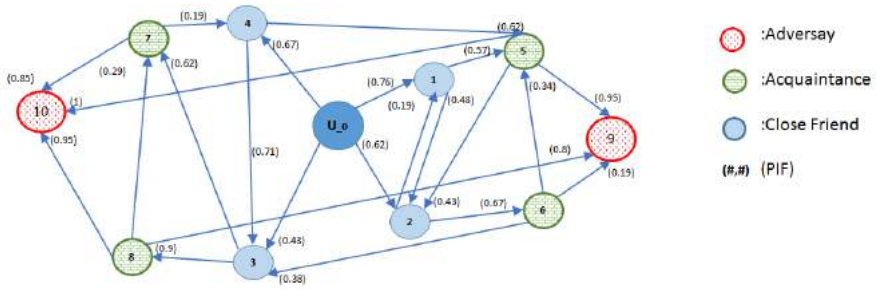

Figure 5.1: Synthetic community graph with collision

In Figure 5.1 U_0 is the source, U_0 has four close friends: $1, 2, 3, 4$, four acquaintances: $5, 6, 7, 8$, and two adversaries: $9, 10$.
Adversary 9 has three incoming edges $\{(6, 9), (5, 9), (8, 9)\}$ with probabilities $(0.19, 0.95, 0.8)$ respectively.
Adversary 10 has also three incoming edges $\{(5, 10), (7, 10), (8, 10)\}$ with probabilities $(1, 0.85, 0.95)$ respectively.

The maximum probability of information flow from U_0 to all other members in the graph is depicted in table 5.1.

User	1	2	3	4	5	6	7	8	9	10
MAX PIF	0.76	0.62	0.4757	0.67	0.4332	0.4154	0.2949	0.4281	0.4115	0.4332

Table 5.1: PIF from U_0 to his community

Next, using this example we show why the contract approach has a better chance to find a good set of edged that can be blocked while satisfying the privacy criteria.

Edge	PIF
(5,9)	0.95
(6,9)	0.19
(8,9)	0.8
(3,8)	0.9
(3,7)	0.62
(5,10)	1

Table 5.2: Candidates found by Min-Cut

Edge	PIF
(4,5)	0.62
(1,5)	0.57
(2,6)	0.67
(3,8)	0.9
(7,10)	0.85

Table 5.3: Candidates found by Contract

Edge	PIF
(5,9)	0.95
(6,9)	0.19
(8,9)	0.8
(7,10)	0.85
(5,10)	1
(8,10)	0.95

Table 5.4: Candidates found by Contract

5.1.1 Block edges by Min-Cut method

The Minimum cut found by Min-Cut method is depicted in table 5.2.

If we remove the initial candidates-set edges from $u_0's$ community graph, the probability of information flow to 7 and 8 will be 0, meaning no flow at all. In the final step of algorithm 6, we try to unblock each edge from the initial candidates-set, to reach the required privacy criteria; in this example the only edge that improves the probability of information flow (PIF) to the community without increasing the information leakage to $u_0's$ adversaries is $(3, 7)$, thus the final candidates-set is $\{(3, 7)\}$. However if we block information flow from $(3, 7)$, the max information flow to 7 is obtained through 8 and since according to table 5.1, the maximum flow to 8 is 0.428, the maximum flow to 7 is: $0.428 * 0.29 = 0.124$, This is clearly a very low value of information flow to community members.

We next show the detailed Flow, unblocking edges according to max PIF:

First, we try unblocking edge $(5, 10)$, the flow to U_0 acquaintances is not improved, but the flow to U_0 adversaries is improved, thus edge $(5, 10)$ remains blocked.

Next we try unblocking edge $(5, 9)$, the flow to U_0 acquaintances is not improved, but the flow to U_0 adversaries is improved, edge $(5, 9)$ remains blocked.

Next we try unblocking edge $(3, 8)$, the flow to 8 is improved and now it is 0.4281, but the flow to 10 was also improved to 0.4067, which is $0.94\times$ original flow to 10, so we can't unblock edge $(3, 8)$, and edge $(3, 8)$ remains blocked.

We continue with the rest edges from the initial-candidates set until we comply with the required privacy criteria, or until we test the entire set of edges from the initial-candidates set. In this example we see that we can't define α, and γ with values that comply with the required privacy criteria, we can try setting α to 0.9, and improve the information flow to u_0 community, but the flow to $u_0's$ adversaries will increase, and we will have to set also γ to 0.9 to comply with criterion (3), and the information from U_0 will leak to adversary 10.

5.1.2 Block edges by Contract method

Two Cuts found by iterations of contract method are depicted in tables 5.3 and 5.4.

If we remove the initial candidates-set edges depicted in table 5.3, the probability of information flow to 5, 6, and 8 will be 0, meaning no flow at all and this does not comply with the required privacy criteria. Therefore following algorithm 6 we unblock each edge from the initial candidates-set, until we meet the required privacy criteria; the final candidates-set is empty, since each edge we unblock not only improves the information flow to $u_0's$ community, but also

increases the information leakage to $u_0's$ adversaries. In this case since users $5, 7,$ and 8 are highly sharing information with the adversaries, the only reasonable cut must include edges to the adversaries.
We next show the detailed Flow of unblocking edges according to max PIF:
First, we try unblocking edge $(3, 8)$, the flow to 8 is improved and now it is 0.4281, but the flow to 10 was also improved to 0.4067, which is $0.94\times$ original flow to 10, so we can't unblock edge $(3, 8)$, and edge $(3, 8)$ remains blocked.
Next we try unblocking edge $(7, 10)$, the flow to U_0 acquaintances is not improved, and to U_0 adversaries is improved, thus edge $(7, 10)$ remains blocked.
Next we try unblocking edge $(2, 6)$, the flow to 5 was improved to 0.1412, which is $0.32\times$ original flow to 5. It also improved the flow to 10 to 0.14124, which is $0.33\times$ original flow to 10, and the flow to 9 to 0.08, which is $0.15\times$ original flow to 9. If we set γ to 0.4 we can unblock edge$(2, 6)$. We continue with the rest of the edges from the initial-candidates set until we comply with the required privacy criteria, or until we test the entire set of edges from the initial-candidates set.
It is obvious that when the edges to the adversaries have high probabilities, the max-flow-min-cut methods might not select those edges, and might not find a solution that comply with the required privacy criteria, while the contract method might find the trivial cut that contains only the edges to the adversaries, as depicted in table 5.4, and thus comply with the required privacy criteria.

5.2 Test on SNAP Database

We evaluated our algorithms using partial real data from Facebook, Twitter and Google+ networks data, available from Stanford Large Network Data-set Collection (SNAP) [15]. The SNAP library is being actively developed since 2004 and is organically growing as a result of Stanford research pursuits in analysis of large social and information networks. The website was launched in July 2009.

The first social network graph describes social circles from Facebook (anonymized) and consists of 4,039 nodes (users), and 88,234 edges. The social network graph of Twitter, is a directed graph describing social circles from Twitter which consists of 81, 306 nodes (users), and 1,768,149 edges. The social network graph of Google+, is a directed graph describing social circles from Google+ which consists of 107, 614 nodes (users), and 13,673,453 edges.

In real world, the user's willingness to share information with friends and acquaintances may be set periodically as described in section 4, or by using learning techniques on user's sharing habits. Since interactions between members are not reported in the SNAP datasets, we use the structure and relationship from this database, and assign random probabilities to the edges in the network graph as

described next.

We define four types of users, to express a user's willingness to share information: very high, medium, low, and very low. For each user in the graph we randomly assign a type. For each edge from a user node we randomly assign a probability that conforms to the user's type according to the following ranges: high (0.75-1), medium (0.5-0.75), low (0.25-0.5), very low (0-0.25). The four types are generated uniformly among all network users. In real life these probabilities can be learned from the traffic patterns and users sharing habits.

In each run a random node is selected as the ego-node, the α, γ and δ parameters are set randomly and the β parameter is set to 1. A random number of adversaries (between 1% and 5% of community vertices) are selected from the $\delta-$community of the ego-node. We used three methods for selecting the final candidate-set of edges to block: edges with max PIF (Probability of Information Flow), edges with Min PIF, and random edges. All three methods yield similar results, thus we only present the results from using the max PIF method.

5.2.1 Test On Facebook Database

Figures 5.2- 5.8 present several sub-communities derived from Facebook database by randomly selecting a sharing user with $\delta = 4$, and the results obtained by our algorithms. From these figures we can see that Facebook database is shaped as clusters of sub-communities connected together into bigger clusters. The users in these figures are colored according to their types, the ego-user is colored in steel blue, first degree friends are colored in turquoise, the adversaries are colored in coral, and community members are colored in gray. The red lines depict the initial cut edges and the thick red lines depict the final cut edges,$V = Vertices, E = Edges, D = Density$.

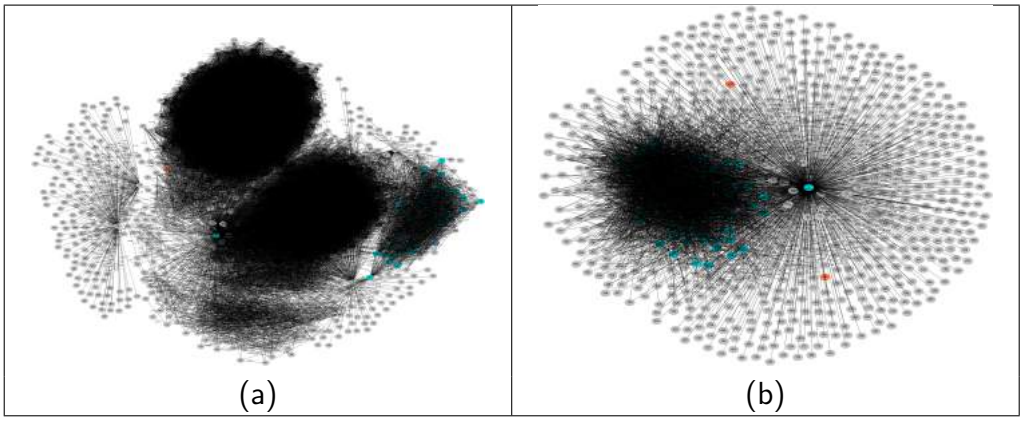

Figure 5.2: Facebook: (a) $V = 987, E = 61831, D = 0.06353$ (b) $V = 789, E = 5205, D = 0.00837$

Figure 5.2 describes two communities derived from Facebook database. The two communities have the same order of vertices, however we can see a clear division to clusters of vertices in community (a) which has a larger number of edges (connections), and consequently larger density.

