PRIVACY PRESERVATION IN SOCIAL GRAPHS

APPROVED BY SUPERVISING COMMITTEE:

Weining Zhang, Ph.D., Chair

Shouhuai Xu, Ph.D.

Tom Bylander, Ph.D.

Jianhua Ruan, Ph.D.

Ram Krishnan, Ph.D.

Accepted:

Dean, Graduate School

DEDICATION

I lovingly dedicate this dissertation to my family, particularly to my husband, Hongwei, who patiently supports me all the way. I must also thank my parents and in laws for their understanding and encouragement.

PRIVACY PRESERVATION IN SOCIAL GRAPHS

by

LIJIE ZHANG, M.Sc.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
May 2012

ACKNOWLEDGEMENTS

I would never have been able to complete my dissertation without the guidance of my committee members and support from the Department of Computer Science at University of Texas at San Antonio (UTSA).

I would like to express my deepest gratitude to my advisor, Dr. Weining Zhang, for the days we spent probing different aspects of research issues. His enthusiasm, insights, patience, and painstakingness inspire me to overcome the difficulties during the work. Also, I must thank Dr. Shouhuai Xu whose wide range of interests and expertise adds important perspective to my work. Additionally, I would like to thank other committee members Dr. Tom Bylander, Dr. Jianhua Ruan, and Dr. Ram Krishnan for their very helpful insights, comments and suggestions on my dissertation. The work is financially supported by the Department of Computer Science at UTSA. I appreciate all the faculty members and staffs who ever helped me.

I acknowledge the technical support of the High Performance Computing Center/Computational Biology Initiative at University of Texas Health Science Center at San Antonio (UTHSCSA) and the University of Texas at San Antonio.

May 2012

PRIVACY PRESERVATION IN SOCIAL GRAPHS

Lijie Zhang, Ph.D.
The University of Texas at San Antonio, 2012

Supervising Professor: Weining Zhang, Ph.D.

Hundreds of millions of people use social network sites daily for entertainment, socialization, and business purposes. Social network sites have accumulated huge amount of personal information, which can be modeled by social graphs, where vertices represent persons, and edges represent relationships. Social graphs have attracted tremendous interest of scholars and application developers in many areas. However, varieties of sensitive personal information become a privacy concern for social network users and owners, and prohibit publishing the graphs for massive usage. Diverse privacy attacks cause privacy disclosure in social graphs. We categorize the privacy attacks into two categories - vertex re-identification attacks and information re-association attacks.

For vertex re-identification attacks, many early researches propose anonymization methods to protect the identity of vertices so that private information on vertices is preserved. Nevertheless, sensitive relationships represented by edges may still disclose. Our work focuses on design anonymization methods to protect the sensitive edges. We deal with three issues in the method design: privacy measure, utility loss measure and performance. Our contribution includes using a strong equivalence relation to define the privacy measure, choosing the number of edges changed as utility loss measure in a theoretic manner, and developing efficient methods based on a condensed graph for the large scale graph.

For information re-association attacks, recent researches have designed attack models based on various techniques, such as statistics models, data mining techniques, or security attacks. We design a new information re-association attack that combines web search and information extraction techniques. Our contribution includes proposing a measurement to evaluate the effectiveness of the attack, and empirically analyzing the privacy disclosure under this attack.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

1.1 Social Graphs

In recent years, the prevalence of online social network attracts hundreds of millions of users all of the world. Millions and millions of people are using social networks every day, such as Facebook, LinkedIn, Twitter, and MySpace[1], to keep in touch with friends, relatives, and business or professional contacts, and to share information about their lives and works. Over the years, these social network websites have accumulated a vast amount of personal information ranging from names to locations, from education to employment, and from shopping habit to music taste.

Social network data can be modeled as social graphs, where vertices represent individuals and edges represent relationships among individuals. Besides vertices and edges, additional information about individuals and relationships can be represented by labels. Vertex label can represent attributes about a single person, such as gender, location, and education, and edge label can represent information about relationships, such as types of relationships, e.g., friendship, kinship, and co-authorship, and weights of relationships, e.g. trustworthiness, frequency of instant messaging or email communications.

Social network has been studied for a long time in many research areas, including sociology [24], anthropology [50], biology [13], epidemiology [55], and criminology [31]. However, the past social network research is restricted by the small scale of social graphs. The availability of large scale social graphs from online social networks drives new business applications and researches, e.g. viral advertising [34], personal influence scoring[2], social networks evolution [32,35], communication patterns in social networks [5], and so on. Thus, publication of social network data is important. It can bring significant social, economic, and political benefit to organizations, government, and the society at large. The owner of social network data can publish large scale data via FTP or file sharing server to the third party application developers, or place the data on a website

[1] http://www.linkedin.com, http://twitter.com, and http://www.myspace.com

[2] Got Twitter? You've Been Scored: http://www.nytimes. com/2011/06/26/sunday-review/26rosenbloom.html

so that people can view data via web browsers.

In addition to online social networks, a social graph can also represent data from offline network sources of personal information, such as hospitals, telephone companies, law enforcement and other government agencies.

1.2 Privacy

In general, privacy is the ability of an individual or group to seclude themselves or information about themselves and thereby reveal themselves selectively. The right not to be subjected to unsanctioned invasion of privacy by the government, corporations or individuals is part of many countries privacy laws, and in some cases, constitutions. For example, in United States, the Health Insurance Portability and Accountability Act of 1996 (HIPAA) [3] provides federal protections for personal health information held by covered entities and gives patients an array of rights with respect to that information; In Europe, the Data Protection Directive (officially Directive 95/46/EC[4] on the protection of individuals with regard to the processing of personal data and on the free movement of such data) is a European Union directive which regulates the processing of personal data within the European Union.

As computer and network technology develop rapidly, electronic data has become the main format of personal information storage and transmission. The need for personal electronic data protection techniques has been widely recognized. Privacy research in academia and industry has over a decade with spectrum range from health care service [43] to transportation service [18], and from wireless network [28] to smart grid [47].

Many techniques can threaten privacy of personal information. Security attacks [57] can break into personal computer, websites or network to steal personal information. Even if personal information is well protected against security attacks, when large scale data are legally published, private information could be breached via data mining techniques [29], graph comparison [27], or

[3] http://www.hhs.gov/ocr/privacy/hipaa/understanding/index.html

[4] http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:EN:HTML

web crawlers [45].

1.3 Privacy Breach in Social Graphs

As mentioned before, publishing social graphs is important for business applications and researches. However, social graphs contain a lot of sensitive personal information. Social network site owners are responsible for abiding the regulations about privacy protection. Thus, privacy issues become a major concern for both social network site users and owners.

To address the privacy concern, social network sites have established certain privacy policies and provided mechanisms[5] to let members control the visibility of their sensitive personal information. In a social graph, a piece of personal information can be designated sensitive by the person or by a system-wide policy. In general, whether a piece of information is sensitive depends on the member who owes the information, and can vary from person to person. For example, on Facebook, a member can designate his profile or past posts sensitive, and set privacy options to allow only friends to view his profile, or his past posts on his wall; on Twitter, a member can designate his tweets sensitive, and allow authorized people to read his tweets.

A member's privacy in a social graph is breached if an adversary obtains or derives with 100% of certainty a piece of sensitive information that the person wants to hide. For example, Bill from New York city is a member of an online social network. He regards his current location sensitive, and sets his account not to release his current city to anyone. As the online social network publishes the social graph, if an adversary is able to derive that Bill's current location is New York city, we say that Bill's privacy is breached from the social graph.

Privacy preservation mechanisms provided by social network sites often fall short for protecting user's privacy against privacy attacks. Many researches [7, 23, 27, 37, 45, 57, 68] study the privacy attacks on social graphs. We can group privacy attacks into two categories depending on whether the published social graph has identities on vertices or not.

The first category of privacy attacks is called vertex re-identification attack. In this attack,

[5] http://www.facebook.com/policy.php?ref=pf, http://www.myspace.com/index.cfm?fuseaction=misc.privacy

the published social graph has no identities on vertices, simply named as anonymized graph. In an anonymized graph, sensitive information can be associated with vertices and edges. Studies show that an adversary can use various background knowledge about victims to re-identify vertices from the anonymized graph, causing the sensitive information exposed. The adversary may employ graph structure knowledge [27,37,45,68] about victims to define some sub-graph patterns, and re-identify vertices or edges of the victims by searching for those sub-graph patterns in the anonymized graph. The adversary may also use attribute information about vertices, e.g. member's group membership [57] in social network site, to re-identify vertices. This type of attacks usually take place when an adversary can get the anonymized graph from a third-party application developer or research organization to whom social network sites publish the anonymized graph.

The second category of privacy attacks is called information re-association attack. This attack is launched on the other type of published graph that has identities on vertices, but no sensitive information associated with vertices or edges. An adversary in this attack could employ various existing techniques. Some techniques exploit privacy policies [19, 23, 62] about access control in social network sites to expose hidden information. Other techniques [29, 67] use data mining or machine learning to discover the hidden sensitive information. We will review the literature about the two categories of privacy attacks in Chapter 2.

1.4 Research Issues

1.4.1 Privacy Preservation against Vertex Re-identification Attack

We mainly discuss three common issues in the privacy protection methods against vertex re-identification attack.

Privacy Measurement. Recent studies propose *vertex anonymization* methods [7, 12, 27, 37, 68, 69] to protect vertex identities against vertex re-identification attack. However, there is no unified privacy measurement to evaluate the protection degree of these methods. Most privacy measurements employed in these methods are inspired by the k-anonymity concept [49]. The k-

anonymity privacy measure is popularly used in tabular data anonymization methods [1–3, 33, 54]. As applied to the graph data, the k-anonymity privacy measure varies in terms of graph features the adversary may use to re-identify vertices. The graph features could be vertex labels, vertex degree, neighborhood, hub connections and so on. When an adversary launches vertex re-identification attack, she can use one type of graph feature to partition the graph into vertex equivalence classes (VECs), so that vertices in each VEC have equivalent feature. The adversary re-identifies the victim successfully if a VEC has only one vertex with the feature equivalent to the victim. A graph satisfies the k-anonymity privacy requirement if each vertex in the graph has equivalent features with at least $k-1$ other vertices.

Utility Loss of Privacy Preservation Methods. Altering graph structures [27, 37, 68], and vertices [66] or edge labels [38]are commonly techniques used by privacy preservation methods. Consequently, anonymized graphs will have less information, and therefore less utility, than original graphs. Many existing methods [63, 65, 68, 69] use the number of edges that are changed (i.e., added/removed/switched) during the anonymization as an estimate of utility loss, and use heuristics to minimize it. However, other utility measures [16], such as degree distributions, cluster coefficients, length of shortest paths, and connectivity, are more closely related to graph applications in the real world than the number of changed edges. The relationship between the number changed edges and those frequently used utility measures is not studied clearly.

Performance of Privacy Preservation Methods on Large Graphs. Many existing anonymization methods typically perform a sequence of graph changes. For example, Hay et al. [27] use simulate annealing to search for a published graph, such that the graph satisfies k-anonymity privacy requirement, and maximizes the likelihood function that measures how well the anonymized graph fits the original graph. Zhou and Pei [68] add/delete edges to find a k-anonymity graph; Ying and Wu [63] develop a random adding/removing edges method to protect sensitive edges against re-identification. In a graph, even adding or removing one edge could cause a rippling effect on privacy measurement or utility loss. The rippling effect is difficult to predict using present graph theories. Thus, many methods need to check the graph privacy [68] or calculate utility loss [27]

after each graph change, namely change-and-check operations. For real-world large social graphs that contain millions of vertices and billions of edges, measuring privacy or computing utility loss is a very time consuming step. Since a large graph may need to add or remove many edges, existing methods can be very inefficient in both execution time and space usage.

1.4.2 Privacy Preservation against Information Re-association Attack

Information Re-association Attacks. In a vertex re-identification attack, the sensitive information can be easily obtained if vertices are re-identified. Compared to vertex re-identification attack, the process of breaching privacy in information re-association attack is more complicated due to the absence of sensitive information in the published graph. Various techniques are employed in the information re-association attack. For example, data mining techniques [67] are applied to discover a person's hidden interests or tastes via his friends public interests or tastes; Security attacks, e.g. Sybil attack or browsing history stealing, break into social network site system or personal computers to breach private information. A variety of data mining or other techniques can bring up the new information re-association attacks.

Personal information present in many places other than the published social graph is another reason that new information re-association attacks can exist. We make three observations about the source of personal information. First, many people join multiple social networks, thus their personal information on these social networks may overlap. Second, the privacy control on different social network websites can be set different by the same person. For example, a person may choose to hide the name of his employer on Facebook (thus this information becomes private for this person on Facebook), but if this person is also a member of LinkedIn, the name of her employer is likely to be publicly visible since the website is a social network for professionals. Third, some personal information may already be available for free or for a nominal fee on the Web beyond of the boundary of social networks. For example, the name of a person may appear on the website of his employer or his university, telephone numbers may appear in on-line phone lists, and the home address of a person may appear in real estate tax list on his county's website.

With the person identity obtained from the published social graph, the adversary can use powerful search engines to discover the personal information on the Web.

Practical Privacy Preservation Methods. Compared to vertex re-identification attack, it is very difficult to develop a practical privacy preservation method against information re-association attack due to several reasons. One reason is that a protection method with too much restrictions on user could prevent user from using social network site, and it is difficult to balance between requirements of the privacy and the easy-of-use. As mentioned before, vertex re-identification attacks usually happen when a large social graph is published to third-party application developers. For the data receiver, they can develop applications or models without access to precise information in individual personal record. Hence, anonymizing a graph in a certain degree is endurable for data receiver. Nevertheless, information re-association attack usually targets the online personal information that is viewed by users via Web browsers. Any perturbation on these personal information is not acceptable to many users.

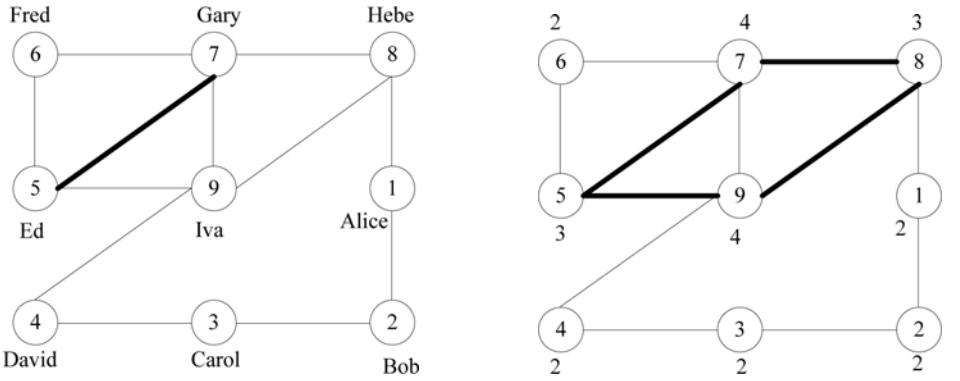
Another reason is that information re-association attack exploits privacy from many sources other than the published graph itself. It is very difficult to control the source of information release. For example, for the privacy breach from loose tweets [29], although it is possible for a user not to release his travel plan in his own tweets, it is difficult to prevent his friends from leaking his travel plan in their tweets.

1.5 Dissertation Contributions

Specifically, this dissertation makes the following contributions.

1.5.1 Edge Anonymization against Sensitive Edge Detection Attack

Edges in a graph may carry plenty of sensitive information, e.g. email communications, the frequency of communication, or trustworthiness. Similar to vertex anonymization method that protects sensitive information on vertices, edge anonymization method protects sensitive information on edges. Edge anonymization is more difficult than vertex anonymization because the sensitive



(a) Original Email Social Graph: Numbers inside circles are vertex ID randomly assigned. The edge between Ed and Gary is a sensitive relationship that Ed and Gary wants to hide.

(b) Anonymized Graph: this graph is a 2-anonymity graph. Numbers outside circles are vertex degree.

Figure 1.1: Example of Sensitive Relationship Disclosure

edges may still be breached even if the graph satisfies vertex anonymity requirement. Consider Figure 1.1 as an example. Figure 1.1a is a small social graph in which vertices represent people and edges represent email communications among people. Assume Ed and Gary want to hide their communications and designate the edge between them (the thick line) to be sensitive.

Figure 1.1b is a 2-anonymity graph in terms of vertex degree obtained from Figure 1.1a using a vertex anonymization method. Assume an adversary wants to know if there is any email between Ed and Gary. Suppose that the adversary also knows that Ed has emails with three people (so his vertex should have a degree of 3) and Gary has emails with four people (i.e., his vertex should have a degree of 4). By partitioning the graph in Figure 1.1b using the vertex degrees, the adversary can obtain three VECs, i.e. $V_1 = \{v_1, v_2, v_3, v_4, v_6\}$, $V_2 = \{v_5, v_8\}$, and $V_3 = \{v_7, v_9\}$. By observing Figure 1.1b, the adversary can only tell that Ed is in V_2 and Gary is in V_3, but cannot tell which of v_5 and v_8 is Ed and which of v_7 and v_9 is Gary. Thus vertex re-identification is prevented. However, the adversary can still infer with 100% certainty that there is an email communication between Ed and Gary because every vertex in V_2 has an edge with every vertex in V_3.

We develop two methods to protect the sensitive edges in an anonymized graph. We assume that an adversary can firstly launch vertex re-identification attack to find the pair of vertices representing

the victims in the published graph. Once the pair of vertices are re-identified, she is able to observe whether a sensitive edge exists between the pair of vertices. This attack is referred as *sensitive edge detection attack*. Our work focuses on coping with the three issues of anonymization methods discussed in Section 1.4.1.

1.5.2 A New Information Re-association Attack

Motivated by the observations that personal information may exist in many places on the Web other than the published social graph, we designs a new information re-association attack model, called information extraction attack. This model integrates web search engines and some existing information extraction techniques, such as named entity recognition (NER) [44] and name disambiguation [64], to automatically search the Web for private information hidden in a specific social network.

We define a measure to quantitatively analyze privacy disclosure under the new attack. This measurement also can evaluate the effectiveness of other similar information re-association attacks, given the privacy policy and protection mechanism of a social network site. We show that with some public information of a victim from a social network, it is not too hard for the adversary to utilize this attack to find private information. Our study can not only help to understand the strength of the privacy protection mechanism of social networks, but it will also help to develop new and more effective privacy preservation methods.

Chapter 2: LITERATURE REVIEW

In this chapter, we firstly review two categories of privacy attacks in social graphs, and then introduce the privacy preservation approaches.

2.1 Privacy Attacks in Social Graphs

2.1.1 Vertex Re-identification Attacks

Publishing an anonymized graph is a simple way to hide private information in a social graph. However, study [7, 27, 68] has shown that a trivially anonymized graph does not sufficiently protect privacy against a vertex re-identification attack whereby an adversary uses some background knowledge about a victim to determine the vertex of that person. Two types of background knowledge has been used in vertex re-identification attacks: vertex attributes and topological graph features.

Using vertex attributes to find the vertex of a victim is similar to finding the tuple of a victim in a published data table [4, 49]. The labels of vertices can be viewed as tuples in a data table. The attributes of the table can be divided into *quasi-identifier* (QI), such as age, gender, or zip code, and *sensitive attribute* (SA), such as disease or salary. Although no personal identity is included in the table, an adversary can still re-identify a victim in the table using the quasi-identifier, and discover personal sensitive information.

Wondracek et al. [57] describe an attack using group membership information of a social network (as a QI information) to re-identify the vertex of a victim in a social graph. The group membership of a member is a list of the public groups that the member participates in the social network site. The adversary can obtain the group membership information of a victim by hijacking the victim's browsing history from the victim's web browser. The stolen group membership listing is used to build a fingerprint of the victim. The public group membership listing of each vertex in a published anonymized social graph is used to build a fingerprint for that vertex. By comparing the

fingerprint of the victim with the fingerprints of vertices, the victim's vertex is re-identified, and other information about the victim is completely disclosed.

Topological graph feature is another type of background knowledge often used in the vertex re-identification attack, e.g. vertex degree [37], vertex neighborhood [68], or some other structural features [27]. In this case, the adversary can use her knowledge about the victim, such as how many friends the victim has on a social network, to infer a structural feature of the person's vertex. As mentioned in Section 1.4.1, the adversary can then use this structural feature to partition the vertices of a trivially anonymized graph into equivalence classes (VECs), so that each VEC contains vertices that have the same graph feature. The victim's vertex is identified if the VEC with the graph feature of the victim has a single vertex. The adversary may obtain the graph feature of a victim by actively participating in a social network site [7], or by scrapping another network site to explore the property of structure overlapping among social network sites [45].

2.1.2 Information Re-Association Attacks

Section 1.3 mentioned that various existing techniques are used in this type of privacy attacks. In this section, we introduce the following techniques.

- Using Statistics Models.

  Zheleva and Getoor [67] present a set of information inference models that can infer sensitive attribute values of users in a social graph. The social graph contains multiple types of information about users, such as links between users, group memberships, locations, and gender. They show that a user's hidden sensitive attribute value could be estimated from the published information in the graph. For example, using a friend-aggregate model, the probability of the (unpublished) location of the victim can be estimated from the frequencies of the locations of the victim's friends who do not consider their location information sensitive.

- Using Data Mining Techniques.

  Mao et al. [29] notice that Twitter users unwittingly disclose their private information in

tweets. They analyzed three types of private information disclosure on Twitter.com: divulging vacation plans, tweeting under the influence of alcohol, and revealing medical conditions. They build automatic classifiers to detect tweets for these three topics in real time and demonstrated a real threat to privacy of users. The classified tweets is a part of vertex information in a published social graph. Their automatic private information detection can be used directly as a privacy attack on the vertices of social graph.

- Using Computer Security Attacks.

  Researches [23] mention that the *Sybil* attack can be used to view a victim's hidden information on a social network. A social network site could grant a user the right to view other user's information according to the user's reputation, so that a person with higher reputation is allowed to view information of more people (or equivalently a larger portion of the graph of the social network). Using a Sybil attack an adversary with a low reputation will create a large number of pseudonymous entities and use these entities to inflate her own reputation disproportionately. As a result, she can gain access to victim's information that would otherwise be forbidden.

2.2 Privacy Preservation of Social Graphs

2.2.1 Vertex Anonymization Methods

Tabular data anonymization methods [1–3, 33, 54] can be easily applied to anonymize vertex labels to prevent vertex re-identification attacks based on vertex attributes. These methods were developed in the past ten years based on k-anonymity [49] concept. To use the k-anonymity methods, we view vertex labels as tuples in a data table and determine the QI and SA attributes in the tuples. Then, we can use the methods to generalize QI values in the data and to group tuples by their generalized QI values. Each QI group will contain at least k tuples. As a result, for each generalized vertex label, there are at least k vertices with that label, and vertex re-identification is prevented. However, as Machanavajjhala et al. [41] has pointed out, sensitive information can still

be disclosed if all tuples in a QI group have the same sensitive value. To solve this problem, they proposed l-diversity as a privacy measure, which requires each QI group to contain at least l well-represented sensitive values. Following this work, many l-diversity based methods [36, 58, 60, 61] were proposed to improve both the privacy and the utility. These methods can also be applied to anonymize vertex labels.

Recent vertex anonymization methods [27, 37, 68, 69] are developed to prevent vertex re-identification attacks based on graph features. Many of these methods are also inspired by the k-anonymity concept. According to different graph features, these methods obtain an anonymized social graph by altering the structure of the original social graph, resulting in different level of privacy guarantees and graph quality. In the following, we briefly discuss the vertex anonymization methods in terms of the graph features they use.

- Vertex Degree.

  Liu and Terzi [37] consider a vertex re-identification attack that uses vertex degree to partition a social graph. They use edge addition/deletion method to construct a k-anonymity graph that preserves the vertex degree distribution of the original graph.

- Neighborhood.

  Zhou and Pei [68] consider a vertex re-identification attack that partitions a social graph according to the neighborhoods of vertices. The neighborhood of a vertex will include direct neighbors of the vertex and the edges among these neighbors. They also apply edge addition/deletion method to construct a published graph in which every vertex has equivalent neighborhood (in terms of graph isomorphism) with at least $k - 1$ other vertices. Their method uses the number of changed edges as the utility loss measure, and endeavors to construct a published graph with minimal edge changed.

- Graph Automorphism.

  Zou et al. [69] consider a vertex re-identification attack that partitions a social graph using graph automorphism. This attack is regarded as the strongest attack based on only graph

features. Given a graph $G = (V, E)$, where V is the vertex set and E is the edge set, an automorphism of G is a function f that maps the vertex set V to itself, such that for any edge $e = (u, v)$ in G, $f(e) = (f(u), f(v))$ is also an edge in G. Their method constructs a k-automorphism graph by adding edges, so that each vertex is automorphic to at least $k - 1$ other vertices. To preserve the utility of the graph, their method also attempts to minimize the number of edges added. Wu et al. [59] define their privacy requirement k-symmetry also based on automorphism partition. They add vertices/edges to construct anonymized graph. Since adding vertices causes huge utility loss, they propose sampling methods to extract approximate versions of the original graph from the anonymized one.

- Graph Isomorphism

  Cheng et al. [12] define k-security privacy requirement to guarantee the privacy of both vertex information and edge information under the vertex re-identification attack using any structural features about victims. They propose an anonymization method relying on k-isomorphism concept. A graph G is k-isomorphic if G consists of k disjoint subgraphs $g_1, ..., g_k$, i.e. $G = g_1, ..., g_k$, where g_i and g_j are isomorphic for $i \neq j$. An k-isomorphic graph must be k-security. Their method deletes edges to partition a connected graph into k disjoint subgraphs, and then adds or deletes edges to ensure pairwise subgraph isomorphism.

- Hub Connections.

  Hay et al. [27] consider a variety of vertex re-identification attacks that partition social graphs using vertex degree or various sub-graph structures, including the vertex neighborhood [68]. In addition, they also consider privacy attacks that employs the hub fingerprint of a vertex. A hub is a vertex in a network that has a high degree and high centrality (i.e., passed through by many shortest paths). The hub fingerprint of a vertex is the set of lengths of the shortest paths from the vertex to hub vertices in the graph. As referred in Section 1.4.1, their method partitions the vertices in the graph into groups via a simulated annealing. Once the partition is done, each vertex group is replaced by a supernode, and edges between a pair of vertex

groups are replaced by a superedge. Since the topology of vertices in a supernode is hidden, the vertex re-identification attack is prevented.

2.2.2 Edge Anonymization Methods

Edge anonymization methods can be grouped into two categories depending on whether the published graph has labels on edge.

Edge Anonymity of Labeled Graphs.

A number of recent edge anonymization methods considered social graphs in which edges have labels and aim to protect sensitive labels in edges. Most of these methods do not change the topology of the original graph. In the following, we introduce these methods according to the types of edge properties carried on edges.

- Multiple Types of Undirected Edges.

  Zheleva et al. [66] study social network graphs in which edges are labeled by types of relationships. Some relationships are sensitive but others are not. They consider a privacy attack whereby the adversary infers sensitive relationships from non-sensitive ones in the graph. For example, knowing that two people were classmates, the adversary may infer that the two people are also friends. They develop several edge anonymity strategies, including removing all sensitive edges, removing some non-sensitive edges, and aggregating vertices/edges into clustered vertices/edges. They measure the privacy by the percentage of sensitive edges that can be re-identified with a high probability, and measure the graph utility loss by the number of labels that have been changed.

- Relationship Weights in a Undirected Graph.

  Liu et al. [38] focus on social graphs in which edges are undirected and are labeled by some weights. They study privacy attacks that aim to discover the true weights in edges. Their method uses a greedy strategy to apply the Gaussian randomization multiplication to perturb edge weights. The privacy is measured by the number of edges of which the weights in the

anonymized graph are different from those in the original graph. Their method maximizes the number of edges that have perturbed weights and preserves the length of shortest paths as many as possible.

- Relationship Weights in a Directed Graph.

  Das et al. [17] study social graphs in which edges are undirected and are labeled by weights. They also consider privacy attacks that aim to discover the true weights. Their anonymization method changes the weights of edges to satisfy a privacy requirement called edge weight k-anonymity, and preserves a linear property of the original graph. The weight k-anonymity requires that the weight of each edge is indistinguishable (defined by a threshold) to at least $k-1$ other edges emanating from the same vertex. A linear property of a graph is a system of linear inequalities involving edge weights as variables. Intuitively, a linear property models the execution of some graph algorithms, such as Kruskal's minimum spanning tree and Dijkstra's shortest path. Thus, preserving the linear property allows these graph algorithms to obtain similar results from the original and the anonymized graphs. Their method uses a linear programming approach to determine the new edge weights of the anonymized graph.

Edge Anonymity of Unlabeled Graphs.

In a social graph without edge labels, all edges are of the same type. Some or all edges may represent sensitive relationships. In this type of graphs, edge anonymity is to protect the existence of a relationship. A number of anonymization methods were proposed recently for this type of graphs.

Ying and Wu [63] study social graphs in which edges are undirected, unlabeled and all edges are sensitive. Their method measures the privacy by the difference between the prior and the posterior probabilities of edges in the original and the anonymized graphs, respectively. Both the prior and the posterior probabilities are calculated from vertex degrees. For example, the prior probability for a vertex of degree d_i and a vertex of degree d_j to have an edge in the original graph of n vertices is given by $\frac{d_i}{n-1} + \frac{d_j}{n-1} - \frac{d_i}{n-1}\frac{d_j}{n-1}$. Intuitively, $\frac{d_i}{n-1}$ is the probability that a

randomly selected vertex is an neighbor of the vertex with the degree d_i. Their edge anonymization method constructs anonymized graph by randomly adding (deleting or swapping) edges, so that the difference between the prior and the posterior probabilities is less than a given threshold. The resulting graph preserves the largest eigenvalue of the adjacency matrix and the second largest eigenvalue of the Laplacian matrix of the original graph.

Bhagat et al. [9] examine a social graph that is bipartite with one type of labeled vertices representing people and another type of labeled vertices representing various types of relationships. Edges connecting people with relationships are undirected and unlabeled. Their approaches are designed to prevent attacks that identify the people involved in a given relationship. There are two approaches: label lists and partitioning.

- Label lists. This approach assigns each vertex a list of labels that contain the vertex's true label and some other labels. Vertices with the identical label lists are grouped together as a class. The assignment of label lists must satisfy a class safety property that requires each vertex participates in interactions with no more than one vertex from the same class. This allows vertices to have multiple interactions (e.g., to participate in a friendship interaction and an email interaction), but prohibits multiple vertices from the same class to be involved in one relationship.

- Partitioning. This approach also partitions vertices into classes, but instead of releasing edges for each vertex, it only releases the number of edges between and within each class. Thus the anonymized graph is similar to the supergraph proposed by Hay et al. [27].

Singh and Zhan [51] propose a privacy measure for graphs called the topological anonymity, which measures the average obscurity of vertex identity and edges existence in a unlabeled graph. The topological anonymity is defined as the average difference between the total number of vertices of a set of selected degrees and the total number of vertices of a degree in a given sequence. A vertex degree is selected if the variance of the clustering coefficient of the vertices of this degree is larger than a threshold. The degree sequence contains degrees from 1 to a given threshold.

Intuitively, a low topological anonymity value indicates a few vertices belong to degree groups with high clustering coefficient thus both vertices and edges are not well-protected. As the topological anonymity value increases, the ability for the structure to hide vertices and edges also increases. They use the topological anonymity to measure two random graphs and one real-world social graph, but do not propose any method to obtain an anonymized graph for a given topological anonymity requirement.

Chapter 3: DEGREE-BASED EDGE ANONYMIZATION

As illustrated in Figure 1.1, sensitive edges, shorted as s-edges, are still disclosed when vertex identities are protected. To protect s-edges, we define *edge confidentiality* as privacy measure in Section 3.1. We conduct experiments to show that s-edges disclosure takes place in real-world social graphs. The experimental results are explained In Section 3.2. Then, several edge anonymization algorithms are introduced in Section 3.3.

3.1 Privacy Measure: Edge Confidentiality

A social graph G is a simple undirected unlabeled graph of which the set of vertices is denoted by $V(G)$ and the set of edges by $E(G)$. Each vertex represents (and is also referred as) a person. We assume that in the original graph, some edges [25] are designated as sensitive. The original graph is the social graph trivially anonymized by removing the identities on vertices.

Definition 1. *(***Graph Partition***) Given a graph G and an equivalence relation R over $V(G)$. The partition of G induced by R (or R-partition) is $P(G,R) = \{C_i | 1 \leq i \leq c\}$, where $c \leq |V(G)|$ and $C_i \subseteq V(G)$ is a vertex equivalence classes (or VECs), that is, vRu for any $v, u \in C_i$. For any $C_i, C_j \in P(G,R)$, the set of edges that connect vertices in C_i with vertices in C_j is the edge equivalence class (or EEC) E_{ij}.*

We assume the following sensitive edge detection attack model. The adversary has some knowledge about two victims and uses it to define an equivalence relation on the published graph. The adversary can partition the graph based on the equivalence relation and locate the VECs of the victims. According to the VECs (and the corresponding EEC), the adversary computes the probability of an edge between the two victims. If the probability is high and the edge is sensitive in the original graph, the privacy is disclosed.

Definition 2. (**Edge Disclosure Probability**) *Let C_i and C_j be two VECs in a partition of an anonymized graph G and each of them contains a victims. The probability that a sensitive edge*

(or s-edge) between the two victims is disclosed is

$$\begin{aligned} & P[C_i, C_j] \\ & = P[X = 1, Y = 1 | C_i, C_j] \\ & = P[Y = 1 | X = 1, C_i, C_j] \cdot P[X = 1, C_i, C_j] \end{aligned} \tag{3.1}$$

where X *and* Y *are binary random variables,* $X = 1$ *if the two victims have an edge in the original graph, and* $Y = 1$ *if an edge between* C_i *and* C_j *in the original graph* G *is sensitive.*

Intuitively, in Eq (3.1), $P[X = 1, C_i, C_j]$ is the probability that the two victims have an edge in original graph and $P[Y = 1 | X = 1, C_i, C_j]$ is the probability that the edge is sensitive. Let α_{ij} be the number of s-edges in E_{ij}, γ_{ij} be the number of edges in E_{ij} that are also in the original graph, $\delta_{ij} = |E_{ij}|$ be the number of edges in E_{ij}, and

$$\beta_{ij} = \begin{cases} \frac{1}{2} \cdot |C_i| \cdot (|C_i| - 1), & i = j; \\ |C_i| \cdot |C_j|, & i < j. \end{cases} \tag{3.2}$$

be the number of pairs of vertices between C_i and C_j. Due to vertex equivalence, the β_{ij} pairs of vertices are equally likely to be the pair of victims. Since γ_{ij} out of β_{ij} pairs of vertices are connected by edges in the original graph, $P[X = 1, C_i, C_j] = \frac{\gamma_{ij}}{\beta_{ij}}$. However, even if there is an edge between the pair of victims, the privacy is not disclosed unless the edge is sensitive. Again, because the equivalent edges in E_{ij} are equally likely to be sensitive, thus $P[Y = 1 | X = 1, C_i, C_j] = \frac{\alpha_{ij}}{\gamma_{ij}}$. In addition, some edges in E_{ij} might be counterfeit. Without additional knowledge, each edge in E_{ij} is equally likely to be genuine. Thus, $P[X = 1, C_i, C_j] = \frac{\gamma_{ij}}{\delta_{ij}} \cdot \frac{\delta_{ij}}{\beta_{ij}}$. Put these together,

$$P[C_i, C_j] = \frac{\alpha_{ij}}{\gamma_{ij}} \cdot \frac{\gamma_{ij}}{\beta_{ij}} = \frac{\alpha_{ij}}{\gamma_{ij}} \cdot \frac{\gamma_{ij}}{\delta_{ij}} \cdot \frac{\delta_{ij}}{\beta_{ij}} \tag{3.3}$$

Example 1. Consider two VECs C_1 and C_2, where C_1 contains 3 vertices, C_2 contains 2 vertices, and E_{12} contains 4 edges among which 3 edges are in the original graph and out of the 3 edges 2

are sensitive. Then, $\alpha_{12} = 2$, $\gamma_{12} = 3$, $\delta_{12} = 4$, $\beta_{12} = 6$, and $P[C_1, C_2] = \frac{2}{3} \cdot \frac{3}{4} \cdot \frac{4}{6} = \frac{1}{3}$.

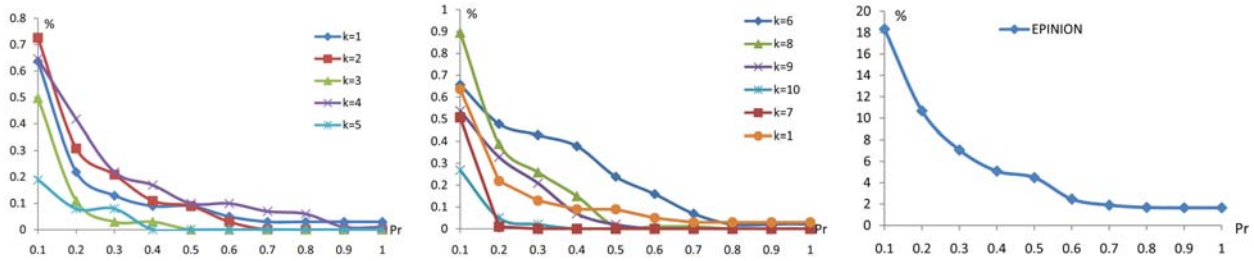
Definition 3. (**Edge Confidentiality**) *A graph G is τ-confidential under an equivalence relation R if $conf(G) \geq \tau$ for some $0 \leq \tau \leq 1$, where $conf(G) = 1 - \max\{P[C_i, C_j] | C_i, C_j \in P(G, R)\}$ is the edge confidentiality of G.*

As we see, edge confidentiality depends on equivalence relation defined on some background knowledge. In this chapter, we assume that the adversary uses vertex degree to define the equivalence relation, and all edges are sensitive in the original or anonymized graph. Thus, the α_{ij} and γ_{ij} both equal to δ_{ij} in E_{ij} of Eq. 3.3. This assumption is a special case of Definition 2. In the next chapter, we will deal with the general case.

3.2 Edge Disclosure in Social Graphs

To conduct the experiments, we considered two data sets: EPINION [20] and COA [14]. The EPINION dataset is the "Web of trust" social graph extracted from Epinion.com. It contains 49,287 vertices and 381,035 edges. The COA dataset is a social graph used in [37]. The dataset is an undirected graph containing 7,955 vertices and 10,055 edges.