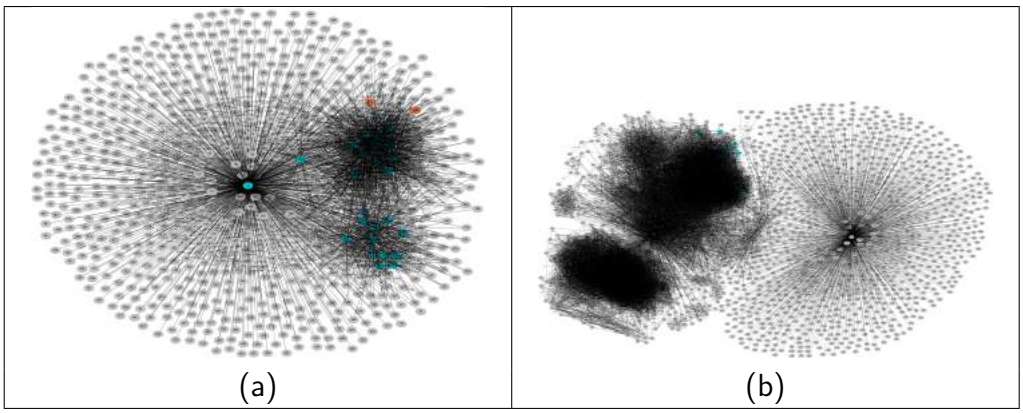

Figure 5.3: Facebook: (a) $V = 789, E = 2038, D = 0.00328$ (b) $V = 1813, E = 30821, D = 0.00938$

Figure 5.3 describes a second set of two communities derived from Facebook database. The two communities have the same order of vertices, however we can see a clear division to clusters of vertices in community (b) which has a larger number of edges (connections), and consequently larger density.

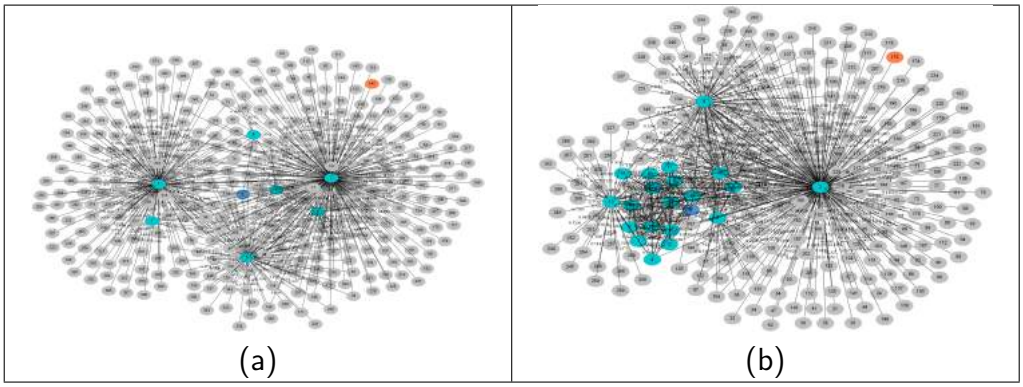

Figure 5.4: Facebook: (a) $V = 345, E = 518, D = 0.00436$ (b) $V = 269, E = 703, D = 0.00975$

Figure 5.4 describes a third set of two communities derived from Facebook database. The two communities have the same order of vertices and the same order of edges, in both communities we can see a clear division to clusters of vertices connected to the ego-node.

Next we show some examples of the initial and final cuts defined by our algorithms.

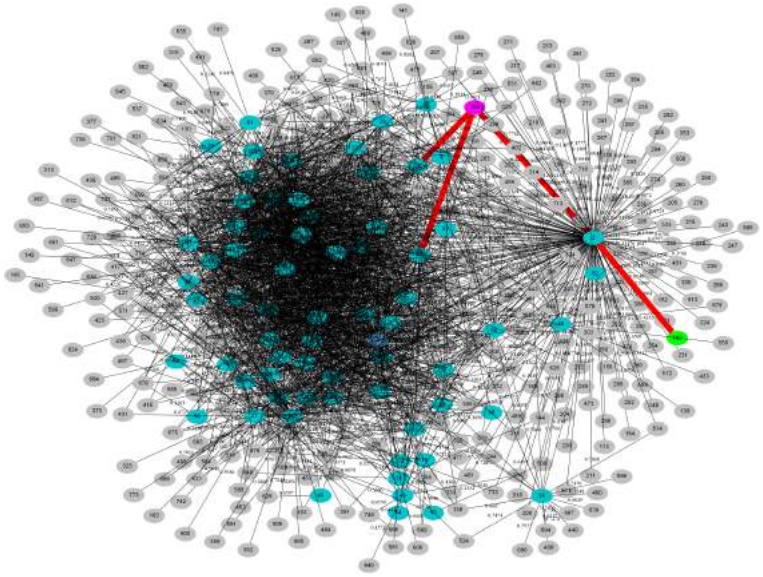

Figure 5.5: Initial Contract edges, sparse

Figure 5.5 is a zoom into the sub-community depicted in figure 5.2 (b), with two adversaries, 182, colored in green, and 209, colored in magenta. The ego-user is marked with red circle, and the red lines depict the initial candidates-set of edges found by the contract algorithm.

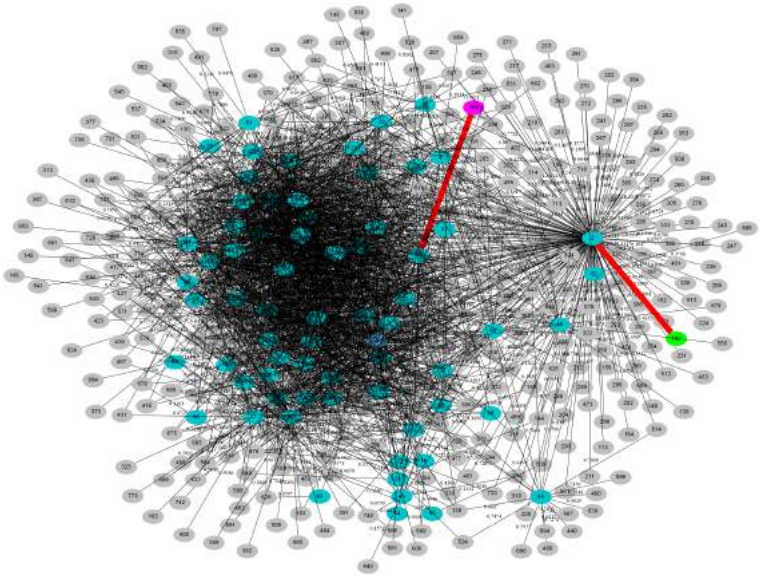

Figure 5.6: Final Contract edges, sparse

In figure 5.6 the red lines represent the final set of edges to be blocked, found after computing the required privacy criteria and unblocking edges from the initial candidates-set of edges depicted in figure 5.5. We can see that the edges to be blocked, are edges on the path to adversary 209 (colored in magenta) only; there is no need to block edges in the paths to adversary 182 (colored in green) since it is connected to the sub-community with one edge from friend 2 to adversary 182 and the probability of sharing information along that edge is very low, 0.0732.

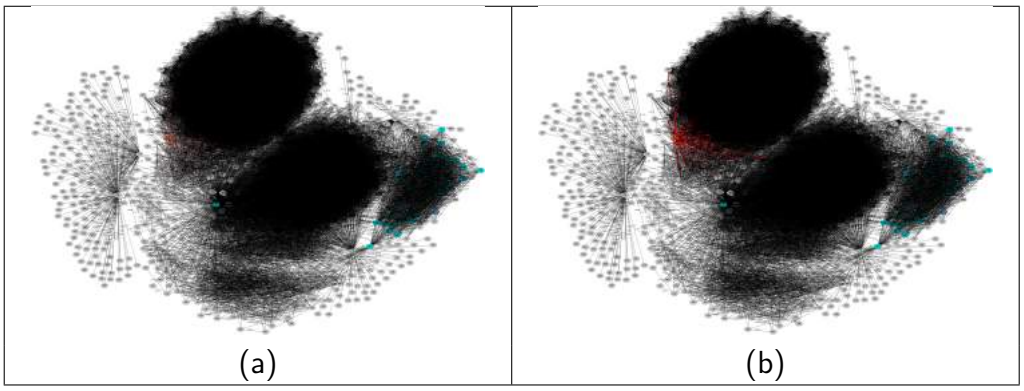

Figure 5.7: Facebook: (a) Initial Min-cut edges (b) Final Min-cut edges

Figure 5.7 describes the initial (a) and final (b) cuts defined by the min-cut algorithm when used on a very large community derived from Facebook database.

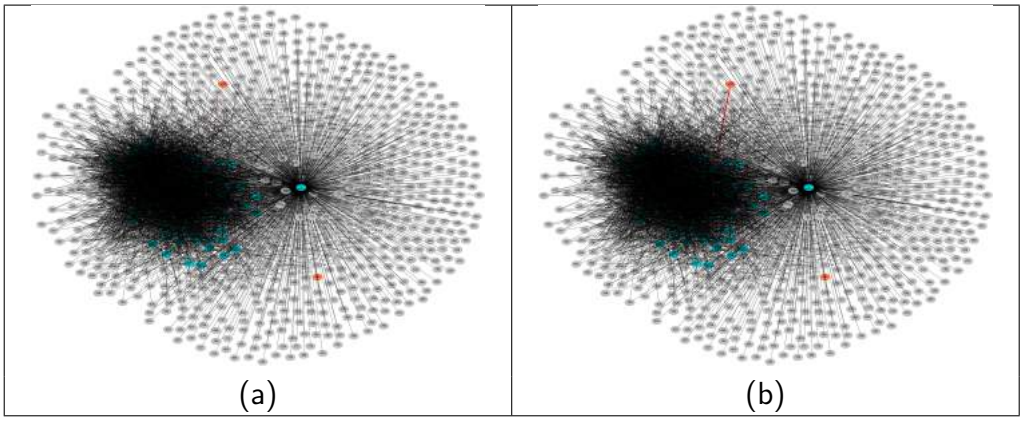

Figure 5.8: Facebook: (a) Initial Min-cut edges (b) Final Min-cut edges

Figure 5.8 describes the initial (a) and final (b) cut defined by the min-cut algorithm when used on a medium size community derived from Facebook database.

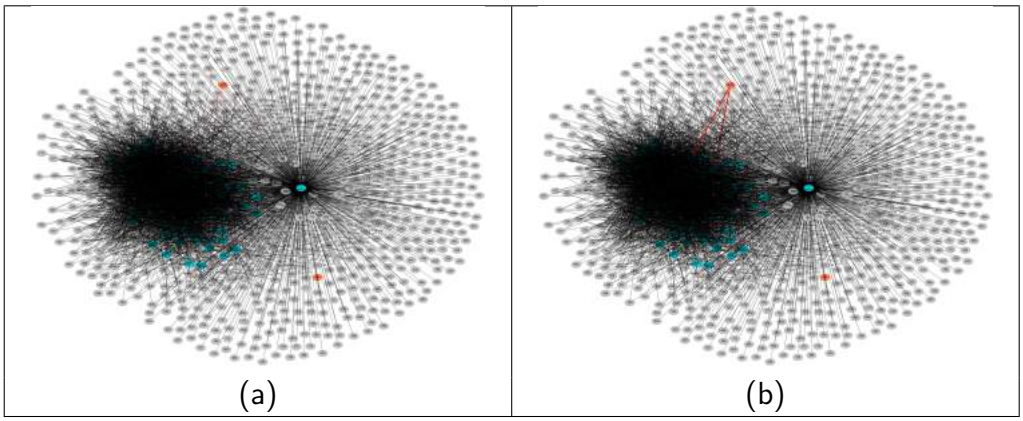

Figure 5.9: Facebook: (a) Initial Contract edges (b) Final Contract edges

Figure 5.9 describes the initial (a) and final (b) cut defined by the contract algorithm when used on the a medium size community derived from Facebook database.