To investigate edge disclosure in the graphs produced by vertex anonymization algorithms, we implemented a vertex k-anonymity algorithm described in [37]: the *priority* algorithm with the probing scheme using vertex degree and edge deletion as anonymization strategy. We applied this algorithm on COA graph to generate anonymized graphs. However, we were unable to obtain vertex anonymized graph of EPINION dataset using this algorithm due to the size and density of the graph. In fact, all existing vertex k-anonymity algorithms have some problems working on EPINION. For example, we also implemented the simulated annealing algorithm of [27], which searches for anonymized graph that optimizes a likelihood estimate. However, the computation of the likelihood of a single graph is $O(|E|)$. Due to the size of EPINION graph, the computation for split or moving one vertex can cause an out-of-memory exception on a computer with 2GB main memory and can take 23 minutes on a computer with 128GB main memory.



(a) Edge Disclosure COA 1 (b) Edge Disclosure COA 2 (c) Edge Disclosure in Epinion

Figure 3.1: Edge Disclosure in Social Networks

We partitioned each anonymized COA graph and original EPINION graph using vertex degree, measured the edge disclosure probability of each EEC, and counted percentage of edges of various edge disclosure probabilities. The results are shown in Figure 3.1, in which the X-axis is the edge disclosure probability and the Y-axis is the percentage of edges. In Figures 3.1a and 3.1b, curves correspond to vertex k-anonymity graphs of COA, where $k = 1$ corresponds to the original graph. A point (x, y) on a curve means that y percent of edges have a edge disclosure probability of at least x. Based on the results, we have the following observations.

1. Edge disclosure is more likely to occur in dense graphs. As shown in Figure 3.1c, in EPINION graph, 5% (or 17064) edges have a edge disclosure probability of 0.5 or higher, and 2% (or 6280) edges are completely disclosed. These numbers become 0.1% (9) and 0.03% (3), respectively, in COA graph.

2. Edge disclosure can still occur in vertex k-anonymity graphs. As shown in Figures 3.1a and 3.1b, even though the original COA graph has much less risk of edge disclosure than EPINION, the vertex k-anonymity algorithm still cannot protect edge anonymity. Interestingly, for $k = 4$ or 6, the anonymized graphs have higher risk of edge disclosure than the original graph.

Based on this analysis, we believe that algorithms specifically designed for edge anonymity are needed.

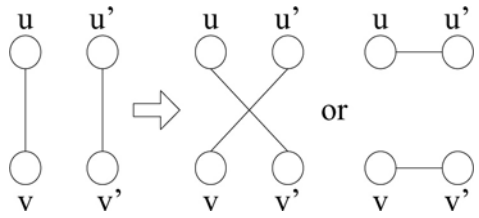


Figure 3.2: Edge Swap

3.3 Algorithms for Degree-based Edge Anonymity

To achieve edge anonymity, we considered graph perturbation strategies. There are four graph perturbation strategies: 1) random edge deletion, 2) random edge addition, 3) random edge swap and 4) random edge addition/deletion. Edge swap is a special type of edge addition/deletion that deletes two edges and adds back two new edges connecting the four vertexes in one of the two specific ways illustrated in Figure 3.2. A basic operation of each strategy makes a minimum change to a graph. For example, deleting one edge is a basic operation of edge deletion, and swapping two edges is a basic operation of edge swap.

For edge anonymity, these strategies have different impact on edge disclosure probability. For example, edge deletion can always reduce edge disclosure probability, but edge addition may increase the probability. On the other hand, the effect of edge swap is often difficult to predict. These anonymization strategies also have different impact on different graph measurements [16]. For example, edge swap does not alter vertex degree, but may change centrality and shortest paths.

In the rest of this section, we present algorithms that perform either edge swap or edge deletion. Intuitively, it is also possible to obtain edge anonymity by adding edges. However, directly adding counterfeit edges to the original graph will not help, because adding more edges to EECs will increase edge disclosure probabilities in terms of Eq. 3.3 and assumptions of this chapter. (Although the added edges do not represent real relationships, they cause those real relationships to be identified more easily.)

Algorithm 3.1 Degree-based Edge Swap

Input: graph $G = (V, E)$ and edge confidentiality threshold τ
Output: τ-confident graph G'
Method:
1. $G' = G$;
2. partition G' by vertex degree;
3. while (confidence of G' is less than τ) do
4. randomly select an edge e_1 from the leading EEC;
5. find second edge e_2 according to Theorem 1;
6. if e_2 exists, perform edge swap with e_1 and e_2;
7. else return an empty graph;
8. end while
9. return G';

3.3.1 Degree-based Edge Swap

Algorithm 3.1 takes as input a graph and a confidentiality threshold, and used edge swap to obtain a τ-confident anonymized graph if one can be found or an empty graph otherwise. The goal is to find a graph that not only satisfies the privacy requirement but also has a good utility. To achieve this goal, the algorithm uses a greedy strategy to improve edge confidentiality, namely, it focuses on reducing the size of the *leading EEC*, which corresponds to the maximum edge disclosure probability of the graph. Intuitively, reducing the size of the leading EEC may improve edge confidentiality more quickly than reducing the size of other EECs, therefore result in fewer edges being swapped and better utility of anonymized graphs.

In each iteration (steps 3-8), the algorithm attempts to swap the pair of edges that can lead to the biggest improvement of the edge confidentiality. If such a pair of edges does not exist, the algorithm will terminates with an empty graph. To choose the edges to swap, Algorithm 3.1 takes an edge from the leading EEC and find a second edge from an EEC that satisfies the conditions of the following theorem.

Theorem 1. Let a graph G be a graph partitioned using vertex degree and G' be the graph obtained by a valid swap of two edges $e_1 \in E_{ij}$ and $e_2 \in E_{st}$. where $i \leq j$, $s \leq t$. Assume that every edge in $V(G)$ or $V(G')$ is sensitive. Then, for each EEC E_{xy} in G' that receives any new edge, the corresponding edge disclosure probability p'_{xy} is less than p_{ij} if and only if indexes i, j, s, t contain

at least two distinct values and each of these values appears at most twice, and one of the following conditions holds.

1. $i = j$, $s = t$, and $\alpha_{is} < \alpha_{ii} \cdot \frac{\beta_{is}}{\beta_{ii}} - 2$;

2. $i < j$ or $s < t$, and for $x \in \{i, j\}$, $y \in \{s, t\}$, $\alpha_{xy} < \alpha_{ij} \cdot \frac{\beta_{xy}}{\beta_{ij}} - 1$;

Where α_{**} is the number of sensitive edges in an EEC, and β_{**} is the number pairs of vertices between two VECs.

Proof. (if) Assume that i, j, s, t contain at least two distinct values and each of these values appears at most twice, and one of the two conditions holds.

If condition 1 holds, $e_1 \in E_{ii}$, $e_2 \in E_{ss}$, and E_{ii} and E_{ss} must be different EECs. The edge swap will delete one edge from each of E_{ii} and E_{ss}, and add two edges to E_{is}. Since $\alpha_{is} < \alpha_{ii} \cdot \frac{\beta_{is}}{\beta_{ii}} - 2$, by Eq. 3.3, $p_{ij} = p_{ii} > p_{is}^{'}$. Since E_{is} is the only EEC receives a new edge, we are done.

If condition 2 holds, we consider three cases.

Case 1: $i = j$ and $s < t$. In this case, $e_1 \in E_{ii}$, $e_2 \in E_{st}$, and since i, j, s, t contain at least two distinct values each of which appears at most twice, E_{ii} and E_{st} must share no VEC. Thus, after the edge swap, EECs E_{is} and E_{it} each receives one new edge. Since for $x \in \{i, j\}$, $y \in \{s, t\}$, $\alpha_{xy} < \alpha_{ij} \cdot \frac{\beta_{xy}}{\beta_{ij}} - 1$, by Eq. 3.3, we have $p_{ij} = p_{ii} > p_{is}^{'}$ and $p_{ij} = p_{ii} > p_{it}^{'}$.

Case 2: $i < j$ and $s = t$. In this case, $e_1 \in E_{ij}$, $e_2 \in E_{ss}$. Similarly, E_{ij} and E_{ss} must share no VEC. Thus, after the edge swap, EECs E_{is} and E_{js} each receives one new edge. Since for $x \in \{i, j\}$, $y \in \{s, t\}$, $\alpha_{xy} < \alpha_{ij} \cdot \frac{\beta_{xy}}{\beta_{ij}} - 1$, by Eq. 3.3, we have $p_{ij} > p_{is}^{'}$ and $p_{ij} > p_{js}^{'}$.

Case 3: $i < j$ and $s < t$. In this case, $e_1 \in E_{ij}$ and $e_2 \in E_{st}$, and after the edge swap, EECs that receive new edges will be depending on the order of i, j, s, t. These orderings are $s < t \leq i < j$, $s < i \leq t < j$, $s \leq i < j \leq t$, $i < s < t < j$, and $i < j \leq s < t$. The proofs are similar for these orderings. We only show the proof for $i < j = s < t$. For this ordering, E_{it} and E_{js} each receives a new edge. Since for $x \in \{i, j\}$, $y \in \{s, t\}$, $\alpha_{xy} < \alpha_{ij} \cdot \frac{\beta_{xy}}{\beta_{ij}} - 1$, by Eq. 3.3, we have $p_{ij} > p_{it}^{'}$ and $p_{ij} > p_{js}^{'}$.

(only if) Assume that for each EEC E_{xy} in G' that received a new edge, the corresponding edge disclosure probability $p^{'}_{xy}$ is less than p_{ij}.

By Eq. 3.3, $p_{ij} > p^{'}_{xy}$ implies $\alpha_{xy} < \alpha_{ij} \cdot \frac{\beta_{xy}}{\beta_{ij}} - c$, where $c = 1$ or 2 depending on the number of new edges E_{xy} receives. Notice that, since the edge swap can at most involve four VECs, we have $x, y \in \{i, j, s, t\}$.

We prove that indexes i, j, s, t contain at least two distinct values and each of these values appears at most twice. This can be proved by contradiction. Assume that there is only one distinct value. Then $i = j = s = t$. That is, $e_1, e_2 \in E_{ii}$. The swap will remove two edges from E_{ii} and add two new edges back to E_{ii}. Therefore, $p_{ij} = p_{ii} = p^{'}_{ii} = p^{'}_{xy}$, a contradiction. All cases in which i, j, s, t contain two distinct values and one value appears three times can be proved similarly.

Thus, indexes i, j, s, t contain at least two distinct values and each of these values appears at most twice. There are 5 cases in which i, j, s, t contains exactly two pairs of identical indexes, where the first pair is: $i = j$, $i = s$, $i = t$, $j = s$, or $j = t$. If $i = j$ then $s = t$. Therefore, $e_1 \in E_{ii}$ and $e_2 \in E_{ss}$. After the edge swap, EEC E_{is} will receive two edges. Thus, $p_{ij} > p^{'}_{xy}$ implies $\alpha_{is} < \alpha_{ii} \cdot \frac{\beta_{is}}{\beta_{ii}} - 2$. This givens the condition 1.

The remaining four cases imply that $i < j$ and $s < t$. The proofs of these cases are similar. We show the proof for the case $i = s$. In this case, $j = t$. Therefore, $e_1, e_2 \in E_{ij}$ and after the edge swap, E_{ii} and E_{jj} each receives one new edge. Thus, $p_{ij} > p^{'}_{xy}$ implies $\alpha_{ii} < \alpha_{ij} \cdot \frac{\beta_{ii}}{\beta_{ij}} - 1$ and $\alpha_{jj} < \alpha_{ij} \cdot \frac{\beta_{jj}}{\beta_{ij}} - 1$.

The cases in which $i, j, s., t$ contain exactly three and exactly four distinct values can be proved similarly. Notice that in these cases, we have either $i < j$ or $s < t$, and the EECs that can ever receive a new edge is among E_{is}, E_{it}, E_{js}, and E_{jt} (notice that $E_{xy} = E_{yx}$). The details are omitted. □

Lemma 1. Given E_{ij} and E_{st} that satisfy the condition of Theorem 1. Any pair of edges $e_1 \in E_{ij}$ and $e_2 \in E_{st}$ will reduce p_{ij} by the same amount.

Proof. As indicated in the proof of Theorem 1, which is independent of the choice of e_1 and e_2. □

Intuitively, Theorem 1 guarantees that the swap of the appropriate pair of edges will always reduce the maximum edge disclosure probability p_{ij} and will not cause probabilities of other EECs to become greater than p_{ij}. Lemma 1 indicates that as long as the appropriate EECs are determined, the choice of edges within these EECs does not make any difference provided a valid swap can be made.

Algorithm 3.2 Edge Deletion with Maximum Choice

Input: graph $G = (V, E)$ and edge confidentiality threshold τ
Output: τ-confident graph G'
Method:

```
1.  G' = G;
2.  partition G' by vertex degree;
3.  while (edge confidentiality of G' is less than τ) do
4.    for each edge e in the leading EEC do
5.      compute RMLP and IOLP of deleting e;
6.    delete the edge of maximum RMLP and
      minimum IOLP;
7.  end while
8.  return G';
```

3.3.2 Degree-based Edge Deletion

Algorithm 3.2 takes as input a graph and an edge confidentiality threshold, and returns a τ-confident graph using edge deletion.

To preserve utility, Algorithm 3.2 also focuses on deleting edges of the leading EEC. To select an edge to delete, it estimates the amount of decrease of the maximum edge disclosure probability and the amount of probability increase of other EECs that will be resulted from the deletion of the edge. The selected edge should result in the largest *reduction to the maximum edge disclosure probability* (RMP). If more than one such edge exists, the one that results in the smallest *increase of other edge disclosure probabilities* (IOP) should be selected. Notice that it is possible that the deletion of the selected edge does not immediately improve the edge confidentiality. However, the algorithm will not stop if this situation occurs because by deleting more edges the maximum edge disclosure probability will eventually be reduced, and a τ-confident graph will be obtained.

Furthermore, this can happen independent of the order in which edges are deleted.

To efficiently estimate the reduction of the maximum edge disclosure probability and the probability increase of other EECs resulted from deleting an edge in the leading EEC, the algorithm does the following.

Suppose that the leading EEC is E_{ij} and the edge (u, v) is under consideration, where $u \in C_i$, $v \in C_j$, and VEC C_i contains vertexes of degree i. If (u, v) is deleted, u and v will be moved into VECs C_{i-1} and C_{j-1}, respectively. If $i \neq j$, the deletion of the edge will decrease the sizes of C_i and C_j and increase the sizes of C_{i-1} and C_{j-1}, each by one. If $i = j$, the deletion of the edge will decrease the size of C_i and increase the size of C_{i-1}, each by two. As u and v are moved, so will edges incident to u or v. To be specific, consider an edge (u, w), where w is in some VEC C_s. Once u is moved, this edge will also be moved from EEC E_{is} into EEC $E_{(i-1)s}$. This move will decrease the size of E_{is} and increase the size of $E_{(i-1)s}$, each by one. The size changes of EECs affected by moving v can be determined similarly. Thus, the size changes of VECs and EECs affected by the edge deletion can be efficiently determined without actually moving vertexes and edges. These size changes can then be used to calculate RMP and IOP.

Algorithm 3.3 Edge Deletion with Random Choice

Input: graph $G = (V, E)$ and threshold τ
Output: τ-confident graph G'
Method:
1. $G' = G$;
2. partition G' by vertex degree;
3. while (edge confidentiality of G' is less than τ) do
4. randomly delete an edge in the leading EEC;
5. return G';

Algorithm 3.3 is an alternative edge deletion method, which chooses a random edge, instead of the best edge, from the leading EEC for deletion.

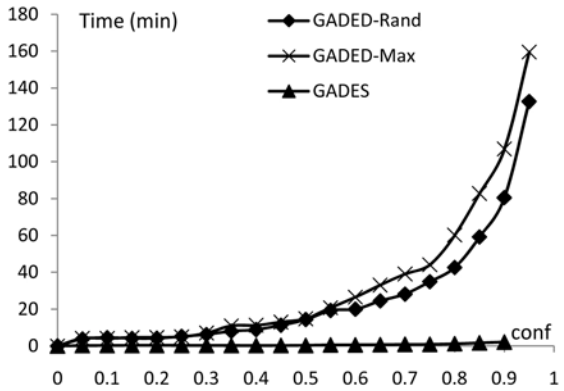


Figure 3.3: Performance of Algorithms on EPINION Dataset

3.4 Experimental Results

We conducted two experiments to study the performance of proposed algorithms and utility of anonymized graphs. Experiments were performed on three datasets. In addition to EPINION and COA datasets, we also used KDDCUP [30] dataset, which is a collection of abstracts of papers in high energy physics published between 1992 through 2003. The graph contains 19,101 vertices and 30,010 edges. Our experiments were performed on a PC with 2.13GHz Pentium Core 2 processor and 2GB memory running a Ubuntu Linux operation system.

To study the performance of the three algorithms, we run the algorithms on all three datasets. For COA and KDDCUP graphs, the execution of the algorithms is fast because only a small number of edges need to be changed. Here we only show the results obtained from the EPINION graph. As shown in Figure 3.3, edge swap (GADES) is always more efficient than edge deletion (GADED) especially for higher confidentiality thresholds. This is perhaps because that edge swap can always reduce the maximum edge disclosure probability but edge deletion may not. It is interesting that the two versions of edge deletion have almost identical performance when $\tau \leq 0.5$. But as confidentiality threshold becomes higher, GADED-Rand becomes more efficient than GADED-Max, which is as expected.

We compare the utility of anonymized graphs obtained by the three algorithms from the three datasets. The utility of anonymized graphs is measured by three different measurements: namely,

the changes of the graph edges, the degree distribution, and the clustering coefficients.

To measure the change of the graph edges, we count the number of edges that are deleted (due to edge deletion or edge swap) and added (due to edge swap), and calculate the relative ratio of edge change $RREC = \frac{|E|-|E' \cap E|}{|E|}$, where E is the set of edges in the original graph and E' is the set of edges in the anonymized graph. This ratio has been used to measure utility of anonymized graphs by several researchers [37, 68].

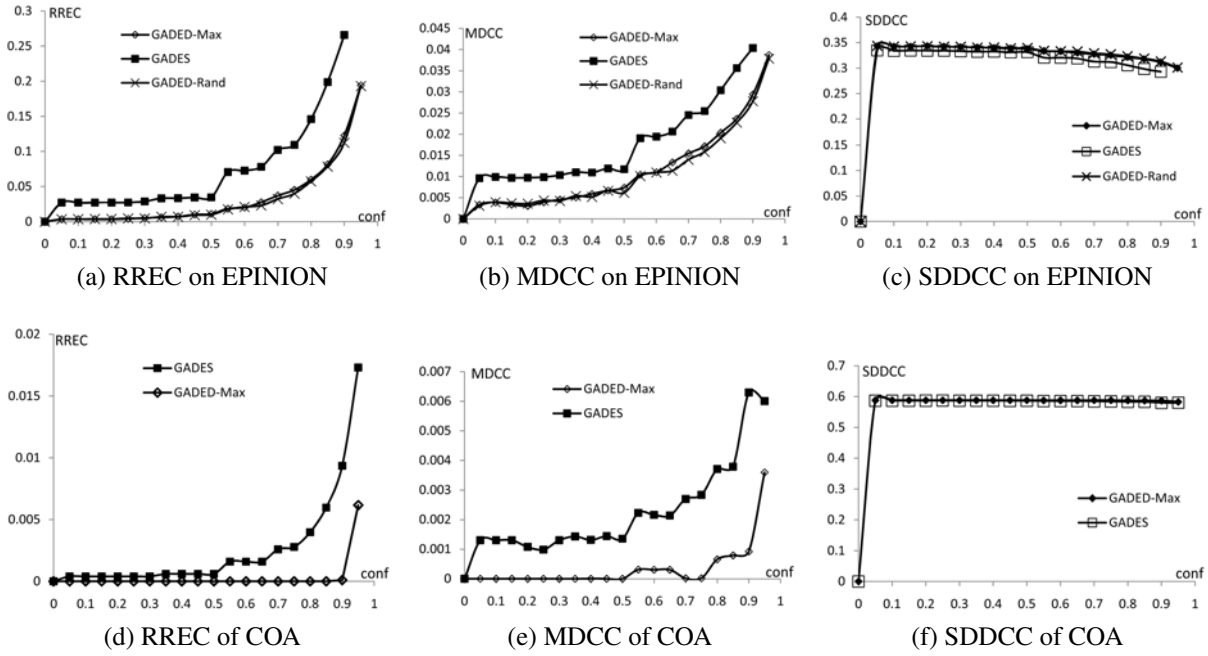
To measure the change of degree distributions, we calculate the distance between degree histograms of the original and the anonymized graphs. There are several distance measures for histograms. In our experiments, we use the earth mover distance (EMD) [48], which is the minimal cost of transforming one distribution into another. EMD has been used by researchers in many different areas [48].