Tables 5.5- 5.6 summarize the results of ten different evaluation runs, for different communities.

Run	Density	δ	Friends	Adversaries	Vertices	Edges
1	0.0087	4	15	2	334	968
2	0.00226	4	26	3	1036	2428
3	0.00308	4	40	10	1495	6886
4	0.0889	4	29	2	206	3755
5	0.06353	4	36	1	987	61831
6	0.00837	4	68	2	789	5205
7	0.00328	4	24	2	789	2038
8	0.00938	4	24	1	1813	30821
9	0.00436	4	7	1	345	518
10	0.00975	4	20	1	269	703

Table 5.5: Facebook sub-communities Data size

Table 5.5 presents ten runs of different sub-communities derived from Facebook database with close friends distance, β set to 1. The community size is derived from the sharing user (ego node) and friends column refers to the amount of first degree friends of this user. These ten runs are used as input to our algorithms and the results are presented next.

Run	α	γ	$Density$	MinCut Initial Edges	MinCut Final Edges	Contract Initial Edges	Contract Final Edges	Remark
1	0.5	0.5	0.0087	2	0	7	7	Blocked edges to adversary
1	0.783	0.5654	0.0087	2	2	7	7	Blocked edges to adversary
1	0.9657	0.5802	0.0087	2	2	7	7	Blocked edges to adversary
2	0.5	0.5	0.00226	2	2	2	2	Blocked edges to adversary
2	0.9587	0.5506	0.00226	2	2	2	2	Blocked edges to adversary
2	0.056	0.4266	0.00226	2	2	2	2	Blocked edges to adversary
3	0.5	0.5	0.00308	29	29	5	0	MinCut blocked edges to adversary
3	0.8867	0.0376	0.00308	29	29	12	0	MinCut blocked edges to adversary
3	0.8385	0.1065	0.00308	29	29	5	0	MinCut blocked edges to adversary
4	0.5	0.5	0.0889	2	0	10	6	Blocked mixed edges
4	0.7776	0.4436	0.0889	2	0	63	43	Blocked mixed edges
4	0.0846	0.6478	0.0889	2	0	286	181	Blocked mixed edges
5	0.5	0.5	0.06353	39	0	2	0	No edges to be blocked
5	0.4292	0.0226	0.06353	39	35	2	2	Blocked edges to adversary
5	0.4454	0.6157	0.06353	39	35	2	2	Blocked edges to adversary
6	0.5	0.5	0.00837	2	1	4	1	Blocked edges to adversary
6	0.9251	0.4224	0.00837	2	1	4	1	Blocked edges to adversary
6	0.0971	0.5569	0.00837	2	1	4	2	Blocked edges to adversary
7	0.5	0.5	0.00328	2	0	18	0	No edges to be blocked, ego-user rarely shares data
7	0.9183	0.408	0.00328	2	0	18	0	No edges to be blocked, ego-user rarely shares data
7	0.2047	0.746	0.00328	2	0	18	0	No edges to be blocked, ego-user rarely shares data
8	0.5	0.5	0.00938	16	0	2	0	No edges to be blocked, adversary is in the middle of sub-community, users frequently share data with the adversary
8	0.0764	0.679	0.00938	16	15	2	2	Blocked mixed edges, adversary is in the middle of sub-community, users frequently share data with the adversary
8	0.3549	0.9095	0.00938	16	0	2	0	No edges to be blocked, adversary is in the middle of sub-community, users frequently share data with the adversary
9	0.5	0.5	0.00436	1	0	1	0	No edges to be blocked, ego-user rarely shares data
9	0.4328	0.3009	0.00436	1	0	1	0	No edges to be blocked, ego-user rarely shares data
9	0.7397	0.9392	0.00436	1	0	1	0	No edges to be blocked, ego-user rarely shares data
10	0.5	0.5	0.00975	1	0	1	0	No edges to be blocked,, ego-user rarely shares data
10	0.8289	0.9999	0.00975	1	0	1	0	No edges to be blocked,, ego-user rarely shares data
10	0.3728	0.0514	0.00975	1	0	1	0	No edges to be blocked,, ego-user rarely shares data

Table 5.6: Facebook Evaluation Runs Results

Table 5.6 presents the results obtained by ten runs, with close friends distance, β set to 1.

Columns 2-3 present the threshold parameters used for each run. For each community graph we performed the algorithms with medium thresholds, ($\alpha = 0.5, \gamma = 0.5$), and with random thresholds. Columns 5-6 and 7-8 present the initial and final set of edges to be blocked found by min-cut and contract algorithm respectively. The remark indicates which kind of edges are the candidates for blocking. We can see that when the adversaries are close to the community's boundary ($\delta = 4$), and no maximum path to a community member passes

through an adversary vertex, (e.g., runs 1,2,5,6 and 7) the solution is trivial and the blocked edges are the edges from community members to the adversaries.

When the adversary is in the middle of the community's boundaries, and there is a path with maximum information flow to a community member that passes through an adversary vertex (e.g., runs 8,9 and 10), it is highly likely that the maximum information flow will be reduced considerably and therefore no good solution can be obtained for high levels of α.

When the ego-user rarely shares data with community member, there is no need to block edges, or the solution is the trivial solution (e.g. runs 7 and 9).

5.2.2 Test On Twitter Database

Figures 5.10- 5.14 present six sub-communities derived from Twitter database by randomly selecting a sharing user with $\delta = 4$, and the results obtained by our algorithms. From these figures we can see that Twitter database is shaped as small clusters of sub-communities with ego-user in the middle, and most of users community are connected directly to the ego-user. The communities are shaped as star-communities, generally connected by a small amount of edges

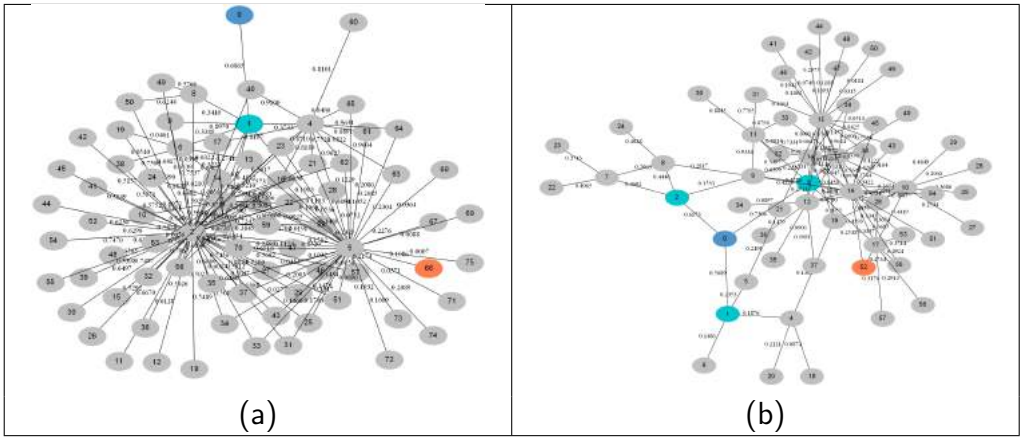

Figure 5.10: Twitter: (a) $V = 75, E = 151, D = 0.0272$ edges (b) $V = 58, E = 120, D = 0.0363$

Figure 5.10 describes two communities derived from Twitter database. The two communities have the same order of vertices and the same order of edges, however we can see a clear division to two main star-clusters of vertices in community (a) while community (b) is shaped as sparse clusters connected together.

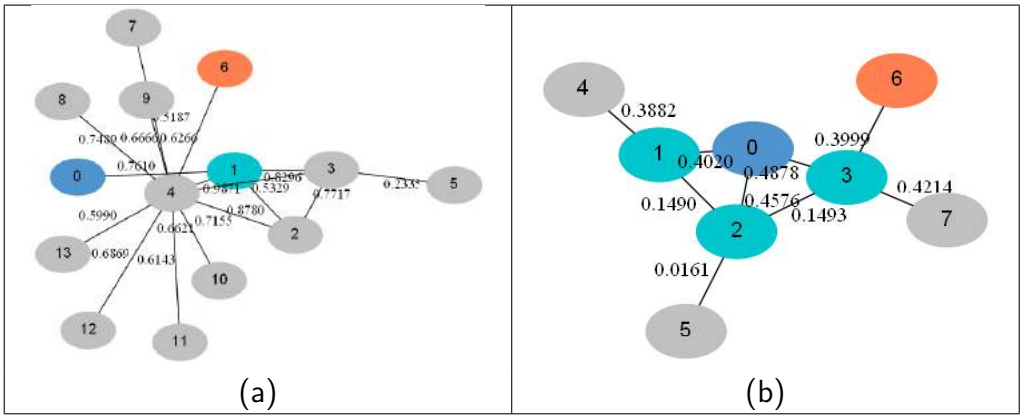

Figure 5.11: Twitter: (a) $V = 15, E = 21, D = 0.1$ (b) $V = 9, E = 9, D = 0.125$

Figure 5.11 describes a second set of two small communities derived from Twitter database. The two communities have the same order of vertices and the same order of edges.

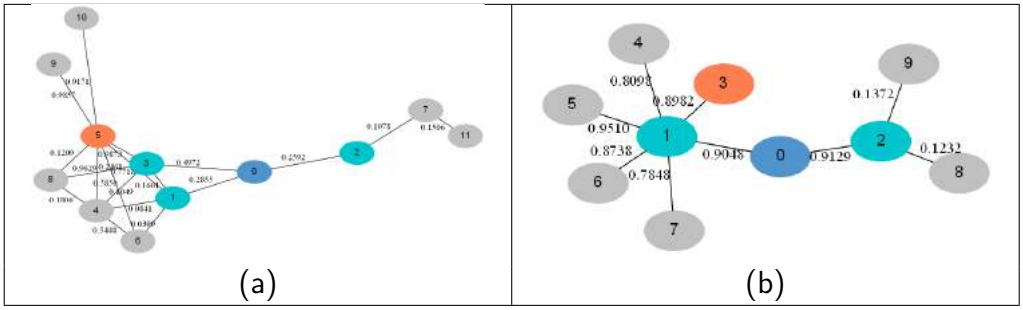

Figure 5.12: Twitter: (a) $V = 13, E = 26, D = 0.16666$ (b) $V = 11, E = 9, D = 0.08181$

Figure 5.12 describes a third set of two small communities derived from Twitter database. The two communities have the same order of vertices while community (a) has a larger number of edges (connections).