The clustering coefficient (CC) of a vertex u is defined as $c_u = \frac{2l_u}{k_u(k_u-1)}$, where k_u is the number of neighbors of vertex u and l_u is the number of edges among neighbors of vertex u. Intuitively, clustering coefficient measures the closeness of the vertex and its neighbors in the graph and determines whether the graph has the small world characteristics. To measure the change of clustering coefficients, we calculate for each vertex u the difference of clustering coefficients $\Delta c_u = |c_u - c_u^{'}|$, where c_u and $c_u^{'}$ are calculated from the original and the anonymized graphs, respectively. We use the mean $MDCC = \frac{1}{|V|} \sum_{u \in V} \Delta c_u$ and the standard deviation $SDDCC = (\frac{1}{|V|-1} \sum_{u \in V} (\Delta c_u - MDCC)^2)^{\frac{1}{2}}$ of these differences as a utility measure.

To obtain more reliable results, we repeated each experiment 5 times and report here the average results.

Figure 3.4 shows utility in terms of edge changes and clustering coefficients. In these figures, the X-axis is the edge confidentiality threshold, which ranges from 0 to 1, in increment of 0.1. The Y-axis is the corresponding utility measurements, also ranges from 0 to 1, where 0 indicates the highest and 1 indicates the lowest utility.

Figures 3.4a and 3.4d show the RREC of EPINION and COA, respectively. In both figures, edge deletion outperforms edge swap. However, for COA, the difference between edge swap and



(a) RREC on EPINION (b) MDCC on EPINION (c) SDDCC on EPINION

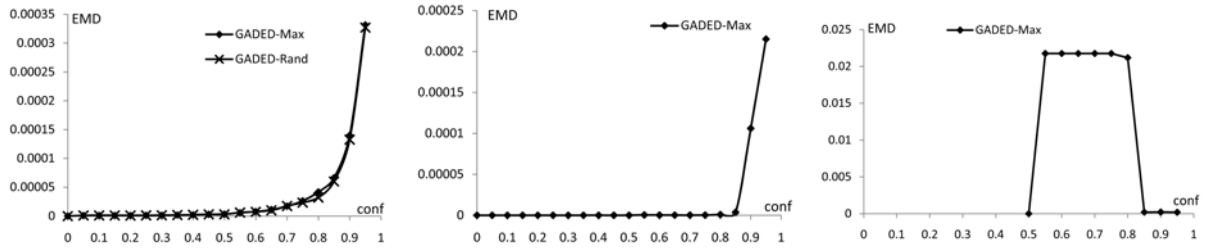
(d) RREC of COA (e) MDCC of COA (f) SDDCC of COA

Figure 3.4: RREC, MDCC and SDDCC of Anonymized Graphs

edge deletion is small for $\tau \leq 0.5$. We omitted the GADED-Rand in this figure because it is very similar to GADED-Max. For EPINION, the difference between edge swap and edge deletion is much bigger even for small τ. It also clearly shows that GADED-Max and GADED-Rand have almost identical performance. We also run the experiment on KDDCUP graph. Since the results are similar to those of COA, we do not show them here.

The remaining figures in Figure 3.4 show the results on clustering coefficient. They show that edge swap causes more changes of clustering coefficient than edge deletion does, and the mean change of clustering coefficient on EPINION graph is (about one order of magnitude) larger than on COA graph. The latter is expected because EPINION is denser, therefore, the change of an edge may change the cluster coefficients of more vertices in EPINION than in COA. In terms of standard deviation, edge swap and edge deletion are not much different on each dataset, but the standard deviation on COA is about twice as large as on EPINION, indicating a wider range of changes of clustering coefficient. It is yet to determine how such differences impact various graph applications.

Figure 3.5 shows the result on degree histogram change. Since edge swap never changes vertex



(a) EMD on EPINION (b) EMD on COA (c) EMD on KDDCUP

Figure 3.5: Earth Move Distance of Anonymized Graphs

degree, it has no impact on degree histogram, therefore, is not shown in the figure. For edge deletion, the results depend on datasets. For COA, the EMD is not noticeable for $\tau \leq 0.85$ and increased to 0.0002 for higher threshold. For EPINION, EMD starts to increase at $\tau = 0.6$ and increases more as the threshold becomes higher. But the largest EMD is still less than 0.00035. Thus for these two datasets, the algorithms seem to preserve the original degree distributions. On the other hand, the result of KDDCUP is quite different. For $0.5 \leq \tau \leq 0.8$, the EMD jumps to and stays at about 0.02, a much higher value as compared to the EMD of COA and EPINION. This suggests that the effect on degree distributions is data-dependent.

To summarize, edge swap performs better than edge deletion. As for utility, edge swap performs worse than edge deletion on edge changes and clustering coefficient, but better than edge deletion on maintaining degree distribution. The two versions of edge deletion have comparable performance with GADED-Rand being slightly better on higher confidentiality threshold.

3.5 Conclusions

This study presents a privacy notion of edge anonymity, the edge confidentiality, which captures privacy breach of an adversary who is able to use a type of equivalence relation to pinpoint VECs of target persons. The notion is general enough to allow any equivalence relation. We also consider a special type of edge anonymity problem that uses vertex degree to define the equivalence relation. We show that in real-world social graphs, especially those dense ones, edge disclosure can occur even if it is vertex k-anonymous. We present three heuristic algorithms to obtain τ-confident

graphs. Our experiments, based on three real-world social graphs and several utility measures, show that these algorithms can effectively protect edge anonymity and can produce anonymized graphs that have acceptable utility.

However, vertex degree is a simple structural information when used to define the equivalence relation, while an adversary could employ more complex structural information in the sensitive edge detection attack. In the next study, we will introduce neighbor-set equivalence to provide stronger protection for edge privacy.

Chapter 4: NEIGHBOR-BASED EDGE ANONYMIZATION

In this chapter, we present another edge anonymization method that still adopts edge confidentiality as privacy measure. We use neighbor-set to define the equivalence relation in the privacy measure, and a partition map in the graph in Section 4.1. Section 4.2 introduces two merge operations and their algorithms on the partition map. Section 4.3 presents the edge anonymity algorithm that constructs and executes merge plans on the partition map. Finally, we explain the results from our empirical study of the edge anonymity algorithm, and draw a conclusion.

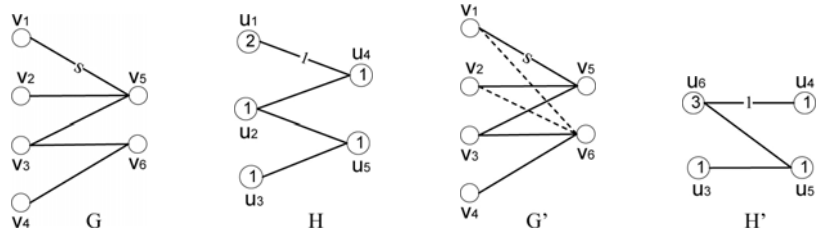
4.1 Neighbor-Set Equivalence and N-Partition Map

As mentioned in Chapter 2, many graph anonymization methods compute graph partitions based on various equivalence relations among vertices. The relative strength of these graph partitions depending on the equivalence relations determines the strength of privacy protection of these methods.

The study of equivalence relations in social graphs is not new. A number of equivalence relations of vertices have been extensively studied in sociology, to determine social roles played by actors in social networks. The structural equivalence, regarded as a pioneering work of social network analysis in [39], requires that two equivalent vertices have the identical connections with all other vertices in the social graph. The automorphism [27] is a generalization of structural equivalence and only requires equivalence rather than identical connections.

4.1.1 Neighbor-Set Induced Partition

Intuitively, two automorphic vertices are structurally indistinguishable from each other. This is why the method of [69] is able to prevent any vertex re-identification attack based on graph structure. However, the computational complexity of automorphism partition (or A-partition) of a graph is known to be neither polynomial nor NP-complete [40]. For large graphs, the computational cost can be prohibitively high. On the other hand, structurally equivalent vertices are also structurally



(a) Original graph. (b) The N-map (c) After $mbu(G|\{u_1, u_2\})$ (d) N-map after merging u_1 and u_2

Figure 4.1: A Running Example

indistinguishable from each other, but structural equivalence can be easily determined by checking the set of neighbors of vertices. Inspired by this idea, we define the following equivalence relation by slightly generalizing the structural equivalence.

Definition 4. *(***N-Partition***) The neighbor-set equivalence (or NE) relation of a graph G is $\sim_N$, such that, for any $u, v \in V(G)$, $u \sim_N v$, if and only if (iff) $NS(u) - \{v\} = NS(v) - \{u\}$, where $NS(u) = \{w|(u, w) \in E(G)\}$ is the set of neighbors of vertex u in G. The partition $P(G, \sim_N)$ is called the N-partition of G.*

Intuitively, two vertices are equivalent if the sets of their neighbors excluding each other are identical. This generalizes the structural equivalence in [39] by allowing two equivalent vertices to be neighbors to each other.

Example 2. Consider Figure 4.1, which is used as a running example in this chapter. Figure 4.1a is a simple graph G, which has a single sensitive edge (v_1, v_5), labeled by "s". Its N-partition contains VECs $\{v_1, v_2\}$, $\{v_3\}$, $\{v_4\}$, $\{v_5\}$, and $\{v_6\}$, and EECs $\{(v_1, v_5), (v_2, v_5)\}$, $\{(v_3, v_5)\}$, $\{(v_3, v_6)\}$, and $\{(v_4, v_6)\}$. The edge disclosure probabilities are $P[\{v_1, v_2\}, \{v_5\}] = 0.5$, $P[\{v_3\}, \{v_5\}] = 0$, $P[\{v_3\}, \{v_6\}] = 0$, and $P[\{v_4\}, \{v_6\}] = 0$. Thus, the edge confidentiality of G is 0.5.

We derive a theorem that supports us to use N-partition as an efficient manner in computing edge confidentiality. Before presenting the theorem, we first define the strength of partition, automorphism equivalence and its induced graph partition, then, prove a useful lemma.

Definition 5. *(***Stronger Partition***) A partition P_1 of a graph is stronger than another partition P_2 of the same graph if every VEC of P_1 is a subset of some VEC of P_2 .*

Definition 6. *(***Automorphism Equivalence***) An automorphism of a graph G is a bijection $\sigma : V(G) \to V(G)$, such that $(u, v) \in E(G)$ iff $(\sigma(u), \sigma(v)) \in E(G)$. Two vertices u and v of G are automorphism equivalent (or A-equivalent), written as $u \sim_A v$, iff there exists an automorphism mapping σ such that $u = \sigma(v)$ or $v = \sigma(u)$. The A-partition of G, denoted by $P(G, \sim_A)$, assigns two vertices to the same automorphism equivalence class (or AEC) iff they are A-equivalent to each other.*

Lemma 2. *The* N*-partition of any graph is stronger than its* A*-partition.*

Proof. We show that each AEC in $P(G, \sim_A)$ contains one or more VEC in $P(G, \sim_N)$, that is, for any $u, v \in V(G)$, if $u \sim_N v$, then $u \sim_A v$. Assume that $u \sim_N v$, let us consider a specific mapping $\sigma : V \to V$ such that $u = \sigma(v)$, $v = \sigma(u)$, and $w = \sigma(w)$ for all $w \in V(G)$ such that $w \neq u \neq v$. We show that such a mapping is an automorphism. To prove that $u \sim_A v$, we need to show that for any edge (x, y) in $E(G)$, the edge $(\sigma(x), \sigma(y))$ is also in $E(G)$.

Let (x, y) be an edge in $E(G)$. If none of u and v is in $\{x, y\}$, we simply have $(\sigma(x), \sigma(y)) = (x, y)$. Now assume without loss of generality that $x = u$ (therefore $y \neq u$). If $y = v$, then $(\sigma(x), \sigma(y)) = (\sigma(u), \sigma(v)) = (v, u)$. Because of $(x, y) = (u, v) \in E(G)$ and G is undirected, $(v, u) \in E(G)$. If $y = w \neq v$, then w is a neighbor of u and $(\sigma(x), \sigma(y)) = (\sigma(u), \sigma(w)) = (v, w)$. Since $u \sim_N v$, w is also a neighbor of v. That is, $(v, w) \in E(G)$. □

Theorem 2. If a graph is τ-confidential based on the N-partition, it is also τ-confidential based on the A-partition.

Proof. Assume the edge confidentiality based on *N*-partition is not less than τ, which according to Definition 2 means that for any $N_i, N_j \in P(G, \sim_N)$, $Pr[N_i, N_j] = P_{N_{ij}} \leq 1 - \tau$ or equivalently $\tau \leq 1 - \max\{P_{N_{hk}} | N_h, N_k \in P(G, \sim_N)\}$. By Lemma 2, each AEC $A_i \in P(G, \sim_A)$ contains

one or more VECs. We will show that for each $A_i, A_j \in P(G, \sim_A)$, $Pr[A_i, A_j] = P_{A_{ij}} \leq \max\{P_{N_{hk}} | N_h \subseteq A_i, N_k \subseteq A_j\}$, therefore,

$$\begin{aligned} \tau &\leq 1 - \max\{P_{N_{hk}} | N_h, N_k \in P(G, \sim_N)\} \\ &\leq 1 - \max\{P_{A_{ij}} | A_i, A_j \in P(G, \sim_A)\}. \end{aligned}$$

We need to consider two cases depending on if $A_i = A_j$.

Case 1: $A_i \neq A_j$.

By Eq. 3.3 and Eq. 3.2, $P_{A_{ij}} = \frac{\alpha_{ij}}{|A_i| \cdot |A_j|}$, where α_{ij} is the number of s-edges between A_i and A_j. Assume that A_i contains s VECs and A_j contains t. Since $A_i \cap A_j = \emptyset$, all VECs contained in A_i and A_j are distinct, thus $|A_i| \cdot |A_j| = \sum_{N_h \subseteq A_i, N_k \subseteq A_j} |N_h| \cdot |N_k|$. The α_{ij} s-edges can be partitioned among the $s \cdot t$ pairs of VECs between A_i and A_j, thus $\alpha_{ij} = \sum_{N_h \subseteq A_i, N_k \subseteq A_j} \alpha_{hk}$, where α_{hk} is the number of s-edges between VECs N_h and N_k. We can rewrite $P_{A_{ij}}$ as follows.

$$\begin{aligned} P_{A_{ij}} &= \frac{\sum_{N_h \subseteq A_i, N_k \subseteq A_j} \alpha_{hk}}{\sum_{N_h \subseteq A_i, N_k \subseteq A_j} |N_h| \cdot |N_k|} = \frac{\sum_{m=1}^{s \cdot t} a_m}{\sum_{m=1}^{s \cdot t} b_m} \\ &\leq \max\{\frac{a_m}{b_m} | 1 \leq m \leq s \cdot t\} \end{aligned} \tag{4.1}$$

where each m corresponding to a unique pair N_h and N_k, and $\frac{a_m}{b_m} = P_{N_{hk}}$. Thus, $\max\{\frac{a_m}{b_m} | 1 \leq m \leq s \cdot t\} = \max\{P_{N_{hk}} | N_h \in A_i, N_k \in A_j\}$. The inequality holds because if $\frac{a_1}{b_1} = \max\{\frac{a_m}{b_m} | 1 \leq m \leq s \cdot t\}$, then $\frac{a_1}{b_1} \geq \frac{a_m}{b_m}$ or $a_1 b_m \geq b_1 a_m$ holds for every $1 \leq m \leq s \cdot t$, and therefore,

$$\begin{aligned} \frac{a_1}{b_1} - \frac{\sum_{m=1}^{s \cdot t} a_m}{\sum_{m=1}^{s \cdot t} b_m} &= \frac{a_1 \cdot \sum_{m=1}^{s \cdot t} b_m - b_1 \cdot \sum_{m=1}^{s \cdot t} a_m}{b_1 \cdot \sum_{m=1}^{s \cdot t} b_m} \\ &= \frac{\sum_{m=1}^{s \cdot t} a_1 b_m - b_1 a_m}{b_1 \cdot \sum_{m=1}^{s \cdot t} b_m} \geq 0. \end{aligned}$$

Thus for the N_h and N_k that correspond to $\frac{a_1}{b_1}$, we have $P_{A_{ij}} \leq P_{N_{hk}} \leq 1 - \tau$. Since this is true

for every $P_{A_{ij}} \in P(G, \sim_A)$, we conclude that

$$\begin{aligned}
&\max\{P_{A_{ij}} | A_i, A_j \in P(G, \sim_A)\} \\
&\leq \max\{P_{N_{hk}} | N_h, N_k \in P(G, \sim_N)\} \\
&\leq 1 - \tau
\end{aligned}$$

and the theorem holds.

Case 2, $A_i = A_j$.