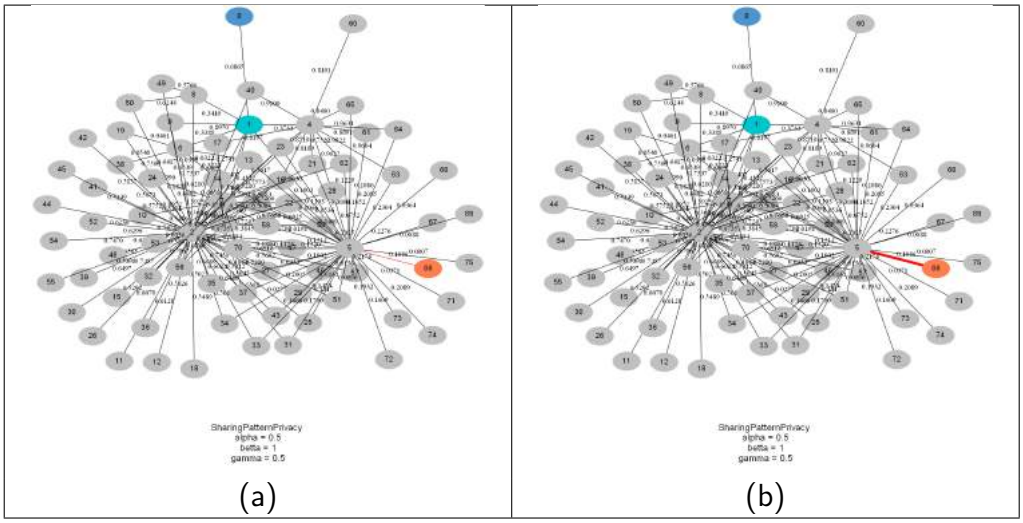

Figure 5.13: Twitter: (a) Initial Min-cut edges (b) Final Min-cut edges

Figure 5.13 is the same sub-community as figure 5.10 (a), with one adversary colored in coral (66). The ego-user is colored in steel blue, and the red lines depict the initial and final candidates-set of edges to be blocked, found by the Min-cut algorithm.

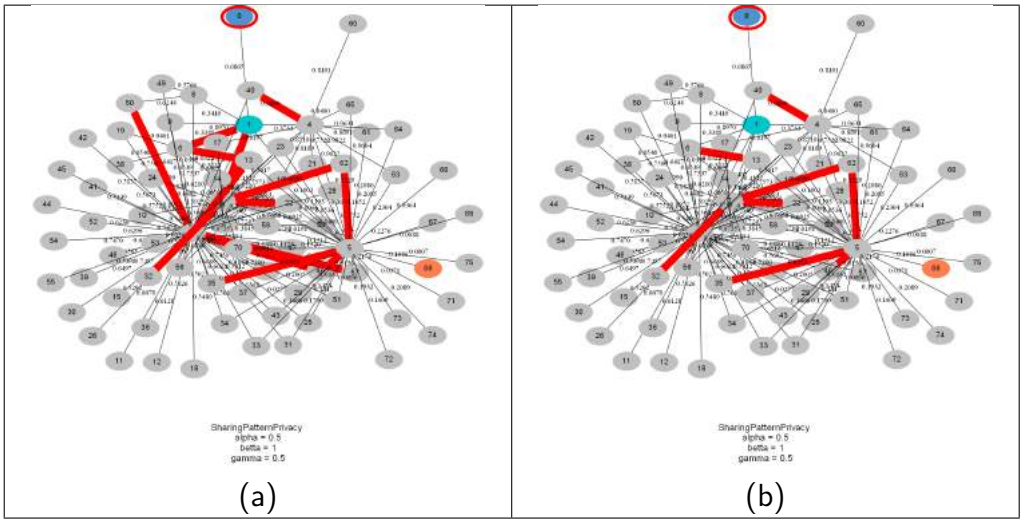

Figure 5.14: Twitter: (a) Initial Contract edges (b) Final Contract edges

Figure 5.14 is the same sub-community as figure 5.10 (a), with one adversary colored in coral (66). The ego-user is marked with red circle, and the red lines depict the initial and final candidates-set of edges to be blocked, found by the contract algorithm.

Tables 5.7- 5.8 summarize the results of six different evaluation runs, for dif-

ferent communities.

Run	Density	δ	Friends	Adversaries	Vertices	Edges
1	0.0272	4	1	1	75	151
2	0.0363	4	3	1	58	120
3	0.1	4	1	1	15	21
4	0.125	4	3	1	9	9
5	0.1666	4	3	1	13	26
6	0.08181	4	2	1	11	9

Table 5.7: Twitter sub-communities Data size

Table 5.7 presents six runs with the different sub-communities derived from Twitter database. The community size is derived from the sharing user (ego node) and friends column refers to the amount of first degree friends of this user. These six runs are used as input to our algorithms and the results are presented next.

Run	α	γ	$Density$	MinCut Initial Edges	MinCut Final Edges	Contract Initial Edges	Contract Final Edges	Remark
1	0.5	0.5	0.0272	1	1	15	9	MinCut blocked edges to adversary, Contract blocked edges to community members
1	0.757	0.4068	0.0272	1	1	4	3	MinCut blocked edges to adversary, Contract blocked edges to community members
1	0.797	0.1729	0.0272	1	1	4	3	MinCut blocked edges to adversary, Contract blocked edges to community members
1	0.2398	0.8356	0.0272	1	1	2	2	MinCut blocked edges to adversary, Contract blocked edges to community members
2	0.5	0.5	0.0363	2	2	4	3	MinCut blocked edges to adversary, Contract blocked mixed edges
2	0.9601	0.3916	0.0363	2	2	7	0	MinCut blocked edges to adversary, Contract did not block edges
2	0.7307	0.3735	0.0363	2	2	6	2	MinCut blocked edges to adversary, Contract blocked mixed edges
2	0.8892	0.6243	0.0363	2	2	1	0	MinCut blocked edges to adversary, Contract did not block edges
3	0.5	0.5	0.1	1	1	4	1	MinCut blocked edges to adversary, Contract blocked edges to community members users frequently share data with the adversary
3	0.39	0.3972	0.1	1	1	1	1	MinCut blocked edges to adversary, Contract blocked edges to community members users frequently share data with the adversary
3	0.8672	0.7831	0.1	1	0	4	0	No edges to be blocked, users shares data with the adversary with high probability high γ value, no need to block edges
4	0.848	0.2021	0.125	1	0	1	0	No edges to be blocked, users rarely share data with adversaries
4	0.1135	0.1099	0.125	1	1	1	1	No edges to be blocked, users rarely share data with adversaries
5	0.5	0.5	0.1666	3	0	4	0	No edges to be blocked, adversary is in the middle of sub-community, users frequently share data with the adversary
5	0.9046	0.7023	0.1666	3	0	4	0	No edges to be blocked, adversary is in the middle of sub-community, users frequently share data with the adversary
5	0.1814	0.7266	0.1666	3	0	2	2	Min-cut did not block edges, adversary is in the middle of sub-community, users frequently share data with the adversary
6	0.5	0.5	0.08181	1	1	1	1	Blocked edges to adversary
6	0.6182	0.2631	0.08181	1	1	1	1	Blocked edges to adversary
6	0.5058	0.9073	0.08181	1	0	5	0	No edges to be blocked, high threshold to the adversary no need to block edges

Table 5.8: Twitter Evaluation Runs Results

Table 5.8 present the results obtained by six runs on Twitter database, with close friends distance, β set to 1.

Columns 2-3 present the threshold parameters used for each run. For each community graph we performed the algorithms with medium thresholds, ($\alpha = 0.5, \gamma = 0.5$), and with random thresholds. Columns 5-6 and 7-8 present the initial and final set of edges to be blocked found by min-cut and contract algorithm respectively. The remark indicates which kind of edges are the candidates for blocking. The results for Twitter database are similar to the results obtained by using Facebook database, we can see that when the adversaries are close to the community's boundary ($\delta = 4$), and no maximum path to a community member passes through an adversary vertex, (e.g. run 1,2 and 4) the solution is trivial and the blocked edges are the edges from community members to the adversaries.

When the adversary is in the middle of the community's boundaries, and there

is a path with maximum information flow to a community member that passes through an adversary vertex (e.g. run 5), it is highly likely that the maximum information flow will be reduced considerably and therefore no good solution can be obtained for high levels of α. When the ego-user rarely shares data with community member, there is no need to block edges, or the solution is the trivial solution (e.g. run 4).

5.2.3 Test On Google+ Database

Figures 5.15- 5.18 present four sub-communities derived from Google+ database by randomly selecting a sharing user with $\delta = 4$, and the results obtained by our algorithms. From these figures we can see that Google+ database is built from small clusters of sub-communities with ego-user in the middle, and most of users community are connected directly to the ego-user.

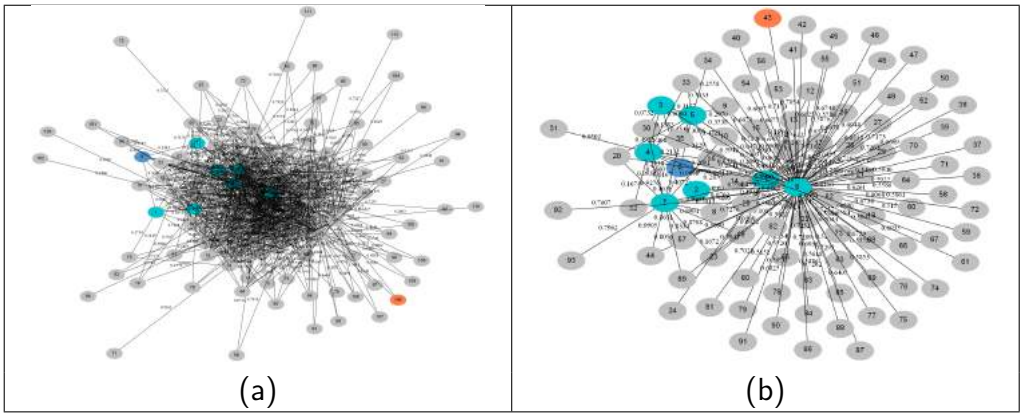

Figure 5.15: Google+: (a) $V = 113, E = 886, D = 0.07$ edges (b) $V = 94, E = 149, 0.1704$

Figure 5.15 describes two sub-communities derived from Google+ database, we can see that Google+ database resembles Facebook database and is built from clusters sub-communities, generally connected by a small amount of edges.