By Eq. 3.3 and Eq. 3.2, $P_{ij} = P_{A_{ii}} = \frac{\alpha_{ii}}{(|A_i| \cdot (|A_i|-1))/2}$, where α_{ii} is the number of s-edges among vertices in A_{ii}. Again, assume that A_i contains s VECs. Then,

$$\begin{aligned}
&\frac{|A_i| \cdot (|A_i| - 1)}{2} \\
&= \frac{1}{2}(\sum_{N_h \subseteq A_i} |N_h|) \cdot ((\sum_{N_k \subseteq A_i} |N_k|) - 1) \\
&= \frac{1}{2}((\sum_{N_h \subseteq A_i} |N_h| \cdot (\sum_{N_k \subseteq A_i} |N_k| - 1) \\
&= \frac{1}{2}(\sum_{N_h \subseteq A_i} (\sum_{N_k \subseteq A_i} |N_h| \cdot |N_k|) - |N_h|)) \\
&= \frac{1}{2}(\sum_{N_h \subseteq A_i} (\sum_{N_k \subseteq A_i, h \neq k} |N_h| \cdot |N_k| + |N_h| \cdot (|N_h| - 1)) \\
&= \frac{1}{2}(\sum_{N_h \subseteq A_i} \sum_{N_k \subseteq A_i, h \neq k} |N_h| \cdot |N_k| + \sum_{N_h \subseteq A_i} |N_h| \cdot (|N_h| - 1)) \\
&= \sum_{N_h, N_k \subseteq A_i, h \neq k} \frac{|N_h| \cdot |N_k|}{2} + \sum_{N_h \subseteq A_i} \frac{|N_h| \cdot (|N_h| - 1)}{2}
\end{aligned}$$

We can partition the α_{ii} s-edges among pairs of VECs in A_i, so that, $\alpha_{ii} = \sum_{N_h, N_k \subseteq A_i, h \neq k} \alpha_{hk} +$

$\sum_{N_h \subseteq A_i} \alpha_{hh}$. We can rewrite $P_{A_{ij}}$ as follows.

$$P_{A_{ij}} = \frac{\sum_{N_h, N_k \subseteq A_i, h \neq k} \alpha_{hk} + \sum_{N_h \subseteq A_i} \alpha_{hh}}{\sum_{N_h, N_k \subseteq A_i, h \neq k} \frac{1}{2} \cdot |N_h| \cdot |N_k| + \sum_{N_h \subseteq A_i} \frac{1}{2} \cdot |N_h| \cdot (|N_h| - 1)} = \frac{\sum_{m=1}^{s \cdot s} a_m}{\sum_{m=1}^{s \cdot s} b_m} \leq \max\{\frac{a_m}{b_m} | 1 \leq m \leq s \cdot s\}$$

where each m corresponds to a unique pair of VECs N_h and N_k, and if $h = k$, $\frac{a_m}{b_m} = P_{N_{hh}}$, otherwise, $\frac{a_m}{b_m} = \frac{1}{2} P_{N_{hk}}$. The inequality holds for the same reason as in case 1. Again, assume that $\frac{a_1}{b_1} = \max\{\frac{a_m}{b_m} | 1 \leq m \leq s \cdot s\}$. If $m = 1$ corresponds to N_h and N_k where $h = k$, we have $P_{A_{ij}} \leq P_{N_{hh}} \leq 1 - \tau$; otherwise (i.e., $h \neq k$), we have $P_{A_{ij}} \leq \frac{1}{2} P_{N_{hk}} \leq P_{N_{hk}} \leq 1 - \tau$, both imply that the theorem holds. □

The complexity of computing N-partition of a graph G is $O(h \cdot |V(G)|^2)$, where h is the size of the largest neighbor set. This is because the neighbor sets of all vertices can be found by scanning the edges in time of $O(|E(G)|)$. Using hashing, it takes $O(h)$ to determine if two neighbor sets are equal. Since each vertex needs to compare its neighbor set with the neighbor sets of other vertices, the number of pairs of vertices to compare is $O(|V(G)|^2)$.

4.1.2 N-Partition Map

To address the performance issue of anonymization methods referred in Section 1.4.1, we want to avoid re-computing the N-partition caused by every change in the underlying graph. For this purpose, we define the concept of a partition map.

Definition 7. *(**N-Partition Map**) The N-partition map (or N-map) of a graph G is a triple (H, ϕ, ψ), where H is the map, ϕ is a mapping from the graph to the map, and ψ is a mapping from the map to the graph. Specifically, H contains a vertex for each VEC and an edge for each EEC of G, $\phi : V(G) \to V(H)$, $\psi : V(H) \to 2^{V(G)}$, and they have the following properties.*

1. *Semantics of ϕ. For any $v, v' \in V(G)$ s.t. $v \neq v'$, $\phi(v) = \phi(v')$ if and only if (iff) $v \sim_N v'$.*

(a) *Semantics of* ψ. *For each* $u \in V(H)$, $\psi(u) \neq \emptyset$. *For any* $u, u' \in V(H)$ *s.t.* $u \neq u'$, $\psi(u) \cap \psi(u') = \emptyset$, $\cup_{u \in V(H)} \psi(u) = V(G)$. *And for any* $v \in V(G)$, $v \in \psi(u)$ *iff* $\phi(v) = u$.

(b) *Edge Correspondence. For each edge* $(u, u') \in E(H)$, *there is an edge* $(v, v') \in E(G)$ *for every pair of vertices* $v \in \psi(u)$ *and* $v' \in \psi(u')$ *s.t.* $v \neq v'$; *for each edge* $(v, v') \in E(G)$, *there is an edge* $(\phi(v), \phi(v')) \in E(H)$.

(c) *Label Consistency. Each vertex* $u \in V(H)$ *has a label* l_u *that has the value* $|\psi(u)|$, *and each edge* $(u, u') \in E(H)$ *has a label* $l_{(u,u')}$ *that has the value* $|S(u, u')|$, *where* $S(u, u')$ *is the set of s-edges between* $\psi(u)$ *and* $\psi(u')$.

We call G *the (underlying) graph of* (H, ϕ, ψ).

Intuitively, the N-map is a graph representation of the N-partition with each vertex in the map being an VEC in the graph. Properties 1 and 2 guarantee that ϕ maps every vertex in G to its VEC in H and ψ maps every VEC in H to its set of equivalent vertices in G. Property 3 guarantees that if two vertices $v \in \psi(u)$ and $v' \in \psi(u')$ are neighbors in G, the corresponding VECs $\phi(v) = u$ and $\phi(v') = u'$ are also neighbors in H and vice versa. Notice that H may have some edge from an VEC to itself.

For convenience, we define the ϕ and ψ mappings of sets of vertices as $\phi(S) = \cup_{w \in S}\{\phi(w)\}$ and $\psi(S') = \cup_{w \in S'}\psi(w)$, where $S \subseteq V(G)$ and $S' \subseteq V(H)$. Also, we use $NS(\phi(v))$ to denote all neighbors of VEC $\phi(v)$ in the N-map. Whenever the ϕ and ψ mappings are understood, we simply refer H as the N-map.

The N-map of a graph can be obtained by computing the N-partition, creating a unique vertex for each VEC and a unique edge for each EEC, and updating the two mappings and edge labels as vertices and edges are created. It is straightforward to obtain the underlying graph from a given N-map.

Example 3. The N-map of the graph in Figure 4.1a consists of the graph H in Figure 4.1b. The number inside each circle is the vertex label and the number on the edge is the edge label. The

mapping ϕ is $\phi(v_1) = \phi(v_2) = u_1$, $\phi(v_3) = u_2$, $\phi(v_4) = u_3$, $\phi(v_5) = u_4$, $\phi(v_6) = u_5$. The mapping ψ is $\psi(u_1) = \{v_1, v_2\}$, $\psi(u_2) = \{v_3\}$, $\psi(u_3) = \{v_4\}$, $\psi(u_4) = \{v_5\}$, $\psi(u_5) = \{v_6\}$.

The intended use of the N-map is to allow the anonymization of the underlying graph be performed on the N-map, so that, the computation is more efficient. To measure the edge confidentiality of the graph from its N-map, we define the edge disclosure probability of an edge in the N-map.

Definition 8. (**Unsatisfied Degree**) Let (H, ϕ, ψ) be the N-map of a graph G, $e = (u, u')$ be an edge (H, ϕ, ψ), and τ be a threshold of the edge confidentiality. The edge disclosure probability of e is $P(e) = P[\psi(u), \psi(u')]$ based on Definitions 2 and 7,

$$P(e) = P[\psi(u), \psi(u')] = \begin{cases} \frac{l_e}{0.5 \cdot l_u \cdot (l_{u'} - 1)}, & \text{if } u = u'; \\ \frac{l_e}{l_u \cdot l_{u'}}, & \text{otherwise.} \end{cases}$$

An edge e in the N-map is unsatisfied if $P(e) > 1 - \tau$. The unsatisfied degree of a vertex u of the N-map, denoted by $ud(u)$, is the number of unsatisfied edges incident on (or covered by) the vertex. The ratio of unsatisfied degree of u is $rud(u) = \frac{ud(u)}{d(u)}$, where $d(u)$ is the degree of u.

Example 4. Assume that the threshold of edge confidentiality is 0.7. Then, edge (u_1, u_4) in Figure 4.1b is unsatisfied, since $P((u_1, u_4)) = 0.5 > 1 - 0.7 = 0.3$. Then, $ud(u_4) = 1$ and $rud(u_4) = 0.5$. Similarly, $rud(u_1) = 1$ and $rud(u_2) = rud(u_3) = rud(u_5) = 0$.

4.2 Merges of VECs in an N-Map

In a change-and-check style anonymization method, it is more efficient to take a set-at-a-time approach instead of the edge-at-a-time approach. By adding/removing an appropriate set of edges, two (or more) VECs can be merged into a new VEC, i.e., all vertices in these VECs become equivalent. This will change some EECs and may improve the edge confidentiality of the graph. Since vertices of the N-map are VECs of the underlying graph, it is natural to define the merge of VECs of a graph as the merge of vertices in its N-map.

Definition 9. *(***VEC-Merge***) Let (H, ϕ, ψ) be the N-map of a graph G and M be a merge-set containing two or more vertices in $V(H)$. An VEC-merge of M is an operation that results in the N-map of a new graph G', such that*

1. $V(G) = V(G')$;
2. For each pair of vertices $v, v' \in V(G')$ such that $v \notin \psi(M)$ and $v' \notin \psi(M)$, $(v, v') \in E(G')$ iff $(v, v') \in E(G)$;
3. There is a non-empty set $W \subseteq V(G')$ such that $\forall v \in \psi(M)$, $NS(v) = W - \{v\}$ in G'.

Intuitively, condition 1 ensures that no vertex is added or removed. Condition 2 ensures that edges involving no vertex in $\psi(M)$ are not added or removed. Condition 3 ensures that a set of edges are added or removed to make W the common neighbors of the vertices in $\psi(M)$.

4.2.1 Two Types of VEC-Merges

There are many possible choices of the common neighbors, W, thus many different types of VEC-merges. To minimize the edges added or removed, the common neighbors should be selected from the neighbors of the input vertices. We consider two special VEC-merges.

Definition 10. (**Merge-by-Union**) *A merge-by-union of M (or $mbu(M)$) is an VEC-merge with*

$$W = \begin{cases} \psi(\cup_{u \in M} NS(u)), & \text{if } \cup_{u \in M} NS(u) \cap M = \emptyset; \\ \psi(\cup_{u \in M} NS(u) \cup M), & \text{otherwise;} \end{cases}$$

Notice that the common neighbors may include some vertex in $\psi(M)$. A merge-by-union makes all neighbors of any input vertex the common neighbors of all input vertices. It will only add edges to the underlying graph.

Example 5. After the merge-by-union $mbu(\{u_1, u_2\})$ is performed on the N-map (see Figure 4.1b) of the graph G (see Figure 4.1a), where $\{u_1, u_2\}$ is the merge set, we obtain the new N-map (see

Figure 4.1d) of a new underlying graph G' (see Figure 4.1c). In G', vertices v_1, v_2, and v_3 are equivalent. Also notice that the common neighbors W in this merge is $\{v_5, v_6\}$. The new mapping ϕ' is $\phi'(v_1) = \phi'(v_2) = \phi'(v_3) = u_6$, $\phi'(v_4) = u_3$, $\phi'(v_5) = u_4$, $\phi'(v_6) = u_5$. The new mapping ψ' is $\psi'(u_6) = \{v_1, v_2, v_3\}$, $\psi'(u_3) = \{v_4\}$, $\psi'(u_4) = \{v_5\}$, $\psi'(u_5) = \{v_6\}$.

Definition 11. (**Merge-by-Intersection**) *A merge-by-intersection of M (or $mbi(M)$) is an VEC-merge with*

$$W = \begin{cases} \psi(\cap_{u \in M} NS(u)), & \text{if } M \subseteq \cap_{u \in M} NS(u); \\ \psi(\cap_{u \in M} NS(u) - M), & \text{otherwise.} \end{cases}$$

A merge-by-intersection only retains the neighbors common to all input vertices and therefore will only remove edges from the underlying graph.

4.2.2 Algorithms of VEC-Merges

Algorithm 4.1 performs a merge-by-union. The main idea is to add as few edges as possible to the input N-map and update the two mappings accordingly so that in the resulting underlying graph, the set of common neighbors of vertices in $\psi(M)$ is exactly the W in Definition 10. This can be achieved by identifying the corresponding set $\phi(W)$ of common neighbors of M in the N-map. The set $\phi(W)$ may contain vertices that are already neighbors of every vertex in M, or vertices that are not yet neighbors of all vertices in M. We need to update the mappings at least involving the vertices in M and vertices that are not yet neighbors of all vertices in M.

Step 1 of Algorithm 4.1 identifies the common neighbors $\phi(W)$ in the N-map based on Definition 10. Step 2 adds new edges to the N-map between vertices in M and vertices in $\phi(W)$ so that all vertices in M have the same neighbor sets. These edges correspond to edges to be added into the underlying graph, which if added, will results in the new underlying graph G'. However, at this point, the map has been changed but the mappings have not. So, the N-map is neither the N-map of the graph G nor that of the graph G'. The remaining steps of the algorithm is to update mappings

Algorithm 4.1 Merge-by-Union

Input: N-map (H, ϕ, ψ) of graph G,
a non-empty set $M \subseteq V(H)$
Output: (H', ϕ', ψ') satisfying Definition 10.

1. compute $\phi(W)$ for $mbu(M)$;
2. for $u \in M$ and $u' \in \phi(W)$ s.t. $(u, u') \notin E(H)$ do
   if ($u \neq u'$ or $|\psi(u)| > 1$) then
   add (u, u') to $E(H)$ with label 0;
3. $T = (\phi(W) - \cap_{u \in M} NS(u)) \cup M$;
4. while there is a vertex u in T do
5. $U = \{u' | u' \in V(H) \wedge (\exists v \in \psi(u), v' \in \psi(u')($
   $\psi(NS(u)) - \{v, v'\} = \psi(NS(u')) - \{v, v'\}))\}$;
6. if $|U| > 1$ then
7. add a new vertex w into $V(H)$ with label
   $l_w = \sum_{u \in U} l_u$;
8. if $\cup_{s \in U} NS(s) \cap U \neq \phi$ then
   add edge (w, w) into $E(H)$ with label
   $l_{(w,w)} = \sum_{u,u' \in U \wedge (u,u') \in E(H)} l_{(u,u')}$;
9. for each $u' \in \cup_{s \in U} NS(s) - U$ do
   add edge (w, u') into $E(H)$ with label
   $l_{(w,u')} = \sum_{u \in U \wedge (u,u') \in E(H)} l_{(u,u')}$;
10. set $\psi(w) = \psi(U)$ and $\forall v \in \psi(w)$, set $\phi(v) = w$;
11. for each $u \in U$ do
    remove $\psi(u)$, u, and any edge incidents on u;
12. $T = T - U$;
13. return (H, ϕ, ψ);

to obtain the correct N-map of G'. Step 3 identifies the set T that contains of vertices in M and vertices that are in $\phi(W)$ but are not yet common neighbors of M. Due to edges added in Step 2, the neighbors of these vertices have been changed and some of them may become equivalent to each other, therefore need to be merged into the same VEC. Step 4-12 is to modify the N-map based on the vertices identified in Step 3. Step 5 takes a vertex u in T and identifies a set of vertices U in the N-map that are candidates for merging with u. The set U always contains u. In particular, if vertex u is in M, it will also contains every vertex in M. In addition, U also contains vertices whose new neighbor sets (resulted from edges being added in Step 2) indicate that their corresponding VECs should be merged with the VEC represented by u. Step 6 verifies that a merge of vertices in U is feasible. To perform a merge on vertices in U, Step 7 adds a new vertex w into the N-map. Step 8 adds a self-loop edge to w if VECs in U are neighbors to each other. Step 9 adds edges between w and other neighbors of vertices in U. Since each of these new edges will replace multiple existing edges, its label is the sum of labels of the replaced edges. Step 10 updates the ϕ and ψ mappings for w. Step 11 cleans up the replaced mappings and removes the replaced vertices and edges. Step 12 removes all vertices in U from T. Step 13 returns the new N-map.

Example 6. Figure 4.1d is the graph of the new N-map after $mbu(\{u_1, u_2\})$ on the N-map in Figure 4.1b using Algorithm 4.1. The new mappings are exactly those given in Example 5. The new vertex u_6 is created to replace vertices u_1 and u_2 in the resulted N-map H'. Notice that u_6 is labeled with 3, the sum of the labels of u_1 and u_2. As a result, the edge disclosure probability on edge (u_6, u_3) becomes $P((u_{6,}u_4)) = \frac{1}{3}$.

Algorithm 4.2 Merge-by-Intersection

Input: N-map (H, ϕ, ψ) of graph G,
and a non-empty set $M \subset V(H)$
Output: (H', ϕ', ψ') that satisfies Definition 11.
1. compute $\phi(W)$ for $mbi(M)$;
2. for $u \in M$ and $u' \notin \phi(W)$ s.t. $(u, u') \in E(H)$ do
if ($u \neq u'$ or $|\psi(u)| > 1$), remove (u, u') from $E(H)$;
3-12 Same as Algorithm 4.1
13. return (H, ϕ, ψ);

Algorithm 4.2 performs a merge-by-intersection. It is almost identical to Algorithm 4.1 except Steps 1 and 2.

The complexity of Algorithms 4.1 and 4.2 are both $O(k \cdot (|V(H)| \cdot |E(H)| + |V(G)|))$, where k is number of vertices in T. The analysis is as follows. In Step 1, the complexity of finding common neighbors of merge-set M on the N-map is $O(|E(H)|)$. In Step 2, the complexity of adding (or removing) edges between M and $\phi(W)$ is $O(|M| \cdot |\phi(W)|)$. In Step 3, the set subtraction and union operations can be performed in $O(|V(H)|)$ using hashing. In Step 5 the vertices in U can be found in $O(|V(H)| \cdot |E(H)|)$. This is because that due to the edge correspondence property of the N-map, a vertex in $\psi(u)$ is equivalent to a vertex in $\psi(u')$ in the underlying graph iff u and u' have the same set of neighbors (therefore, is also NE) in the N-map. Thus, vertices in U can be identified by checking neighbor sets of vertices in H. In Step 7, it takes $O(|U|)$ to calculate the sum of labels of vertices in U. In Step 8, it takes $O(|E(H)|)$ to check for edges that among vertices of U and to sum up labels of those edges. In Steps 9 and 11, the adding and removing edges on H each takes $O(|U| \cdot |E(H)|)$. In Step 10, it takes $O(|V(G)|)$ to update the ψ and ϕ mappings for each vertex in U. In Step 12, the set subtraction takes $O(|U|)$. Notice that M, $\phi(W)$, and U are all subsets of $V(H)$. According to this analysis, if H is very small, the complexity will be dominated by the updating of the two mappings, and if H is large, the complexity will be dominated by the construction of U.

4.2.3 Correctness of VEC-Merges Algorithms

In this section, we prove the correctness of Algorithm 4.1 and Algorithm 4.2. For this purpose, we first prove that both Algorithm 4.1 and Algorithm 4.2 indeed return the N-maps that satisfies Definitions 10 and 11, respectively. The basic idea of the proof is to show that some properties, similar to those in Definition 7, hold at various steps of Algorithms 4.1 and 4.2. The proof requires a definition and a set of lemmas, which we give first. Finally, we present a theorem about the correctness of VEC-Merges algorithms.

Let (H, ϕ, ψ) be the N-map of G and G' be the new underlying graph specified in Definition

Table 4.1: Differences of N-map in various Steps: "—" in any step denotes no change in that step.