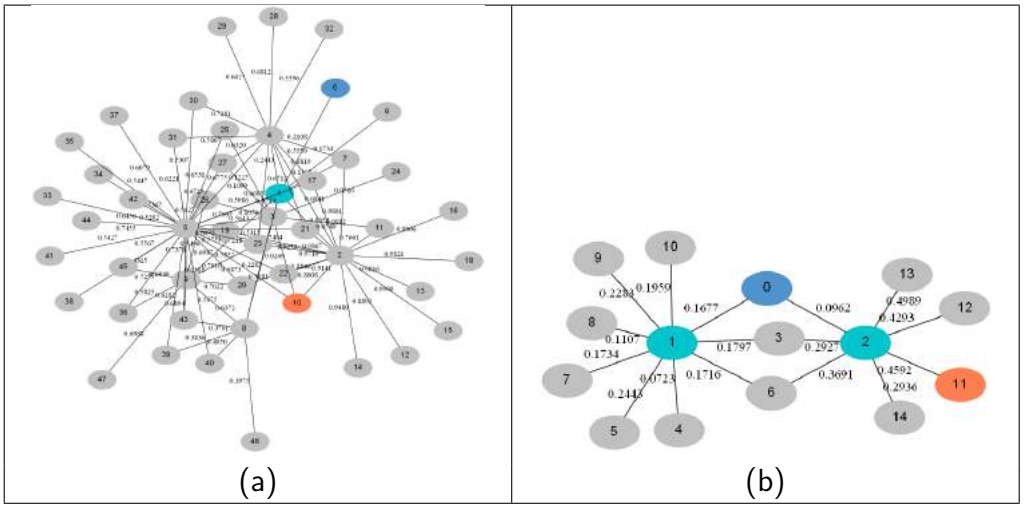

Figure 5.16: Google+: (a) $V = 48, E = 89, D = 0.03945$ (b) $V = 16, E = 16, D = 0.0625$

Figure 5.16 describes a second set of two small sub-communities derived from Google+ database.

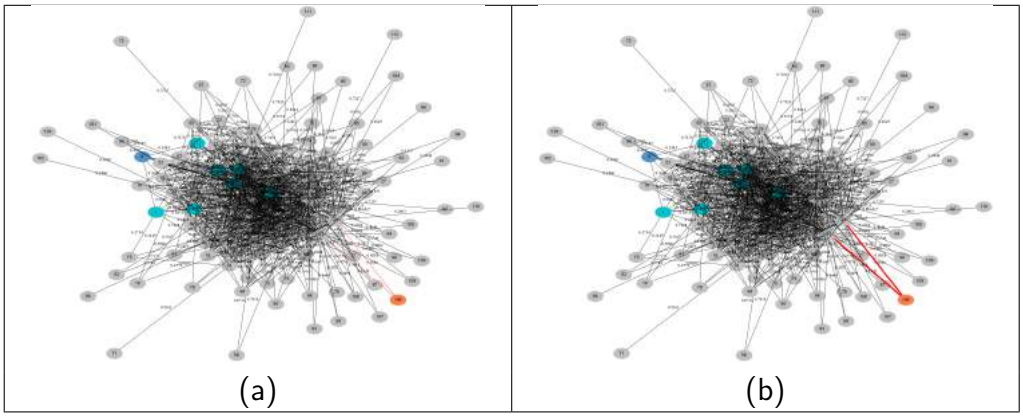

Figure 5.17: Google+: (a) Initial Min-cut edges (b) Final Min-cut edges

Figure 5.17 is the same sub-community as figure 5.15 (a), with one adversary, 106, colored in coral. The ego-user is colored in steel blue, and the red lines depict the initial and final candidates-set of edges to be blocked, found by the Min-cut algorithm.

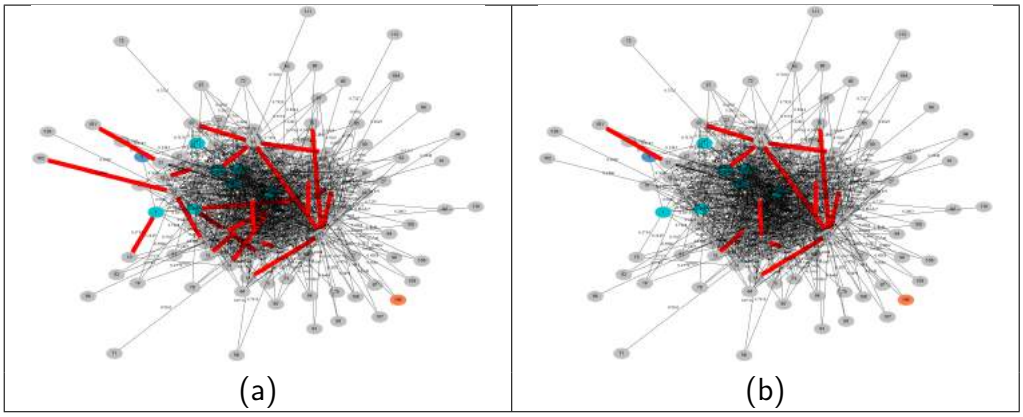

Figure 5.18: Google+: (a) Initial Contract edges (b) Final Contract edges

Figure 5.18 is the same sub-community as figure 5.15 (a), with one adversary, 106, colored in coral. The ego-user is marked with red circle, and the red lines depict the initial and final candidates-set of edges to be blocked, found by the contract algorithm.

Tables 5.9- 5.10 summarize the results of several different evaluation runs, for different communities.

Run	Density	δ	Friends	Adversaries	Vertices	Edges
1	0.07	4	7	1	113	886
2	0.1704	4	7	1	94	149
3	0.03945	4	1	1	48	89
4	0.0625	4	2	1	16	16

Table 5.9: Google+ sub-communities Data size

Table 5.9 presents four runs with the different sub-communities derived from Google+ database. The community size is derived from the sharing user (ego node) and friends column refers to the amount of first degree friends of this user. These four runs are used as input to our algorithms and the results are presented next.

Run	α	γ	$Density$	MinCut Initial Edges	MinCut Final Edges	Contract Initial Edges	Contract Final Edges	Remark
1	0.5	0.5	0.07	2	2	21	12	MinCut blocked edges to adversary, contract blocked edges to community members
1	0.6523	0.1732	0.07	2	2	60	7	MinCut blocked edges to adversary, contract blocked edges to community members
1	0.7732	0.0685	0.07	2	2	89	31	MinCut blocked edges to adversary, contract blocked edges to community members
1	0.7411	0.0402	0.07	2	2	104	26	MinCut blocked edges to adversary, contract blocked edges to community members
2	0.5	0.5	0.1704	1	1	1	1	Blocked edges to adversary
2	0.8077	0.4405	0.1704	1	1	1	1	Blocked edges to adversary
2	0.519	0.8825	0.1704	1	1	1	1	MinCut blocked edges to adversary, contract blocked edges to community members
3	0.5	0.5	0.03945	3	3	3	0	MinCut blocked edges to adversary, contract did not block edges
3	0.5973	0.3756	0.03945	3	3	16	5	MinCut blocked edges to adversary, contract blocked edges to community members
3	0.8614	0.151	0.03945	3	3	6	3	MinCut blocked edges to adversary, contract blocked edges to community members
4	0.5	0.5	0.0625	1	0	1	0	No edges to be blocked users rarely share data with the adversary
4	0.9827	0.0816	0.0625	1	1	1	1	Blocked edges to adversary
4	0.869	0.8929	0.0625	1	1	1	1	Blocked edges to adversary

Table 5.10: Google+ Evaluation Runs Results

Table 5.10 present the results obtained by four runs on Google+ database, with close friends distance, β set to 1.

Columns 2-3 present the threshold parameters used for each run. For each community graph we performed the algorithms with medium thresholds, ($\alpha = 0.5, \gamma = 0.5$), and with random thresholds. Columns 5-6 and 7-8 present the initial and final set of edges to be blocked found by min-cut and contract algorithm respectively. The remark indicates which edges were found as candidates for blocking..

The results for Google+ database are similar to the results obtained by using Facebook database, we can see that when the adversaries are close to the community's boundary ($\delta = 4$), and no maximum path to a community member passes through an adversary vertex, (e.g. run 2) the solution is trivial and the blocked edges are the edges from community members to the adversaries. When the ego-user rarely shares data with community member, there is no need to block edges, or the solution is the trivial solution (e.g. run 4).

5.3 General Discussion

We evaluated our algorithms on various databases with random ego-nodes, community distance, and information sharing probabilities. When the density of $\delta-$community graph for the selected ego-node is high we got better results using the contract method in terms of number of blocked edges. When the min-cut method finds a solution it blocks a larger amount of edges (users) than the

contract method, (e.g., runs $3, 5, 8$ using Facebook database). The results are discussed in terms of efficiency, i.e. CPU time, and Quality, i.e. blocked flow to community members and leakage of flow to adversaries, and the ability to find a solution. In average 54% of the initial candidate set found by min-cut did not lead to a solution that complies with the required privacy criteria, while only 34% in average of the initial candidate set found by contract did not lead to a solution that complies with the required privacy criteria.

While both algorithms are complete, in the non trivial cases, min-cut finds the the best solution in terms of flow ratio between the initial and final flow to community members (e.g. run 3 using Twitter database). Contract, on the other hand, may not find the best solution but returns a compromised solution that is less efficient in blocking adversaries but allows more sharing with friends (e.g. runs $5, 8$ using Facebook database). Executing the contract algorithm multiple times ensures that each time a different set of initial candidate- edges is selected and may result in different final cuts. This way we avoid cases such as 3 using Facebook database, where contract result in no solution.

We run the contract algorithm only 10 times for each set of parameters, thus there are runs where the min-cut algorithm finds a solution while the contract algorithm doesn't. In the case where min-cut does not find a solution, the contract algorithm will provide the best cut that satisfies the threshold.

Figures 5.19- 5.20 presents the CPU time results obtained by our algorithms for the first phase that builds the initial candidate set on sub-communities with different densities as described in 4.4.2. The sub-communities were derived from Facebook database by randomly selecting a sharing user with $\delta = 4$.