Step i	$V(H_i)$	$E(H_i)$	ϕ_i	ψ_i
1	$V(H)$	$E(H)$	ϕ	ψ
2	—	add edges (Alg.4.1) remove edges (Alg. 4.2)	—	—
3 - 6	—	—	—	—
7	add w	—	—	—
8	—	may add (w, w)	—	—
9	—	may add (w, u')s	—	—
10	—	—	reset $\phi(v) = w$ for $v \in \psi_4(U)$	add $\psi(w) =$ $\psi_4(U)$
11	remove U	remove edges	—	remove $\psi_4(u)$ for $u \in U$
12-13	—	—	—	—

10 (or Definition 11 for Alg. 4.2[1]). Although at many steps of both algorithms, the triple (H, ϕ, ψ) may not be the N-map of either the graph G or the graph G', we still refer it as an N-map (actually, the N-map under transition) to keep the description simple. However, we use (H_i, ϕ_i, ψ_i) to denote the snapshot of this N-map in Step i of the algorithm. Table 4.1 on page 47 shows changes made to the N-map at each step in the algorithms. We note that the only difference between Algorithm 4.1 and Algorithm 4.2 is on row 2 of the table. Similarly, we use $S_i(u, u')$ to denote the set of sensitive edges between $\psi_i(u)$ and $\psi_i(u')$.

Definition 12. Each of the following properties is said to hold in Step i of Algorithm 4.1 or Algorithm 4.2, if the corresponding conditions between (H_i, ϕ_i, ψ_i) and G' are satisfied.

1. The semantics of ϕ. $\forall u \in V(H_i)$, $\exists v \in V(G')$, $\phi_i(v) = u$ and $\forall v' \in V(G')$ s.t. $v' \neq v$, $\phi_i(v') = u$ iff $v \sim_N v'$.

2. The semantics of ψ. $\forall u \in V(H_i)$, (a) $\psi_i(u) \neq \emptyset$, (b) $\forall u' \in V(H_i)$ s.t. $u' \neq u$, $\psi_i(u) \cap \psi_i(u') = \emptyset$, (c) $\cup_{u \in V(H_i)} \psi_i(u) = V(G')$, and (d) $\forall v \in V(G')$, $v \in \psi_i(u)$ iff $\phi_i(v) = u$.

[1]Since the proof for Algorithm 4.1 is almost identical to that of Algorithm 4.2, in the remaining of this section, we focus on Algorithm 4.1, and note the difference for Algorithm 4.2 in parenthesis wherever appropriate.

3. The edge correspondence. (a) $\forall(u, u') \in E(H_i)$, $\forall v \in \psi_i(u), \forall v' \in \psi_i(u')$ s.t. $v \neq v'$, $(v, v') \in E(G')$; and (b) $\forall(v, v') \in E(G')$, $(\phi_i(v), \phi_i(v')) \in E(H_i)$.

4. The label consistency. $\forall u \in V(H_i)$, $l_u = |\psi_i(u)|$ and $\forall(u, u') \in E(H_i)$, $l_{(u,u')} = |S_i(u, u')|$.

The meaning should also be obvious if we say that one of these properties holds on a vertex or an edge.

Lemma 3. *In both Algorithms 4.1 and 4.2, the semantics of ψ, the edge correspondence, and the label consistency hold in Step 3.*

Proof. We consider the two algorithms separately.

(***For Algorithm 4.1***): We have the following observations in Step 3: A) the input (H, ϕ, ψ) is the N-map of G, which satisfies Definition 7; B) by Definitions 9 and 10, $V(G) = V(G')$ and $E(G) \subset E(G')$; and C) according to Table 4.1, $V(H_3) = V(H)$, $E(H) \subseteq E(H_3)$, $\phi_3 = \phi$ and $\psi_3 = \psi$.

(1) Consider the semantics of ψ. Consider any $u \in V(H_3)$. Because of the aforementioned observations, (a) $\psi(u) \neq \emptyset$ implies $\psi_3(u) \neq \emptyset$; (b) $\forall u' \in V(H_3)$ s.t. $u' \neq u$, $\psi(u) \cap \psi(u') = \emptyset$ implies $\psi_3(u) \cap \psi_3(u') = \emptyset$; and (c) $\forall v \in V(G')$, $v \in \psi(u)$ iff $\phi(v) = u$ implies $v \in \psi_3(u)$ iff $\phi_3(v) = u$.

(2) Consider edge correspondence. For any $(u, u') \in E(H_3)$, either $(u, u') \in E(H)$ or $(u, u') \notin E(H)$.

If $(u, u') \in E(H)$, by edge correspondence of Definition 7, $\forall v \in \psi(u)$ and $\forall v' \in \psi(u')$, $(v, v') \in E(G)$, thus by aforementioned observations B and C, $v \in \psi_3(u)$, $v' \in \psi_3(u')$ and $(v, v') \in E(G')$.

If $(u, u') \notin E(H)$, it must be added in Step 2. Without lose of generality, let $u \in M$ and $u' \in \phi(W)$ in H. Since $\phi_3 = \phi$, $\psi_3 = \psi$ and $V(G) = V(G')$, for any $v \in \psi_3(u)$ and $v' \in \psi_3(u')$ in G', we must have $v \in \psi(u)$ and $v' \in \psi(u')$ in G. Because $\psi(u) \subseteq \psi(M)$ and $\psi(u') \in W$, by Definition 9, (v, v') is in $E(G)$. Since $E(G) \subset E(G')$, (v, v') is in $E(G')$, too.

Consider any edge $(v, v') \in E(G')$, where $v \in \psi_3(u)$, $v' \in \psi_3(u')$ for some $u, u' \in V(H_3)$. Because $V(G) = V(G')$, v and v' are both in $V(G)$. The edge (v, v') may or may not be in $E(G)$. If $(v, v') \in E(G)$, by Definition 7, $(\phi(v), \phi(v')) \in E(H)$. Since $E(H) \subseteq E(H_3)$ and $\phi_3 = \phi$ by Table 4.1, $(\phi_3(v), \phi_3(v')) \in E(H_3)$.

If $(v, v') \notin E(G)$, it must be added by $mbu(M)$. By Definition 9 and without loss of generality, let $v \in \psi(M)$ and $v' \in W$. By Definition 7, $\exists u \in M$ and $\exists u' \in \phi(W)$, s.t., $\phi(v) = u$ and $\phi(v') = u'$. According to the algorithm, (u, u') is added in Step 2 implying $(u, u') \in E(H_3)$. Because $\phi_3 = \phi$, $(\phi_3(v), \phi_3(v')) \in E(H_3)$.

(3) Consider label consistency. Consider any $u \in V(H_3)$. By observation A, $l_u = |\psi(u)|$. By observations B and C, $u \in V(H)$ and $\psi(u) = \psi_3(u)$. Since there is no change to any vertex label in Steps 1 through 3, $l_u = |\psi_3(u)|$. For each edge $(u, u') \in E(H_3)$, we consider two cases: $(u, u') \in E(H)$ and $(u, u') \notin E(H)$. If $(u, u') \in E(H)$, because $\psi_3 = \psi$, edge correspondence holds in Steps 1 through 3, and there is no change to edge labels in Steps 1 through 3, $l_{(u,u')} = |S(u, u')| = |S_3(u, u')|$. If $(u, u') \notin E(H)$, (u, u') is added in Step 2. The added edges are counterfeit, the $l_{(u,u')} = 0$ is correct label added in Step 2.

(***For Algorithm 4.2***): Observe that in Step 3, A) the input (H, ϕ, ψ) is the N-map of G, which satisfies Definition 7; B) by Definitions 9 and 11, $V(G) = V(G')$ and $E(G) \supset E(G')$; and C) according to Table 4.1, $V(H_3) = V(H)$, $E(H) \supseteq E(H_3)$, $\phi_3 = \phi$ and $\psi_3 = \psi$.

(1) Consider the semantics of ψ. The proof is the same as for Algorithm ***4.1***.

(2) Consider edge correspondence. For any $(u, u') \in E(H_3)$, $(u, u') \in E(H)$ because $E(H) \supseteq E(H_3)$.

If $(u, u') \in E(H)$, by edge correspondence of Definition 7, $\forall v \in \psi(u)$ and $\forall v' \in \psi(u')$, $(v, v') \in E(G)$. Because $\psi_3 = \psi$, we have $v \in \psi_3(u)$ and $v' \in \psi_3(u')$. Then we only need to show $(v, v') \in E(G')$. We prove it by contradiction. Assume that $(v, v') \notin E(G')$. Then the edge (v, v') must be removed by $mbi(M)$. By Definition 9 and without loss of generality, let $v \in \psi(M)$ and $v' \notin W$. By Definition 7, $\phi(v) = u \in M$ and $\phi(v') = u' \notin \phi(W)$. According to Algorithm 4.2, (u, u') is removed in Step 2 implying $(u, u') \notin E(H_3)$, contradicting with $(u, u') \in E(H_3)$.

Consider any edge $(v, v') \in E(G')$. Because $V(G) = V(G')$ and $E(G) \supseteq E(G')$, $(v, v') \in E(G)$. By Definition 7, $(\phi(v), \phi(v')) \in E(H)$. Then we need to show $(\phi_3(v), \phi_3(v')) \in E(H_3)$. We also prove it by contradiction. Assume that $(\phi_3(v), \phi_3(v')) \notin E(H_3)$. Because $\phi_3 = \phi$, $(\phi(v), \phi(v')) \notin E(H_3)$. Then $(\phi(v), \phi(v'))$ must be removed in Step 2. According to the algorithm, $(\phi(v), \phi(v'))$ is removed in Step 2 implying that $\phi(v) \in M$ and $\phi(v') \notin \phi(W)$ in H. So we get $v \in \psi(M)$, $v' \notin W$. Then (v, v') must be removed by $mbi(M)$ implying that $(v, v') \notin E(G')$, contradicting with $(v, v') \in E(G')$.

(3) Consider label consistency. Consider any $u \in V(H_3)$. By observation A, $l_u = |\psi(u)|$. By observations B and C, $u \in V(H)$ and $\psi(u) = \psi_3(u)$. Since there is no change to any vertex label in Steps 1 through 3, $l_u = |\psi_3(u)|$. For each edge $(u, u') \in E(H_3)$, $(u, u') \in E(H)$ by observation C. Since $\psi_3 = \psi$, edge correspondence holds in Steps 1 through 3, and there is no change to edge labels in Steps 1 through 3, $l_{(u,u')} = |S(u, u')| = |S_3(u, u')|$. □

For the following lemmas, the proofs are identical for both algorithms. Therefore, we do not distinguish between the algorithms in the proofs.

Lemma 4. *In both Algorithms 4.1 and 4.2, if the semantics of ψ holds in Step 4, it also holds in Step 12.*

Proof. We show that conditions 2.(a) to 2.(d) of Definition 12 are satisfied in Step 12.

Consider condition 2.(a) and 2.(d). For any vertex $u \in V(H_{12})$, either $u \in V(H_4)$ or $u \notin V(H_4)$.

If $u \in V(H_4)$, according to Table 4.1, $u \neq w$ and $u \notin U$, where w is the new vertex added in Step 7 and U is a set of vertices removed in Step 11. Therefore, $\psi_4(u) = \psi_{12}(u)$ and $\forall v \in \psi_4(u)$, $\phi_4(v) = \phi_{12}(v)$. Since $\psi_4(u) \neq \emptyset$ by assumption, thus $\psi_{12}(u) \neq \emptyset$; also $\forall v \in V(G')$, since $\phi_4(v) = u$ iff $v \in \psi_4(u)$ by assumption, thus $\phi_{12}(v) = u$ iff $v \in \psi_{12}(u)$.

If $u \notin V(H_4)$, it follows $u = w$, as well as $|U| > 1$ in Step 6. By the assumption, $\forall u' \in U$, $\psi_4(u') \neq \emptyset$, therefore, $\psi_4(U) \neq \emptyset$. Since $\psi_{10}(w) = \psi_4(U)$, it follows $\psi_{10}(w) \neq \emptyset$. Also, $\forall v \in \psi_{10}(w)$, $\phi_{10}(v) = w$. Because by Table 4.1, $\psi_{12}(w) = \psi_{10}(w)$ and $\phi_{12} = \phi_{10}$, as a result,

$\psi_{12}(w) \neq \emptyset$ and $\phi_{12}(v) = w$. Since $\forall u \in U$, $\psi_{11}(u)$ is removed, as a result, $\forall v \in \psi_{11}(U)$, v only belongs to $\psi_{12}(w)$. So $\phi_{12}(v) = w$ iff $v \in \psi_{12}(w)$.

Consider condition 2.(b). Consider any pair of vertices $u, u' \in V(H_{12})$. Since only one new vertex is added between Steps 4 and 12, at most one of u and u' can be the new vertex. If neither is the new vertex, they are in both $V(H_4)$ and $V(H_{12})$, which as argued previously means $\psi_4(u) = \psi_{12}(u)$ and $\psi_4(u') = \psi_{12}(u')$. Since by assumption, $\psi_4(u) \cap \psi_4(u') = \emptyset$, it follows $\psi_{12}(u) \cap \psi_{12}(u') = \emptyset$. If one vertex, say u', is new vertex w, by Table 4.1, $\psi_{10}(u') = \psi_4(U)$ and $\psi_{10}(u) = \psi_4(u)$. By assumption, $\psi_4(u) \cap \psi_4(U) = \emptyset$. So $\psi_{10}(u) \cap \psi_{10}(w) = \emptyset$. Because $\psi_{12}(w) = \psi_{10}(w)$ and $\psi_{12}(u) = \psi_{10}(u)$, it follows $\psi_{12}(u) \cap \psi_{12}(w) = \emptyset$.

Consider condition 2.(c). We show $\cup_{u \in V(H_{12})} \psi_{12}(u) = V(G')$. Step 12 can be reached from Step 4 via Step 6 or Step 11. If it is via Step 6, $\cup_{u \in V(H_{12})} \psi_{12}(u) = \cup_{u \in V(H_4)} \psi_4(u) = V(G')$ by the assumption. If it is via Step 11, $V(H_{12}) = V(H_4) - U + \{w\}$. By Table 4.1, $\forall u \in V(H_{12})$, if $u \in V(H_4) - U$, $\psi_{12}(u) = \psi_{10}(u) = \psi_4(u)$; and if $u = w$, $\psi_{12}(u) = \psi_{10}(u) = \psi_4(U)$. So $\cup_{u \in V(H_{12})} \psi_{12}(u) = \cup_{u \in V(H_4) - U + \{w\}} \psi_{12}(u) = \cup_{u \in V(H_4) - U} \psi_4(u) \cup \psi_4(U) = \cup_{u \in V(H_4)} \psi_4(U) = V(G')$. □

Lemma 5. *In both Algorithms 4.1 and 4.2, if the semantics of ψ and the edge correspondence hold in Step 4, then in Step 5, (a) $\forall u, u' \in U, \forall v \in \psi_5(u), \forall v' \in \psi_5(u')$, $(v \sim_N v')$ in G'; (b) $\forall v, v' \in V(G')$, if $(v \sim_N v')$ then either both $\phi_5(v)$ and $\phi_5(v')$ are in U or both not in U.*

Proof. By assumption, the semantics of ψ and the edge correspondence hold in Step 4. By Table 4.1, $\psi_4 = \psi_5$ and $E(H_4) = E(H_5)$. Thus the semantics of ψ and the edge correspondence also hold in Step 5. Therefore, for each $v \in G'$, $\exists u \in V(H_5)$ s.t. $v \in \psi_5(u)$ and $NS(v) = \psi_5(NS(u)) - \{v\}$. Consequently, $\forall u \in V(H_5), \forall v, v' \in \psi_5(u)$, $v \sim_N v'$ in G'.

For (a), consider any pair of $u, u' \in U$ in Step 5. By definition of U, $\exists v \in \psi_5(u), \exists v' \in \psi_5(u')$ s.t., $\psi_5(NS(u)) - \{v, v'\} = \psi_5(NS(u')) - \{v, v'\}$, or equivalently, $v \sim_N v'$ in G'. This is because $NS(v) - \{v'\} = \psi_5(NS(u)) - \{v, v'\} = \psi_5(NS(u')) - \{v, v'\} = NS(v') - \{v\}$. Since all vertices in $\psi_5(u)$ are equivalent to v and those in $\psi_5(u')$ to v', $v \sim_N v'$ implies that every vertex in $\psi_5(u)$ is equivalent to every vertex in $\psi_5(u')$.

For (b), consider any pair of $v, v' \in V(G')$ s.t. $(v \sim_N v')$. Without loss of generality, let $\phi_5(v) = u \in U \subseteq V(H_5)$. By assumption of the semantics of ψ, $v \in \psi_5(u)$. Let $\phi_5(v') = u' \in V(H_5)$. By the semantics of ψ, $v' \in \psi_5(u')$. Since $v \sim_N v'$, by the semantics of ψ and edge correspondence, $\psi_5(NS(u)) - \{v, v'\} = NS(v) - \{v'\} = NS(v') - \{v\} = \psi_5(NS(u')) - \{v, v'\}$. So by definition of U, $\phi_5(v') = u' \in U$. □

Lemma 6. *In both Algorithms 4.1 and 4.2, if the semantics of ψ and edge correspondence hold in Step 4, and $\cup_{u \in U} NS(u) \cap U \neq \emptyset$ in H_8, then $\forall v \in \psi_8(U)$ in G', $NS(v) \supseteq \psi_8(U) - \{v\}$.*

Proof. In both algorithms, U is computed in Step 5 and stays unchanged through Step 8. Thus $w \notin U$ and $\forall u \in U$, $u \in V(H_4)$. By Table 4.1, $\forall u \in V(H_4)$, $\psi_4(u) = \psi_i(u)$ for $i = 5, 6, 7, 8$ and $E(H_4) = E(H_i)$ for $i = 5, 6, 7$. Also, $E(H_8) = E(H_4)$ or $E(H_8) = E(H_4) \cup \{(w, w)\}$. As a result, the semantics of ψ holds on every vertex in $V(H_8)$ other than w and the edge correspondence holds on each edge in $E(H_8)$ other than (w, w).

Since $\cup_{u \in U} NS(u) \cap U \neq \emptyset$ in H_8, there must be at least one edge $(u', u'') \in E(H_8)$ (i.e. $u' \in NS(u'')$ and $u'' \in NS(u')$) for some (not necessarily distinct) vertices $u', u'' \in U$. Since $w \notin U$, it follows that $u' \neq w$ and $u'' \neq w$. Thus by edge correspondence, $\exists (s', s'') \in E(G')$ s.t. $s' \in \psi_8(u')$ and $s'' \in \psi_8(u'')$, therefore, $s \in NS(s')$ (and also $s' \in NS(s'')$).

Now, consider some vertex $v \in V(G')$ s.t. $v \in \psi_8(U)$. Since $\psi_8(U) = \psi_5(U) = \cup_{u \in U} \psi_5(u)$, $\exists u \in U$ s.t. $v \in \psi_5(u)$. Since $u, u', u'' \in U$, by Lemma 5, v, s' and s'' are pair-wise equivalent in G'. Since $s' \sim_N v$ and $s'' \in NS(s')$, if $v \neq s''$, it must follow that $s'' \in NS(v)$ and $v \in NS(s'')$. Thus, $\forall v \in \psi_8(U)$, if $v \neq s''$, $v \in NS(s'')$. In other words, $NS(s'') \supseteq \psi_8(U) - \{s''\}$. Since $s'' \sim_N v$, we also have $NS(v) = NS(s'') \cup \{s''\} - \{v\}$, thus $NS(v) \supseteq \psi_8(U) - \{v\}$. □

Lemma 7. *In both Algorithms 4.1 and 4.2, if the semantics of ψ and edge correspondence hold in Step 4, then in Step 9, $\forall u \in (\cup_{s \in U} NS(s) - U), \forall u' \in U, \forall v \in \psi_9(u), \forall v' \in \psi_9(u')$, $(v, v') \in E(G')$.*

Proof. According to Table 4.1, the ψ mapping does not change between Steps 4 and 9. Thus the semantics of ψ will hold on vertices in $V(H_4)$ in Steps 4 through 9. Also, since no edge in H_4 is

removed between Steps 4 and 9, the edge correspondence will hold in Step 9 on any edge that does not incident on w.