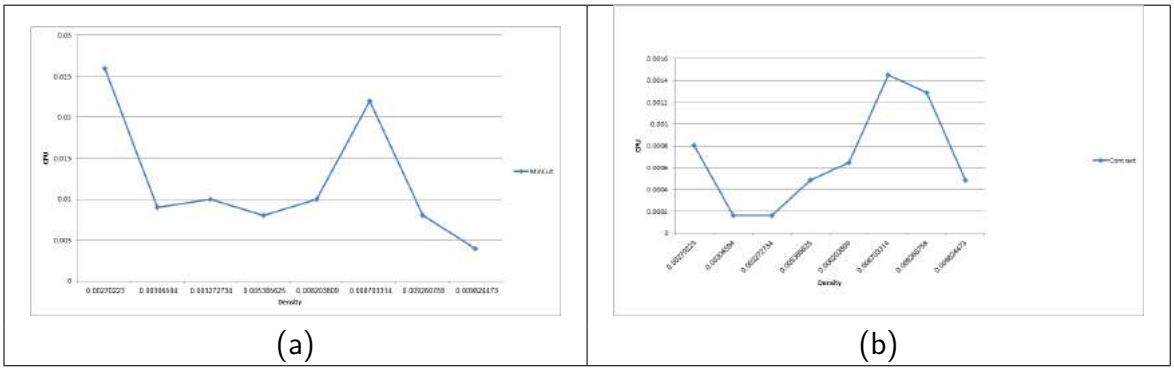

Figure 5.19: Facebook sub-community :(a) Initial Min-Cut CPU (b) Initial Contract CPU

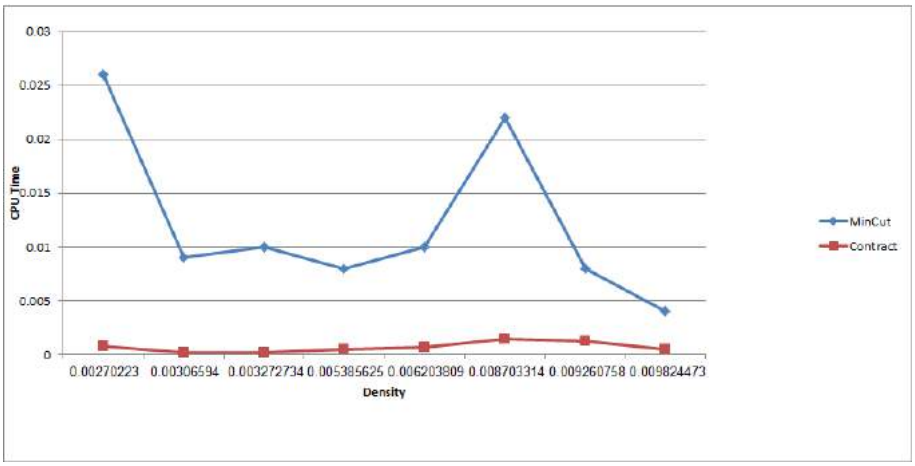

Figure 5.20: Facebook community Min-cut and Contract CPU time

The figures present the CPU time for Min-Cut and Contract methods with $\alpha = 0.5$ and $\gamma = 0.5$. The Contract is always more efficient than the Min-Cut, this is more significant for sub-communities with low densities. The results for different α and γ values remain with similar CPU time ratio as for $\alpha = 0.5$ and $\gamma = 0.5$.

Although we did not optimize our algorithms with respect to CPU time and memory usage, both algorithms run very fast and can be used in real world networks. For example, using a graph with density 0.0635 ,(987 vertices and 61831 edges), the initial contract run time is 34.86 seconds, and min-cut run time is 35.85 seconds. The run time of the final step, the *compute criteria*, varies according to the size of the initial cut: 0.045 seconds for a cut with 35 edges, and 19.79 seconds for a cut with 286 edges. The CPU time for both algorithms on sub-community graphs with low density is very low and almost the same. When running our algorithms on sub-communities with higher density, the Min-cut algorithm CPU time grows faster than the contract algorithm CPU time. The contract CPU time depends not only on the density of the graph but also on the probability of flow on the edges of the graph, thus each run may provide different CPU time that is bounded to $O(|E| \, log \, |E|)$. The contract algorithm can be optimized and paralleled to obtain better results as described in [5]. In some cases, as the graph grow denser, contract deteriorate faster than min-cut. However, in most cases the cuts found by contract lead to solution while the cut found by min-cut does not.

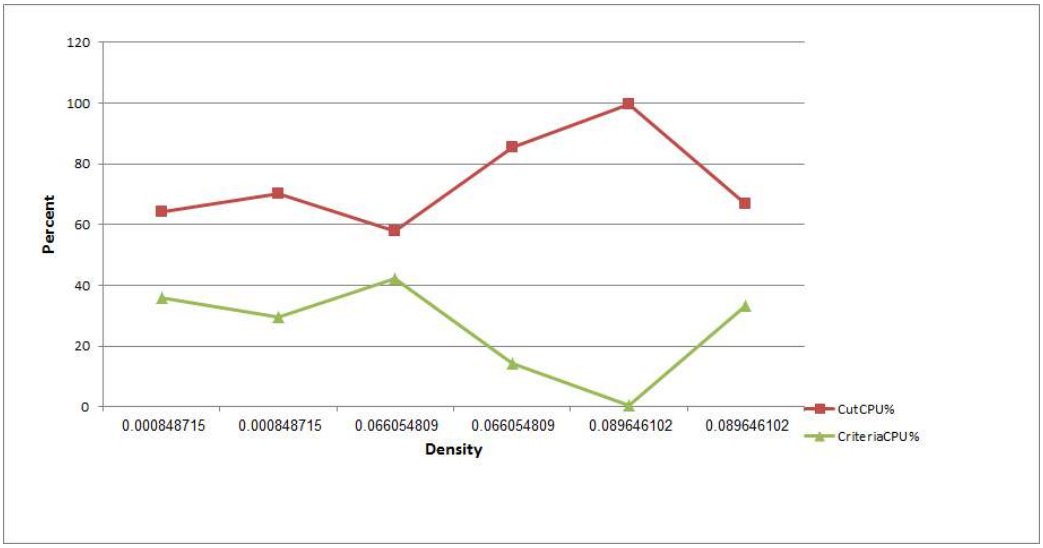

Figure 5.21: %CPU of the two main part of the algorithm

Vertices	Edges	Density	Cut Method	Initial Cut CPU %	Final Cut CPU %	Initial Set	Final Set
787	525	0.0008487	Min-Cut	64.01%	35.99%	2	0
787	525	0.0008487	Contract	70.34%	29.66%	2	0
968	61831	0.0660548	Min-Cut	57.71%	42.29%	39	0
968	61831	0.0660548	Contract	70.34%	29.66%	174	164
205	3749	0.0896461	Min-Cut	66.75%	33.25%	19	16
205	3749	0.0896461	Contract	99.57%	0.43%	4	4

Table 5.11: CPU % Per Phases

Figure 5.21 and table 5.11 demonstrate the relative CPU time of each of the two main parts of the algorithm: find initial candidate set and compute criteria. The runs presented were configured with $\alpha = 0.25$ and $\gamma = 0.25$, however similar results were obtained from all other configurations. The major part of CPU time is required for finding the candidate set of edged. Once a candidate set is found the time required to compute the final cut that meets the criteria, depends mainly on the size of the cut if a solution exists. When no solution exists even for a small candidate set the relative CPU time is large due to the exhaustive search for a solution.

5.4 Complexity

The algorithm we propose is composed of three major steps: the first is the initialization step that creates a multi-graph with a super-vertex s_1 containing

$u_i's$ β-community, the second step finds the candidates-sets for blocked edges, and the last step evaluates the candidates sets of edges and constructs the final set of edges to be blocked.

5.4.1 Initialization Complexity

In the initialization step, the BFS (Breadth-First Search) traversal algorithm is used starting at the ego-node. The time complexity is $O(|V| + |E|)$, since in the worst case, every node and every edge will be explored. $O(|E|)$ may vary between $O(1)$ and $O(|V|^2)$, depending on how sparse the graph is.

5.4.2 Find Candidates Complexity

We use two methods derived from flow problems, to find the initial candidates-set of edges to be blocked. The candidate set is actually a cut between super-vertex s_1 that contains u_i and his close friends, and each of $u_i's$ adversaries. *Min-Cut* which is based on Ford-Fulkerson [19], Max-flow-min-cut algorithm, finds the cut with the minimal flow value, and *Contract* which is based on Karger et al. [5], contract algorithm, finds any cut.

5.4.2.1 Min-Cut Complexity

There are various implementations of the Max-flow-min-cut algorithm, each with different complexity; let $|V|$ be the number of vertices in a graph, $|E|$ is the number of edges in the graph, U is the maximum edge capacity, and F is the maximum flow value, the min-cut complexity depends on the max-flow-min-cut implementation, and is varied from $O(|V|^2 \cdot U)$ [7] to $O(|E| \cdot |V|^{2/3} \cdot \log(\frac{|V|^2}{|E|}) \cdot \log U)$ [3].

We implemented the Edmonds-Karp [13] algorithm for finding the initial candidates set by *Min-Cut*, implying time complexity of $O(|E|^2 |V|)$.

Table 5.12 lists known algorithms for solving the max-flow-min-cut problem and their complexity:

Year	Presented By	Method	Complexity
1950	G. B. Dantzig [7]	Simplex	$O(\|V\|^2 \cdot U)$
1955	L. R. Ford and D. R. Fulkerson [19]	Augmenthing path	$O(F \cdot (\|V\| + \|E\|))$ $O(\|V\| \cdot \|E\| \cdot U)$
1970	J. Edmonds and R. M. Karp [13]	Shortest path	$O(\|E\|^2 \|V\|)$
1970	J. Edmonds and R. M. Karp [13]	Max capacity	$O(\|E\| \cdot \log U \cdot (\|E\| + \|V\| \log \|V\|))$
1970	Yefim Dinic [10]	Shortest path	$O(\|V\|^2 \|E\|)$
1974	A. V. Karzanov [18]	Preflow push	$O(\|V\|^3)$
1986	A. V. Goldberg and R. E. Tarjan [2]	FIFO preflow push	$O(\|E\| \cdot \|V\| \cdot \log(\|V\|^2 / \|E\|))$
1994	V. King, S. Rao, and R. E. Tarjan [33]	Randomized with Deterministic play	$O(\|E\| \cdot \|V\| \cdot \log_{\frac{\|E\|}{\|V\| \log \|V\|}} \|V\|$
1997	A. V. Goldberg and S. Rao [3]	Length function	$O(\|E\|^{3/2} \cdot \log(\frac{\|V\|^2}{\|E\|}) \cdot \log U)$ $O(\|E\| \cdot \|V\|^{2/3} \cdot \log(\frac{\|V\|^2}{\|E\|}) \cdot \log U)$

Table 5.12: Max-Flow-Min-Cut known algorithms

5.4.2.2 Contract Complexity

The contract algorithm is a randomized algorithm that repeatedly contract vertices to super-vertices, until it gets two super-vertices connected by a set of edges that defines a cut between the two sets of vertices contained in each super-vertex. The algorithm randomly selects an edge (u, v) and merges the nodes u and v into one super-vertex, reducing the total number of nodes of the graph by one. All other edges connecting either u or v are re-attached to the merged node producing a multi-graph.

Each iteration of the contract algorithm finds a different cut between the super-vertex containing $u_i's$ $\beta-$community and the super-vertex containing $u_i's$ adversaries.

When using permutations to define the order for selecting edges for contraction, one iteration takes $O(|E| \, log \, |E|)$ [17].

If the algorithm is repeated $O(|V|^2 \log^3 |V|)$ times, it finds the minimum cut in some iteration [5]. The algorithm is strongly polynomial, and can be paralleled to run with $O(|V|^2)$ using $|V|^2$ processors. However, as we learn from the evaluation, most of the time we do not need the minimum cut solution either because it does not comply with the privacy criteria or because the contract provides reasonable solution after a small number of iterations.

We implemented the KargerStein's [5] contract algorithm using an adjacency list representation of the graph, for finding the initial candidates set by contract cut.

5.4.3 Compute Final Candidates Set Complexity

The final step of our algorithm attempts to remove edges from the candidates-set of edges to be blocked, as long as the remaining δ-community graph for user u_i complies with the required privacy criteria. We use BFS to find the maximum flow from u_i to each node in $u_i's$ $\delta-$community, and check if it complies with the required privacy criteria, using the original flow, α and γ thresholds. If it doesn't comply with the required privacy criteria, we try to remove edges from the initial blocked candidates-set, and insert them back into $u_i's$ δ-community graph, until the remaining community graph complies with the required criteria, or until we tested the entire edges in the initial candidate-set, and couldn't find a set of edges to be blocked. In each iteration we insert one edge back to the graph, thus in the worst case we have $|E|$ iterations. The BFS complexity is $O(|V| + |E|)$, and in the worst case we have $|E|$ iterations, thus the total complexity is $O(|V| \cdot |E| + |E|^2)$.
As we show in Appendix A the problem presented is NP-Complete, the overall complexity of our proposed model is $O(|V|^2 + |V| \cdot |E| + |E|^2)$ when using paralleled contract for finding the initial candidates-set for edges to be blocked, or $O(|V| \cdot |E|^2 + |E|^2)$ when using min-cut for finding the initial candidates-set for edges to be blocked.
Table 5.13 summarizes the overall complexity of our algorithms.

Method	Initialization	Find Candidates	Evaluate and Find Final	Overall Complexity
Min-Cut	$O(\|V\| + \|E\|)$	$O(\|E\|^2\|V\|)$	$O(\|V\| \cdot \|E\| + \|E\|^2)$	$O(\|V\| \cdot \|E\|^2 + \|E\|^2)$
Contact - one iteration	$O(\|V\| + \|E\|)$	$O(\|E\| log \|E\|)$	$O(\|V\| \cdot \|E\| + \|E\|^2)$	$O(\|E\| log \|E\| + \|V\| \cdot \|E\| + \|E\|^2)$
Contact - paralleled using $\|V\|^2$ processors	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$	$O(\|V\| \cdot \|E\| + \|E\|^2)$	$O(\|V\|^2 + \|V\| \cdot \|E\| + \|E\|^2)$

Table 5.13: Algorithm's complexity

5.5 Practical Implementation

In order to evaluate our algorithms we developed a prototype application named "SharingPatternPrivacy.exe", described in appendix B.
The "SharingPatternPrivacy.exe" application is able to run on any database saved in a basic graph structure, kept in text files as described in B.3.
Our application main options are:

- **Manipulate a social graph** : add or delete vertices and edges, define, load and save a social graph.

- **Set parameters required by our algorithms** : define which vertex is the ego-node and who are the ego-node's adversaries, set comunity distance, set privacy criteria thresholds.
- **Run our algorithms with different social graphs and privacy criteria** : run the main algorithm "ConstructBlockedEdges" by using the "Evaluate" button that randomly chooses an ego-node, edge's probabilities, and assigns the evaluation parameters, run the algorithm that finds edge-candidates sets, and computes the required privacy criteria to find the final set of edges to be blocked. Run each step of the algorithms separately by using dedicated buttons for each step.

Chapter 6

Conclusion and Future work

The problem of uncontrolled information flow in social network is a true concern to ones privacy. In this paper we address the need to follow the social trend of information sharing while enabling the owner to prevent their information from flowing to undesired recipients. The goal of the suggested method is to find the minimal set of edges that should be excluded from ones community graph to allow sharing of information while blocking adversaries. To reduce side effect of limiting legitimate information flow, we minimize this impact according to the flow probability.

One of the main purposes of our evaluation was to compare the solutions obtained by the different algorithms. Based on our experiments we can conclude that except for cases where adversaries are connected with very few edges, the solutions acquired by Mincut are inferior to those obtained with the Contract algorithm. Furthermore contract is usually more efficient even if run for several iterations.

Our algorithms can be used within the ORIGIN CONTROL access control model [23]. In this model every piece of information is associated with its creator forever. The set of cut edges found by our algorithms, is stored for each user when the data is released and can be checked when the origin controlled information is accessed. This way the administrator can check whenever this information is accessed by a certain user, if the edge between them was cut for the originator user, and thus prevent the information to pass through that edge to that certain user.

6.1 Optimizations

In the last part of our algorithm we check if by removing all initial-candidates-set of edges from $u_i's$ δ-community graph, the remaining δ-community graph of user

u_i complies with the required privacy criteria. If not, we try to insert the removed edges back into $u_i's$ δ-community graph, until the remaining community graph complies with the required criteria, or until we tested the entire edges in the initial candidate-set, and couldn't find a set of edges to be blocked. In each iteration we insert back one edge to the graph, and compute the maximum flow from u_i to all members in $u_i's$ δ-community. In the worst case we have $|E|$ iterations, each computes the maximum flow from u_i to all members in $u_i's$ δ-community. Instead of computing the shortest path each time, we can examine methods for building accumulative paths to calculate the best path from u_i to all members in $u_i's$ δ-community to improve the average computation time.

6.2 Other Extensions

In this work we used two approaches to identify the set of edges to be blocked, the max-flow-min-cut method, and the contract method that finds any cut between two sets in a graph.
In future work these algorithms can be extended in several ways. One approach is to use k-shortest-paths as the source for edges to be blocked. In this method one can start by finding the k-shortest path from the ego-node (source) to the adversary (sink), and set the edges on the first shortest-path as candidates for blocking. If the required privacy criteria is not achievable, the second shortest path is set as the initial candidates set, etc. Another approach is to set the combination of all k-shortest paths from the ego-node to the adversaries as the initial candidates set of edges for blocking. Node centrality can be use to select edges from central nodes to adversaries as the initial candidates set
Another challenge is to automatically identify $u_i's$ adversaries, by fields of interest, joined groups, posts or pre-defined characteristics.
Finally, this model of controlling flow needs to be integrated not only within the ORIGIN CONTROL access control model [23], but with other models of privacy in social networks such as the models described in [21].

6.3 Practical Aspects

6.3.1 Access Rules

Most access control models define access rules in terms of the degree of relationship required to access the shared data. For example, Facebook *restricted lists* enables Facebook users to define a list of undesired recipients from their direct friends, to whom the shared data will be blocked. Google+ *cycles* enables

Google+ users to define communities (cycles) from their direct friends, for information sharing. The definition of these lists or cycles are applicable only to friends and not to friends of friends. Moreover, these definitions are applied to the entire shared data, and not a specific post. The algorithms we present in this thesis allow the user to define, for each shared information, the users to block and the desired privacy level. Accordingly we dynamically create lists of blocked edges through which the user's shared data does not flow. A user can define with whom he would like to share the entire shared information, what would be the maximum percentage of data he is willing to share with undesired recipients (adversaries), and what would be the minimum percentage amount he is willing to avoid from his community acquaintances, in order to achieve maximum privacy level. The blocked edges restrict data from any member of the social network, not only from friends. Our algorithms can be combined with current access rules. For example Google+ can use our algorithms for defining cycles. The adversaries can be defined per subject, and our algorithms will automatically define the cycles. The cycles can be defined for different distances: 1,2,3, etc. The amount of data that is blocked to friends, and the amount of data that is leaked to adversaries is computed when trying to define the required privacy criteria, this data can be displayed to the user that can decide if it is enough or he would like to change the privacy criteria parameters defined in section 4.1.1. It is important to understand that an edge is cut always with respect to a specific ego user and a specific data content. It is never removed from the actual graph, thus information from other ego users can still flow through that edge.

Bibliography

[1] A. Ranjbar, M. M. (2014). Using community structure to control information sharing in online social networks. *Computer Communications 41*, 11–21.

[2] A. V. Goldberg, R. E. T. (1988). A new approach to the maximum flow problem. *Journal of the ACM 35*, 921–940.

[3] A. V. Goldberg, S. R. (1997). Length functions for flow computations. Technical report, NEC Research Institute.

[4] Barabasi, A. L. (2011). *Introduction and Keynote to A Networked Self*, Chapter Introduction, pp. 14. New York, NY: Routledge, Taylor and Francis Group.

[5] D. R. Karger, C. S. (1996). A new approach to the minimum cut problem. *Journal of the ACM 43 (4)*, 601–604.

[6] D. Vatsalan, P. Christen, V. S. V. (2013). A taxonomy of privacy-preserving record linkage techniques. *Information Systems 38*, 946–969.

[7] Dantzig, G. B. (1951). Applications of the simplex method to a transportation problem. In I. T. C. Koopmans (Ed.), *Activity analysis and production and allocation*, New York, pp. 359–373. Wiley.

[8] D.F. Ferraiolo, D. K. (1992). Role-based access control. In *In Proceedings of the 15th National Computer Security Conference*, pp. 554–563.

[9] Dijck, J. V. (2013). *The Culture of Connectivity : A Critical History of Social Media*, Chapter 1, pp. 240. Oxford: Oxford University Press.

[10] Dinic, E. (1970). Algorithm for solution of a problem of maximum flow in a network with power estimation. *Doklady Akademii nauk 11*, 1277–1280.

[11] F. Carmagnola, F. Osborne, I. T. (2014). Escaping the big brother: An empirical study on factors influencing identification and information leakage on the web. *Journal of Information Science 40(2)*, 180–197.

[12] Feige., U. (1998). A threshold of ln n for approximating set cover. *Journal of the ACM 45(4)*, 634–652.

[13] J. Edmonds, R. M. K. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM 19(2)*, 248–264.

[14] J. Kleinberg, K. L. (2013). Information-sharing in social networks. *Games and Economic Behavior 82*, 702–716.

[15] J. Leskovec, A. K. (2014, June). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

[16] K. Yoojung, S. Dongyoung, M. C. S. (2011). Cultural difference in motivations for using social network sites: A comparative study of american and korean college students. *Computers in Human Behavior 27*, 365–372.

[17] Karger, D. R. (1993). Global min-cuts in rnc, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, New York, pp. 21–30. ACM.

[18] Karzanov, A. V. (1974). Determining the maximal flow in a network by the method of preflows. *Soviet Mathematics-Doklady 15*, 434–437.

[19] L. R. Ford, D. R. F. (1956). Maximal flow through a network. *Canadian Journal of Mathematics 8*, 399–404.

[20] M.R. Garey, D. J. (1990). *Computers and Intractability*, Chapter 3, pp. 64. New York, NY: W.H. Freeman & Co.

[21] Nadin Kokciyan, P. Y. (2016). Priguard : A semantic approach to detect privacy violations in online social networks. *IEEE Trans. Knowl. Data Eng. 28(10)*, 2724–2737.

[22] on Privacy Rights Clearinghouse, P. (2016, December). Social networking privacy: How to be safe, secure and social. `https://www.privacyrights.org`.

[23] P. Jaehong, S. R. (2002). Originator control in usage control. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, Washington DC, pp. 60–66.