Consider any $u \in \cup_{s \in U} NS(s) - U$. $\exists u'' \in U$ s.t., $u \in NS(u'')$ (hence $(u, u'') \in E(H_9)$) but $u \notin U$. By the semantics of ψ, $\psi_9(u) \neq \emptyset$ and $\psi_9(u'') \neq \emptyset$. By edge correspondence, $(u, u'') \in E(H_9)$ implies $\forall v \in \psi_9(u), \forall v'' \in \psi_9(u'')$, $(v, v'') \in E(G')$, or equivalently, $\psi_9(u) \subseteq NS(v'')$ and $\psi_9(u'') \subseteq NS(v)$.

Consider any vertex $u' \in U$ in Step 9. Again by the semantics of ψ, $\psi_9(u') \neq \emptyset$ in G'. Consider any vertex $v' \in \psi_9(u')$. By Lemma 5, since $u', u'' \in U$, $\forall v'' \in \psi_5(u'')$, $v' \sim_N v''$. Because $V(H_9) = V(H_5) \cup \{w\}$ and $u'' \neq w$, $\forall v'' \in \psi_9(u'')$, $v' \sim_N v''$. Since $\psi_9(u) \subseteq NS(v'')$ and $v' \sim_N v''$, it follows $\psi_9(u) \subseteq NS(v')$. That is, $\forall v \in \psi_9(u)$, $(v, v') \in E(G')$. □

Lemma 8. *In both Algorithms 4.1 and 4.2, if the semantics of ψ and the edge correspondence hold in Step 4, the edge correspondence will also hold in Step 12.*

Proof. Consider any edge $(u, u') \in E(H_{12})$. Either $(u, u') \in E(H_4)$ or $(u, u') \notin E(H_4)$.

If $(u, u') \in E(H_4)$, none of u and u' can be w (the new vertex added in Step 7) or be removed in Step 12. Thus, $\psi_{12}(u) = \psi_4(u)$ and $\psi_{12}(u') = \psi_4(u')$; and by assumption, $\forall v \in \psi_{12}(u)$ and $\forall v' \in \psi_{12}(u')$, $(v, v') \in E(G')$.

If $(u, u') \notin E(H_4)$, the edge (u, u') must be either (w, w), the edge added in Step 8 or (w, u') for some $u' \in \cup_{s \in U} NS(s) - U$, one of the edges added in Step 9. In both cases, $|U| > 1$ in Step 6.

If $(u, u') = (w, w)$, it follows $\cup_{s \in U} NS(s) \cap U \neq \phi$. By Lemma 6, $\forall v \in \psi_8(U)$, $NS(v) \supseteq \psi_8(U) - \{v\}$. Therefore, $\forall v, v' \in \psi_8(U)$ s.t. $v \neq v'$, $(v, v') \in E(G')$. By Table 4.1, $\psi_8(U) = \psi_4(U) = \psi_{10}(w) = \psi_{12}(w)$ and $\psi_4(U)$ is removed in Step 11. Hence, $\forall v, v' \in \psi_{12}(w)$ s.t. $v \neq v'$, $(v, v') \in E(G')$.

If $(u, u') = (w, u')$, by Lemma 7, $\forall v \in \psi_9(U)$ and $\forall v' \in \psi_9(u')$, $(v, v') \in E(G')$. Similarly, $\psi_9(U) = \psi_4(U) = \psi_{10}(w) = \psi_{12}(w)$ and $\psi_4(U)$ is removed in Step 11. Because $u' \notin U$, $\psi_{12}(u') = \psi_9(u')$. So $\forall v \in \psi_{12}(w), \forall v' \in \psi_{12}(u')$, $(v, v') \in E(G')$.

Consider any edge $(v, v') \in E(G')$. By Lemma 4, we can assume that $v \in \psi_4(u)$ and $v' \in \psi_4(u')$ for some $u, u' \in V(H_4)$. By the edge correspondence and the semantics of ψ, $(u, u') \in$

$E(H_4)$. Let us consider what happens to (u, u') in Step 12.

If $(u, u') \in E(H_{12})$, then neither of u and u' is involved in Step 8, 9, or 11. By Table 4.1, $\psi_{12}(u) = \psi_{12}(u')$. Thus, $v \in \psi_{12}(u)$ and $v' \in \psi_{12}(u')$ and by Lemma 4, $\phi_{12}(v) = u$ and $\phi_{12}(v') = u'$. So $(u, u') = (\phi_{12}(v), \phi_{12}(v')) \in E(H_{12})$.

If $(u, u') \notin E(H_{12})$, at least one of u and u' must be removed in Step 11. According to the algorithms, this happens only if at least one of u and u' is in U in Step 5 and $|U| > 1$. Without loss of generality, assume $u \in U$ and consider u'. Either $u' \in U$ or $u' \notin U$.

If $u' \in U$, then $(u, u') \in E(H_4)$ implies $\cup_{u \in U} NS(u) \cap U \neq \phi$ in Step 8, therefore, $(w, w) \in E(H_8)$. Because (w, w) is not removed in Step 11, it follows $(w, w) \in E(H_{12})$. Because $\phi_{10}(v) = \phi_{10}(v') = w$ and $\phi_{12} = \phi_{10}$, it follows $(\phi_{12}(v), \phi_{12}(v')) \in E(H_{12})$.

If $u' \notin U$, then $u' \in \cup_{u \in U} NS(u) - U$ in Step 9, therefore, $(w, u') \in E(H_9)$. Since (w, u') is not removed in Step 11, it follows $(w, u') \in E(H_{12})$. Because $\psi_4(u) \subseteq \psi_4(U)$, $\phi_{10}(v) = w$, and because $\psi_4(u') \nsubseteq \psi_4(U)$, $\phi_{10}(v') = \phi_4(v') = u'$. Because $\phi_{12} = \phi_{10}$, $(\phi_{12}(v), \phi_{12}(v')) \in E(H_{12})$. □

Lemma 9. *In both Algorithm 4.1 and 4.2, if in Step 4, the semantics of ψ and the edge correspondence hold and the semantics of ϕ holds on a vertex $u \in V(H_4)$, then in Step 12, $u \in V(H_{12})$ and the semantics of ϕ still holds on u.*

Proof. We prove by contradiction.

Assume that in Step 4, the semantics of ψ and the edge correspondence hold and the semantics of ϕ holds on a vertex $u \in V(H_4)$, but in Step 12, $u \notin V(H_{12})$. Then, u must be removed in Step 11. So $u \in U$ and $|U| >= 2$ in Step 6. Thus $\exists u' \in U$, $u' \neq u$. By Lemma 5, $\forall v \in \psi_5(u)$ and $\forall v' \in \psi_5(u')$, $v \sim_N v'$. Because $\psi_5 = \psi_4$ and the semantics of ψ holds in Step 4, $u' \neq u$ implies $\phi_4(v) \neq \phi_4(v')$. Thus, $v \sim_N v'$ but $\phi_4(v) \neq \phi_4(v')$, which is contrary to the assumption that the semantics of ϕ on u holds in Step 4.

Thus, $u \in V(H_4)$ and $u \in V(H_{12})$. Now assume that the semantics of ϕ on u does not hold in Step 12. That is, $\forall v, v' \in V(G')$ s.t. $v \neq v'$ and $\phi_4(v) = u$, $\phi_4(v') = u$ iff $v \sim_N v'$ in G', but in Step 12, $\exists v, v' \in V(G')$ s.t. $v \neq v'$ and $\phi_{12}(v) = u$, either (1) $\phi_{12}(v') = u$ but $v \nsim_N v'$ or (2)

$\phi_{12}(v') \neq u$ but $v \sim_N v'$. Notice that since G' is fixed, we only need to consider changes to the ϕ-mapping.

In case (1), since $v \nsim_N v'$ in G', the change to the ϕ-mapping is from $\phi_4(v') \neq u$ to $\phi_{12}(v') = u$, which can only occur in Step 10. Since $\phi_{10}(v') = w$ and $\phi_{12} = \phi_{10}$, it follows $u = w$, where w is the new vertex added in Step 7. However, since $w \notin V(H_4)$ but $u \in V(H_4)$, it follows $u \neq w$, a contrary.

In case (2), since $v \sim_N v'$ in G', the change to the ϕ-mapping is from $\phi_4(v') = u$ to $\phi_{12}(v') \neq u$, which again can occur only in Step 10. Thus, $\phi_{12}(v') = w$ and $u \in U$. Since $\phi_{12}(v) = u \neq w = \phi_{12}(v')$, it follows $u \notin U$, a contrary. □

Lemma 10. *In both Algorithms 4.1 and 4.2, if in Step 4, the semantics of ψ and the edge correspondence hold and if a new vertex w is added in Step 7, then the semantics of ϕ will hold on w in Step 12.*

Proof. We need to show $\forall v, v' \in V(G')$ s.t. $v \neq v'$ and $\phi_{12}(v) = w$, (1) if $\phi_{12}(v') = w$, $v \sim_N v'$ in G' and (b) if $v \sim_N v'$ in G', $\phi_{12}(v') = w$. Notice that since w is added to H in Step 7, U must contain at least two vertices.

(a) If $\phi_{12}(v') = w$, then $\phi_{10}(v) = \phi_{10}(v') = w$ by Table 4.1. So $v, v' \in \psi_{10}(w) = \psi_4(U)$. Because $\psi_4 = \psi_5$ and the semantics of ψ holds in Step 4, $\exists u, u' \in U$ in Step 5, s.t. $v \in \psi_5(u)$ and $v' \in \psi_5(u')$. By Lemma 5, $v \sim_N v'$ in G'.

(b) If $v \sim_N v'$ in G', where $v \neq v'$ and $\phi_{12}(v) = w$, then $v \in \psi_{12}(w)$ by Lemma 4. By Table 4.1, $\psi_{12}(w) = \psi_{10}(w) = \psi_4(U)$ and $\phi_5 = \phi_4$. So $\exists u \in U$, $v \in \psi_4(u)$. And $\phi_4(v) = u = \phi_5(v) \in U$ by the assumption. Because $v \sim_N v'$, $\phi_5(v') \in U$ by Lemma 5. Thus, $\phi_4(v') \in U$. Therefore, $v' \in \psi_4(U) = \psi_{12}(w)$. By Lemma 4, $\phi_{12}(v') = w$. □

Lemma 11. *In both Algorithms 4.1 and 4.2, if the label consistency holds in Step 4 for vertices in H_4, it will still hold in Step 12 for vertices in H_{12}.*

Proof. Consider any vertex in $u \in H_{12}$. Either $u \in V(H_4)$ or $u \notin V(H_4)$. If $u \in V(H_4)$, by assumption, $l_u = |\psi_4(u)|$. But, $\psi_4(u) = \psi_{12}(u)$, otherwise, it must be changed in Step 11, where

u must be removed from H_4. Thus, $l_u = |\psi_{12}(u)|$. If $u \notin V(H_4)$, we must have $u = w$ and $l_u = l_w = \sum_{u \in U} l_u = \sum_{u \in U} \psi_4(u)$. By the semantics of ψ, $\psi_4(u) \cap \psi_4(u') = \emptyset$ for any $u \neq u'$. So $\psi_4(U) = \cup_{u \in U} \psi_4(u)$. $|\psi_4(U)| = \sum_{u \in U} |\psi_4(u)|$. Since $\psi_{10}(w) = \psi_{10}(U) = \psi_4(U)$ by Table 4.1, $|\psi_{10}(w)| = \sum_{u \in U} l_u$. Since $\psi_{12}(w) = \psi_{10}(w)$, $l_w = |\psi_{12}(w)|$. □

Lemma 12. *In both Algorithms 4.1 and 4.2, if in Step 4, the semantics of ψ holds and the label consistency holds for all edges in $E(H_4)$, then in Step 12, the label consistency will also hold for all edges in $E(H_{12})$.*

Proof. Consider any edge $(u, u') \in E(H_{12})$. Either $(u, u') \in E(H_4)$ or $(u, u') \notin E(H_4)$.

If $(u, u') \in E(H_4)$, by assumption, $l_{(u,u')} = |S_4(u, u')|$. Since $(u, u') \in E(H_4)$ and $(u, u') \in E(H_{12})$, none of u and u' is w (because w is not in H_4) or is in U in Step 11 (otherwise it must be removed already). By Table 4.1, $\psi_{12}(u) = \psi_4(u)$ and $\psi_{12}(u') = \psi_4(u')$. So $S_{12}(u, u') = S_4(u, u')$. Thus, $l_{(u,u')} = |S_{12}(u, u')|$.

If $(u, u') \notin E(H_4)$, it must be a new edge added to H_4 in Step 8 or Step 9.

If $(u, u') = (w, w)$, it is labeled by the value $\sum_{u,u' \in U \wedge (u,u') \in E(H)} l_{(u,u')} = \sum_{u,u' \in U} |S_4(u, u')|$ because $S_4 = S_8$ according to Table 4.1. Let (v, v') be an s-edge in $S_{12}(w, w)$. By Table 4.1, $\psi_{12}(w) = \psi_{10}(w) = \psi_4(U)$. Thus, $S_{12}(w, w) = S_4(U, U)$. By the assumption and the semantics of ψ, for any two distinct edges $(u, u') \neq (s, s')$ in H_4, $|S_4(u, u')| + |S_4(s, s')| = |S_4(u, u') \cup S_4(s, s')|$. So there must be a unique $(u, u') \in H_4$ s.t. $(v, v') \in S_4(u, u')$. Thus $S_4(U, U) = \cup_{u,u' \in U} S_4(u, u')$, and $|S_{12}(w, w)| = \sum_{u,u' \in U} |S_4(u, u')|$.

If $(u, u') = (w, u')$, it is labeled by the value $\sum_{u \in U} |S_4(u, u')|$ where $u' \in \cup_{s \in U} NS(s) - U$. Similarly, if the new edge is (w, u') for some $u' \in \cup_{u \in U} NS(u) - U$, each s-edge $(v, v') \in S_{12}(w, u')$ will be in $S_4(u, u')$ for some $u \in U$. This implies $S_{12}(w, u') = S_4(U, u') = \cup_{u \in U} S_4(u, u')$. Again, due to disjoint sets of s-edges $|S_{12}(w, u')| = |\cup_{u \in U} S_4(u, u')| = \sum_{u \in U} |S_4(u, u')|$. □

Theorem 3. *Algorithms 4.1 and 4.2 are correct.*

Proof. We need to show that all four properties of Definition 12 hold in Step 13. In Step 1, (H, ϕ, ψ) is the N-map of graph G, therefore the properties of Definition 7 are satisfied.

By Lemma 3, the semantics of ψ, the edge correspondence, and the label consistency hold in Step 3 of both algorithms, and therefore in Step 4 of iteration 1. By Lemma 4, 8, 11 and 12, these three properties hold in Step 12 of iteration 1 and in Step 4 of iteration 2. Likewise, these three properties hold in Steps 4 and 12 of each iteration. Since the set T must be empty eventually, the while loop will always terminate. So the three properties hold in Step 13. We only need to prove that the semantics of ϕ holds as the algorithms terminate.

Assume the algorithm executes n iterations. Let U_k and w_k denote the set U and the vertex w in iteration k of the loop. Consider any $u \in V(H_{13})$. Vertex u is w_k for some k ($1 \leq k \leq n$) or $u \in V(H)$.

If $u = w_k$, by Lemma 10, the semantics of ϕ holds on u in Step 12 of iteration k, and in Step 4 of iteration $k+1$. By Lemma 9, the semantics of ϕ holds on u in the subsequent iterations. So it also holds in Step 13.

If $u \in V(H)$, then by Table 4.1, $u \in V(H_3)$. The semantics of ϕ may or may not hold on u in Step 3.

If the semantics of ϕ holds on u in Step 3, it must also hold in Step 4 of iteration 1. By Lemma 9, it holds on u in Step 4 and 12 in each subsequent iterations. So it also holds in Step 13.

If the semantics of ϕ does not hold on u in Step 3, we prove that $u \in U_k$ and $|U_k| > 1$ for some iteration k. Therefore, it will be removed from the N-map in Step 11 of iteration k. Since vertices removed from the N-map in an iteration are never added back, $u \notin V(H_{13})$. As a result, for any vertex that is in both $V(H)$ and $V(H_{13})$, the semantics of ϕ must hold on u.

Since the semantics of ϕ does not hold on u in Step 3, therefore, $\exists v, v' \in V(G')$, s.t. $v \neq v'$ and $\phi_3(v) = u$, either (1) if $\phi_3(v') = u$, $v \nsim v'$ in G' or (2) if $v \sim_N v'$ in G', $\phi_3(v') \neq u$. By Table 4.1, $\phi_3 = \phi$. If $\phi_3(v') = u$, then $\phi(v) = \phi(v') = u$. Because (H, ϕ, ψ) is an N-map of G, $v \sim_N v'$ in G. Since $\psi_3 = \psi$, vertices $v, v' \in \psi_3(u)$. By Lemma 3, the edge correspondence holds in Step 3. So $\forall v \in \psi_3(u)$, $NS(v) = \psi_3(NS(u)) - \{v\}$ in G'. Hence, $v \sim_N v'$ in G'. Therefore, if the semantics of ϕ does not hold on u, only (2) is possible. By Lemma 3 and the semantics of ψ, we can assume $\phi_3(v') = u' \neq u$ and $v' \in \psi_3(u')$ for some $u' \in V(H_3)$. By Table 4.1, if $\phi_3(v) \neq \phi_3(v')$, then

$\phi(v) \neq \phi(v')$ implies $v \nsim_N v'$ in G. However, $v \sim_N v'$ in G'. Hence, at least $NS(v)$ or $NS(v')$ is changed by $mbu(M)$ in Algorithm 4.1 (or $mbi(M)$ in Algorithm 4.2).

Suppose $NS(v)$ is changed by $mbu(M)$ (or $mbi(M)$ in case of Algorithm 4.2). Thus, $v \in (W - \psi(\cap_{u \in M} NS(u))) \cup \psi(M)$. By the semantics of ϕ, $u \in T$ in Step 3 of the algorithm. Then, vertex u must be in U in Step 5 in some iteration k. Let $u \in U_k$. We must have $\phi_4(v) = \phi_3(v) = u$. Otherwise, u must be removed before iteration k. Since $v \sim_N v'$, by Lemma 5, $\phi_4(v') \in U_k$. Because $\phi_3(v') = u'$, $\phi_4(v') = u'$. Otherwise, $\phi_4(v') = w$, a new vertices added in Step 7 in iteration $l < k$. By Lemma 10, the semantics of ϕ holds on w, contradict with $w \neq u$ but $v \sim_N v'$. Thus, $u, u' \in |U_k|$. So $|U_k| \geq 2$. U_k will be removed in Step 11. Thus, $u \notin V(H_{13})$. □

4.2.4 Utility Loss of an VEC-Merge

Algorithms 4.1 and 4.2 use heuristics to minimize the number of changed added or removed. In this section, we characterize the relationship between the number of changed edges and a distance measure of vertex degree distributions.