[24] Papacharissi, Z. (2011). *A Networked Self:*, Chapter 14, pp. 337. New York, NY: Routledge, Taylor and Francis Group.

[25] Pierangela Samaratiy, L. S. (1998). Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. *Technical Report SRI-CSL-98-04, SRI International*.

[26] R. K. Ahuja, T. L. Magnanti, J. B. O. (1993). *Network Flows*, Chapter 6, pp. 863. New Jersey: Prentice Hall.

[27] R. LaRose, K. Junghyun, W. P. (2011). *Social Networking Addictive, Compulsive, Problematic, or Just Another Media Habit?*, Chapter Introduction, pp. 24. New York, NY: Routledge, Taylor and Francis Group.

[28] R. Raz, S. S. (1997). A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *IN PROC. 29TH ACM SYMP. ON THEORY OF COMPUTING, 475-484. EL PASO*.

[29] Rob Hall, S. E. F. (2010). Privacy-preserving record linkage. In *in: Privacy in Statistical Databases,*, Washington DC, pp. 269–283. Springer Lecture Notes in Computer Science.

[30] Search, P. (2007, June). People search engine and meta search. `https://www.123people.com`.

[31] T. H. Cormen, C. E. Leiserson, R. L. R. (1990 (2009)). *Introduction to Algorithms*, Chapter 27, pp. 581. Cambridge, Massachusetts: MIT Press.

[32] T. Tassa, D. J. C. (Feb2013). Anonymization of centralized and distributed social networks by sequential clustering. *Proceedings of the IEEE Transactions on Knowledge & Data Engineering 25 Issue 2*, 311–324.

[33] V. King, S. Rao, R. E. T. (1994). A faster deterministic maximum flow algorithm. *Journal of Algorithms 17*, 447–474.

[34] Yehuda Lindell, B. P. (2009). Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality 1*, 5.

Appendices

B Prototype Implementation

We have built a prototype application named "SharingPatternPrivacy.exe" that enables a user to test our algorithms on different social graphs with different privacy criteria thresholds, manipulate a social graph, and display the results. It is a simple Windows application developed in C++ and MFC. The application is able to run on any database saved in a basic internal graph structure, kept in text files as described in B.3.

B.1 Description and User Interface

Figure 1 is the main screen of the prototype application which is divided into four parts:

- **Left part that enables the user to manipulate the social graph** : add or delete vertices and edges, define, load and save a social graph, load SNAP databases and save them in the application internal format described in B.3.
- **The second part enables the user to edit and show the evaluation parameters** : define the ego-users (source), α, β, and γ, and maximum community distance.
- **The third part enables the user to run our algorithms** : use the "Evaluate" button that randomly chooses the ego-node, assigns the evaluation parameters, and perform the algorithms, or run each algorithm step by step by using dedicated buttons for every step of the algorithms, and evaluate the intermediate results which are kept in text files.

Figure 1: SharingPatternPrivacy main screen

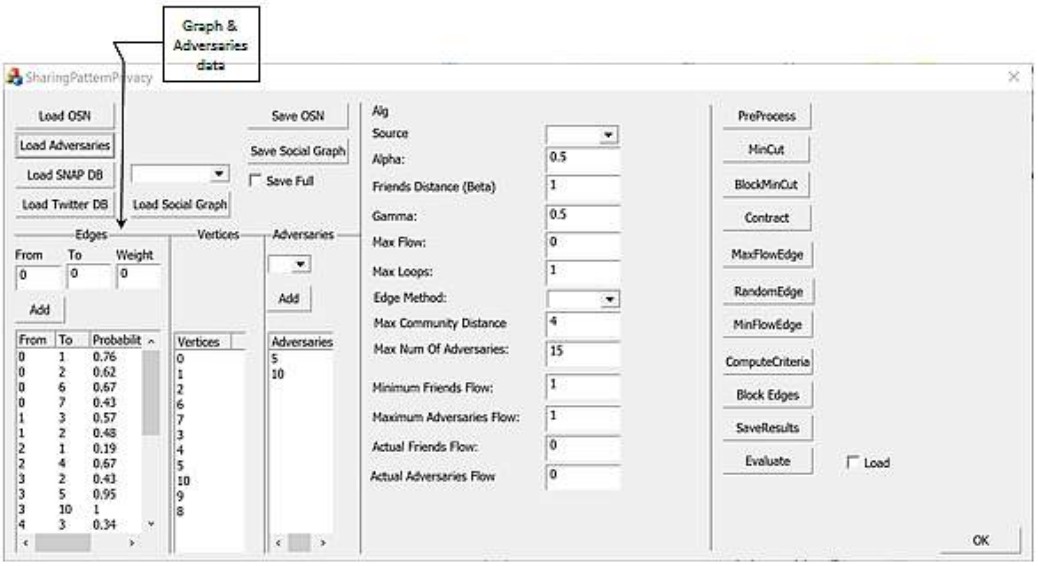

Figure 2: SharingPatternPrivacy after loading a social graph and adversaries' list

B.2 High Level Design

A class named "GraphAdjList" implements our algorithms, it uses a graph represented as adjacency list, each vertex in the graph contains the vertex data, like vertex id, flow values, and a list of neighbors (edge id, destination vertex). Each edge contains the edge data like edge id, edge probability and from vertex id , to vertex id. The graph adjacency list contains vertices map which holds the vertices data, edges map which holds the edges data, adversaries map which holds the adversaries id list, private function used by the algorithms, and public functions for each user operation. A user interface MFC class named "SharingPatterPrivacyDlg" uses the "GraphAdjList" to implement the user interface functions.

B.3 Input Format

The application uses a basic graph structure, kept in text files, every database placed in a separate directory. The main file in each directory is "Vertices.txt" which contains the number of vertices in the social graph. For each vertex there is a file named "VertexId.follower" which holds the vertex followers along with their edge's probability. The ".followers" file has one line per follower id, each line with two numbers, the followers' vertex id and its edge probability.
For Example, for a social graph containing 482 vertices, "Vertices.txt" will contain the number "482", and the social graph directory will include 482 files, a file for each vertex. The file names will be "0.followers", "1.followers", ..., "481.followers".
Assuming vertex id 0 has an edge to vertices 403, 334, 372, 436, 416, 428, and 439, the content of file "0.followers" will be:
403,0.988731
334,0.764183
372,0.790628
436,0.867725
416,0.985557
428,0.930807
439,0.794435

Our algorithms can be used on any type of database converted to that structure.

תקציר

בעבודה זו אנו חוקרים את התנהגות המשתמשים ברשתות חברתיות מקוונות (OSN), כאמצעי לשמירה על פרטיותם. אנשים מרבים להשתמש ברשתות חברתיות מקוונות עבור מגוון רחב של מטרות ותחומים, עדכון הפרופיל שלהם, אינטראקציות חברתיות או מקצועיות, שיתוף מדיה דיגיטלית כגון וידאו ותמונות או כדי להגיב על מידע אשר שותף על ידי חברים. לכל רשת חברתית מקוונת יש מאפיינים שונים, דרישות שונות ונקודות תורפה שונות של המידע הפרטי אשר שותף. ***הרגלי שיתוף*** מתייחסים לדפוסי שיתוף המידע של המשתמשים. הרגלי שיתוף-אלה המשתמעים מהתקשורת בין המשתמשים לבין עמיתיהם, מסתירים בנוסף לתקשורת עצמה, מידע פרטי רב אשר נועד להישאר חסוי. רוב המשתמשים אינם מודעים לכך כי המידע הפרטי הרגיש אותו הם חולקים עלול לדלוף למשתמשים בלתי מורשים.

רוב המודלים לבקרת גישה לנתונים, מגדירים כללי גישה המבוססים על מידת הזיקה הנדרשת כדי לקבל גישה לנתונים אלה. כללים אלה אינם מעודנים מספיק בכדי לאפשר מניעת תוכן באופן דינמי מעמיתים מסוימים של הקהילה.

בעבודה זו אנו מתארים את הרשת החברתית המקוונת כגרף אשר בו הקדקודים מייצגים את המשתמשים, עמיתיהם ויריביהם, והקשתות המחברות בין הקדקודים מייצגות את היחסים בין המשתמשים ברשת החברתית. אנו מסתכלים על קהילת החברים והמכרים של משתמש במרחק מוגדר מראש, ומאפשרים למשתמש להגדיר את רמת הפרטיות הנדרשת עבור כל מידע אותו ברצונו לשתף, בכדי להשיג רמת פרטיות מרבית. המשתמש יכול להגדיר עם מי הוא רוצה לחלוק את המידע המשותף, מה יהיה שיעור הנתונים המזערי אותו הוא מוכן למנוע ממכריו ומה יהיה אחוז הנתונים המקסימאלי אותו הוא מוכן לחלוק עם יריביו.

אנו משתמשים בכמה אסטרטגיות ידועות מתורת הזרימה בגרפים ומהרגלי שיתוף המידע של המשתמשים ברשתות חברתיות מקוונות, בכדי להגדיר אלגוריתמים יעילים וקלים ליישום, המאפשרים הבטחת פרטיות המידע של המשתמשים יחד עם רמת פרטיות מוגדרת מראש.

כדי להעריך את ביצועי האלגוריתמים הרצנו אותם על מספר בסיסי נתונים אמיתיים של רשתות חברתיות והראינו את האפקטיביות שלהם בשימוש בפרמטרים שונים.

תוכן עניינים

תודות

ברצוני להביע את תודתי העמוקה למנחי לתזה, פרופסור אהוד גודס ודוקטור נורית גלעוז, על עזרתם ותמיכתם. אהוד ונורית סיפקו לי עידוד, הכוונה, עצות מעשיות והערות עם כמות מדהימה של חזון, מסירות וסבלנות, לאורך כל התהליך של כתיבת תזה זו.
למדתי רבות משניהם ואני חבה להם הכרת תודה עצומה. הבעיה התיאורטית הועשרה בעזרתו פרופסור זאב נוטוב, ובאמצעות הידע הנרחב שלו.
יותר מכל אני חייבת להודות למשפחתי. לבעלי האוהב איציק על עידודו, אמונתו ותמיכתו מרגע ההתחלה. לילדי הילה, אפרת, אסף ונדב, על סבלנותם ואהבתם בזמן עבודתי על תזה זו.

האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב

אבטחת פרטיות מבוססת דפוסי שיתוף ברשתות חברתיות

עבודת תזה (עבודה מסכמת) זו הוגשה כחלק מהדרישות לקבלת תואר
"מוסמך למדעים" .M.Sc במדעי המחשב
באוניברסיטה הפתוחה
החטיבה למדעי המחשב

על-ידי
לוי סילביה

העבודה הוכנה בהדרכתם של
פרופ' אהוד גודס, האוניברסיטה הפתוחה ואוניברסיטת בן גוריון, ישראל
דר' נורית גלעוז, מכללת ספיר, שדרות, ישראל

11 מאי, 2018