Let G' be the new underlying graph obtained by an VEC-merge on the N-map of a graph G. The difference between the degree distributions of G and G' can be measured by some distance metric. Such a distance can be regarded as a utility loss with respect to degree distribution. One well-known distance measure between two distributions is the Earth Move Distance (EMD), which was originally defined in the context of a linear programming problem, called the transportation problem [48].

Consider histogram $H = \{(b_1, p_1), ..., (b_n, p_n)\}$, where b_i is a domain value (such as a vertex degree) and p_i is the frequency (or a count) of b_i. In the transportation problem, b_i is a bin and p_i is the weight of earth in the bin. The change from H to another histogram $H' = \{(c_1, q_1), ..., (c_m, q_m)\}$, where c_j is a bin and q_j is the weight of earth in c_j, is viewed as the result of moving earth from bins of H into bins of H'. Let the flow of the earth from bin b_i to bin c_j be

f_{ij} with a cost of d_{ij}. The problem is to determine the flow that minimizes

$$cost = \sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij} \cdot d_{ij} \tag{4.2}$$

subject to following constraints:

$$\begin{aligned}
&(i) && f_{ij} \geq 0, \forall i \in [1,n], j \in [1,m] \\
&(ii) && \sum_{j=1}^{m} f_{ij} \leq p_i, \forall i \in [1,n] \\
&(iii) && \sum_{i=1}^{n} f_{ij} \leq q_j, \forall j \in [1,m] \\
&(iv) && \sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij} = \min\{\sum_{i=1}^{n} p_i, \sum_{j=1}^{m} q_j\}
\end{aligned}$$

The EMD between the two distributions is the average cost per unit of the optimal flow, i.e., $emd(H,H') = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} \hat{f}_{ij} \cdot d_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{m} \hat{f}_{ij}}$, where $\hat{f}_{ij}$ is the optimal flow from b_i to c_j. If H and H' are the degree distributions of G and G', respectively, we can use $emd(H,H')$ to measure the utility loss of the degree distribution.

Although there are several distance metrics other than EMD that has less computation complexity, such as Euclidean L_2, Cosine, and Kullback-Leibler, [10], EMD is especially interesting because as indicated by the following theorem, it is proportional to the number of edges changed by the two types of VEC-merges defined in 4.2.1, and therefore can be easily computed. This makes it efficient to choose VEC-merges during anonymization to minimize utility loss specially regarding the degree distribution.

Theorem 4. *Let G and G' be two graphs such that $V(G) = V(G')$ and $E(G) \subseteq E(G')$. Let the degree distributions of G and G' be respectively $H = \{(b_1,p_1),...,(b_n,p_n)\}$ and $H' = \{(c_1,q_1),...,(c_m,q_m)\}$, where b_i (resp. c_j) is a vertex degree and p_i (resp. q_j) is the count of vertices with degree b_i (resp. c_j). If in a graph anonymization, the cost to increase or decrease the degree of any vertex by one*

is a constant w, the EMD between H and H' is

$$emd(H, H') = \frac{2w}{|V(G)|} \cdot (|E(G') - E(G)|). \tag{4.3}$$

Proof. Let $H = \{(b_1, p_1), ..., (b_n, p_n)\}$ and $H' = \{(c_1, q_1), ..., (c_m, q_m)\}$ be the vertex degree histograms of G and G', respectively. Then, $\sum_{i=1}^{n} p_i = \sum_{j=1}^{m} q_j = |V(G)|$. Thus, the constraints (ii)-(iv) of Eq. (4.2) become (ii') $\sum_{j=1}^{m} f_{ij} = p_i$, (iii') $\sum_{i=1}^{n} f_{ij} = q_j$, and (iv') $\sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} = |V(G)|$.

Since the cost of increasing the degree of any vertex by 1 is $w > 0$, the cost of changing the degree of a vertex from b_i to c_j will be $d_{ij} = w \cdot |b_i - c_j|$. We now show that $f_{ij} = n_{ij}$ is the optimal solution to Eq. (4.2) given (ii') - (iv') and d_{ij}, where n_{ij} is the number of vertices whose degrees change from b_i to c_j, if $c_j \geq b_i$; and 0, if $c_j < b_i$.

First, it is easy to verify that $f_{ij} = n_{ij}$ satisfies all constraints. Obviously, $n_{ij} \geq 0$ for each n_{ij}. Since the VEC-merge may only add edges to vertices, the degree of each vertex is either increased or the same in G', thus, $p_i = \sum_{j=1}^{m} n_{ij}$, $q_j = \sum_{i=1}^{n} n_{ij}$, and $\sum_{i=1}^{n} \sum_{j=1}^{m} n_{ij} = |V(G)|$.

We now show that $f_{ij} = n_{ij}$ is the optimal solution of Eq. (4.2).

Notice that for any given set of flow f_{ij}, $1 \leq i \leq n$ and $1 \leq j \leq m$, we have $\sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} \cdot w \cdot |b_i - c_j| \geq |\sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} \cdot w \cdot (b_i - c_j)|$. By constraints (ii') and (iii'), $|\sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} \cdot w \cdot$

$(b_i - c_j)| \geq |\sum_{i=1}^{n} p_i \cdot w \cdot (b_i - c_1) + \sum_{j=2}^{m} q_j \cdot w \cdot (c_1 - c_j)|$. However,

$$
\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=1}^{m} n_{ij} \cdot w \cdot |b_i - c_j| \\
&= w \cdot \sum_{i=1}^{n} \sum_{j\in[1,m] \text{ s.t.} b_i \leq c_j} n_{ij} \cdot |b_i - c_j| \\
&\quad + w \cdot \sum_{i=1}^{n} \sum_{j\in[1,m] \text{ s.t.} b_i > c_j} n_{ij} \cdot |b_i - c_j| \\
&= w \cdot |\sum_{i=1}^{n} \sum_{j\in[1,m] \text{ s.t.} b_i \leq c_j} n_{ij} \cdot (b_i - c_j)| \\
&\quad + w \cdot |\sum_{i=1}^{n} \sum_{j\in[1,m] \text{ s.t.} b_i > c_j} n_{ij} \cdot (b_i - c_j)| \\
&= w \cdot |\sum_{i=1}^{n} \sum_{j\in[1,m] \text{ s.t.} b_i \leq c_j} n_{ij} \cdot (b_i - c_j) \\
&\quad + \sum_{i=1}^{n} \sum_{j\in[1,m] \text{ s.t.} b_i > c_j} n_{ij} \cdot (b_i - c_j)| \\
&= w \cdot |\sum_{i=1}^{n}\sum_{j=1}^{m} n_{ij} \cdot (b_i - c_j)| \\
&= w \cdot |\sum_{i=1}^{n} p_i \cdot (b_i - c_1) + \sum_{j=1}^{m} q_j \cdot (c_1 - c_j)|
\end{aligned}
$$

Thus, $\sum_{i=1}^{n}\sum_{j=1}^{m} n_{ij} \cdot w \cdot |b_i - c_j|$ is the minimum of $\sum_{i=1}^{n}\sum_{j=1}^{m} f_{ij} \cdot w \cdot |(b_i - c_j)|$. Since $\sum_{i=1}^{n}\sum_{j=1}^{m} n_{ij} = V(G)$, $EMD = \frac{w \cdot \sum_{i=1}^{n}\sum_{j=1}^{m} n_{ij} \cdot |b_i - c_j|}{|V(G)|}$. Because $\sum_{i=1}^{n}\sum_{j=1}^{m} n_{ij} \cdot |b_i - c_j|$ is the total number of vertices degree change, it equals to $2 \times |E(G') - E(G)|$. □

Since both merge-by-union and merge-by-intersection have a cost 1 for adding (resp. removing) an edge to (resp. from) any vertex, by Theorem 4, the EMD of these two VEC-merges is $emd(H, H') = \frac{2}{|V(G)|} \cdot (|E(G') - E(G)|)$. We leave it for the future work to study the relationship between the number of changed edges and other distance metrics of degree distributions.

4.3 Merge Plan Construction and Execution

Our edge anonymity algorithm, Algorithm 4.3, is based on the ideas of constructing and executing one or more static plan of a sequence of VEC-merges and postponing the checking of edge confidentiality until a plan is executed completely.

4.3.1 The Main Algorithm

One goal of the algorithm is to eliminate unsatisfied edges in the N-map. Recall that an unsatisfied edge represents an EEC in the underlying graph whose edge disclosure probability does not satisfy a given edge confidentiality requirement. By eliminating all unsatisfied edges, the resulting underlying graph will satisfy the given edge confidentiality. The second goal is to minimize the number of changed edges, thus to minimize utility loss relative to degree distribution. To determine statically a sequence of VEC-merges that achieves both goals for large graphs is very difficult due to three reasons. First, there are many possible merge sequences to explore. Second, we can check if all unsatisfied edges are eliminated only after an entire sequence is executed. Third, it is difficult to estimate the number of edges to be changed by a given sequence. To overcome these difficulties, our algorithm only considers merge sequences that *covers* rather than *eliminates* all unsatisfied edges.

Definition 13. (**Merge Plan**) Given the N-map (H, ϕ, ψ) of a graph, a merge plan is a sequence of sets $\mathcal{M} = (M_0, ..., M_{n-1})$, where each M_i, for $0 \leq i \leq n-1$, is a set of two or more vertices in the N-map; all sets are pair-wise disjoint, i.e., for any $0 \leq i, j \leq n-1$ and $i \neq j$, $M_i \cap M_j = \phi$; and every unsatisfied edge is covered by some set in the plan.

Intuitively, a merge plan represents a sequence of VEC-merges $(vm(M_0), vm(M_1), ..., vm(M_{n-1}))$, where $vm(M_i)$ is either a merge-by-union or a merge-by-intersection to be performed on the N-map produced by the previous VEC-merge. It is worth noting that each merge-set M_i contains only vertices of the initial N-map. This allows the plan to be created statically without the need of finding the intermediate N-maps.

Algorithm 4.3 Merge Plan Construction and eXecution (MPCoX)

Input: a graph G, edge confidentiality threshold τ,
and plan execution type t
Output: a τ-confidential graph or an empty graph

1. m = the N-map of graph G;
2. if $1 - \frac{\#\ of\ sensitive\ edges\ in\ G}{\frac{1}{2}|V(G)|\cdot(|V(G)|-1)} < \tau$ then
   return an empty graph;
3. while (m is not τ-confidential) do
4. $\mathcal{M}$= MergePlanConstruction(m, τ);
5. m = MergePlanExecution($\mathcal{M}$, t, m);
6. return the underlying graph of m;

In Step 1 of Algorithm 4.3, the N-map of the original graph is computed. Step 2 calculates the edge confidentiality assuming edges are added to the original graph to make it a complete graph. This estimates the possibility of fulfilling the given edge confidentiality requirement. If making the graph a complete graph still cannot satisfy the edge confidentiality requirement, the algorithm terminates without any further operation. In Step 3, the edge confidentiality is measured on the N-map. If the privacy requirement is not satisfied, Step 4 will build a merge plan using Algorithm 4.4. The merge plan is executed in Step 5 using Algorithm 4.5 with a user-specified type of execution (see Section 4.3.3 for details). After the execution of a merge plan, if the edge confidentiality is still not achieved, Steps 4-5 will be repeated again. Finally, in Step 6, the underlying graph of the resulted N-map is constructed and returned as the anonymized graph.

Discussion. Algorithm 4.3 can be extended to prevent vertex re-identification in two ways. One option is to extend the merge plan construction so that for each merge set M_i, $|\psi(M_i)|$ is no less than a threshold, say k or $\lceil\frac{1}{1-\tau}\rceil$. Another option is to modify the final N-map by merging VECs of which the sizes are smaller than the threshold. Both options can be implemented efficiently.

4.3.2 Merge Plan Construction

Given the N-map of a large graph, there may be a large number of possible merge plans. Algorithm 4.4 constructs a merge plan by a greedy choice of merge-sets. On the one hand, it guarantees to cover all unsatisfied edges in the N-map, and on the other hand, it heuristically minimizes utility

loss measured by EMD of degree distributions.

Algorithm 4.4 Merge Plan Construction

Input: the N-map (H, ϕ, ψ) of graph G,
the edge confidentiality threshold τ
Output: a merge plan $\mathcal{M}$

1. $\mathcal{M}$ = ();
2. $S = \{v | v \in V(H) \wedge ud(v) > 0\}$;
3. while ($|S| \geq 1$) do
4. if ($|S| \geq 2$) then
5. $M = \{v_1, v_2\}$ s.t. $rud(v_1)$ and $rud(v_2)$ are highest in S;
6. else if there exists $v \in V(H)$ s.t. $v \notin S \cup \mathcal{M}$, then
7. $M = S \cup \{v\}$;
8. else break the loop;
9. $S = S - M$;
10. $\mathcal{M} = concat(\mathcal{M}, M)$;
11. if ($M_{n-1} \neq \emptyset$ is the last set in $\mathcal{M}$) then $M_{n-1} = M_{n-1} \cup S$;
12. return $\mathcal{M}$;

Step 1 of Algorithm 4.4 initializes the merge plan. Step 2 finds the set of vertices of the N-map that has at least one unsatisfied degree. Notice that for each unsatisfied edge, both end vertices are included in this set. In Steps 3-10, merge-sets are constructed one at a time. Each merge-set, except the last one, is constructed heuristically by including two vertices that have the maximum ratio of unsatisfied degree. Although there are other heuristics, such as choosing vertices with the minimum ratio or choosing one vertex with the maximum ratio and the other with the minimum ratio, none of the alternatives always outperforms the others. Due to the small word property of the social graphs, many vertices have low degrees, most low ratios of unsatisfied degree occur on high-degree vertices. Thus selecting the minimum ratios will often merge high-degree vertices at early stage of the process, resulting slow improvement of the privacy and quick loss of utility. On the other hand, low-degree vertices are more likely to have high ratio of unsatisfied degree. Merging low-degree vertices often can reduce unsatisfied edges with less number of edges added or removed. Therefore, we choose vertices by the maximum ratio. The last merge-set may contain up to three vertices. The merge plan is completed after every unsatisfied edge is

covered. The algorithm may return an empty merge plan if the input N-map already satisfies the edge confidentiality requirement.

The complexity of Algorithm 4.4 is $O(|V(H)| \log |V(H)|)$, where H is the input N-map. This is because one can keep vertices in T in descending order of their unsatisfied degrees. This will take $O(|V(H)| \log |V(H)|)$. Then the loop in Steps 3-10 will take $O(|V(H)|)$.

4.3.3 Merge Plan Execution

Algorithm 4.5 executes a merge plan according to a given type of execution. We consider three strategies for executing a merge plan. For each merge-set, Strategy U performs only the merge-by-union, Strategy I performs only the merge-by-intersection, and Strategy H performs either a merge-by-union or a merge-by-intersection, depending on which of the two will change fewer edges (according to an estimation). If both types of VEC-merges change the same number of edges, an additional strategy is used to break the tie. Specifically, H-a breaks the tie by performing the merge-by-union, H-d by merge-by-intersection, and H-r by a random choice between the two types of VEC-merges. Consequently, we have five execution types: U, I, H-a, H-d, and H-r.

Algorithm 4.5 Merge Plan Execution

Input: a merge plan $\mathcal{M}$ on an N-map (H, ϕ, ψ) and
an execution type t
Output: a new N-map
1. for each merge-set M_i in $\mathcal{M}$ do
2. if $M_i \subseteq V(H)$ and covers some unsatisfied edge, then
3. if (t is type U) then
4. current N-map = $mbu(M_i)$;
5. else if (t is type I) then
6. current N-map = $mbi(M_i)$;
7. else /* t is H-a, H-d, or H-r */
8. n_u = number of edges to be added by $mbu(M_i)$;
9. n_i = number of edges to be removed by $mbi(M_i)$;
10. if ($n_i < n_u$) then current N-map = $mbi(M_i)$;
11. else if ($n_i > n_u$) then current N-map = $mbu(M_u)$;
12. else current N-map = $mbi(M_i)$ or $mbu(M_u)$
according to t;
13. return current N-map;

In Steps 1-12 of Algorithm 4.5, merge-sets are considered one by one. For each merge-set, Step 2 verifies that all the vertices in the merge-set are still in the current N-map and they still cover at least one unsatisfied edge. If not, this merge-set can be abandoned. Otherwise, an appropriate merge operation is performed in Steps 3-12. For type H-x (where x = a,d,or r) execution, the number of edges to be added/removed by each of the two types of merge operations is estimated in Steps 8 and 9. The justification of using these edge counts as a measure of utility loss is given in Section 4.2.4. The algorithm returns a new N-map in Step 13.

The complexity of Algorithm 4.3 is $O(h \cdot |V(G)|^2 + c \cdot L \cdot K \cdot (|V(H)| \cdot |E(H)| + |V(G)|))$, where h is the size of the largest neighbor set, c is the number of times the loop in Algorithm 4.3 is executed, L is the greatest number of unsatisfied edges among all N-maps derived by Algorithm 4.3, and K is the size of largest T in Step 3 of Algorithms 4.1 and 4.2 executed in Algorithm 4.3. Specifically, the construction of the N-map takes $O(h \cdot |V(G)|^2)$, due to the calculation of the N-partition. The checking of edge confidentiality takes $O(1)$ with unsatisfied edges sorted in advance according to edge confidentiality. The construction of a merge plan (Algorithm 4.4) takes $O(|V(H)| \log |V(H)|)$, and the execution of a merge plan (Algorithm 4.5) takes $O(L \cdot K \cdot (|V(H)||E(H)| + |V(G)|))$.

4.4 Experimental Results

We performed extensive experiments to study the performance of Algorithm 4.3. Due to the lack of common basis for comparison (in terms of the privacy measures and the types of social graphs), we were unable to compare our method with existing edge anonymity methods. Therefore, we focused on evaluation of the properties and implementation alternatives of our algorithm. In particular, we empirically validate the following features of our algorithm.

1. Use VEC-merge as a basic operation.

2. Work on the N-map rather than on the underlying graph.

3. Construct a merge plan according to heuristics.

4. Execute a merge plan with one of the five execution types.

5. Check the edge confidentiality after the execution of an entire merge plan.

We implemented the MPCoX algorithm (i.e., Algorithm 4.3) and compared the five execution types. In this section, the executions of the algorithm with execution types U, I, H-a, H-d and H-r are denoted as MPCoX-U, MPCoX-I, MPCoX-H-a, MPCoX-H-d, and MPCoX-H-r respectively.

We used five data sets, EPINION, FB, ER, SW-1, and SW-2, in our experiments. The EPINION date set was crawled from the Epinion.com in 2009. The graph contains 89,672 vertices and 462,923 edges with vertices representing members and edges representing trust relationship between two member. For the purpose of our study, we treated the directed trust relationships as undirected edges. The FB data set was crawled from the Facebook.com in 2010. The graph has 44,745 vertices and 506,250 edges with vertices representing members and edges representing friendships among members. The ER data set is a Erdos-Renyi random graph [21] with 5,000 vertices and 50,000 edges. The SW-1 data set and SW-2 data set both are small world graphs generated using the Watts and Strogatz model [56]. The two data sets each has 5,000 vertices and an edge rewiring probability of 0.5. The SW-1 set has 25,000 edges and the SW-2 has 50,000 edges. The average vertex degree of SW-1 is similar to FB, and that of SW-2 is similar to EPINION. The experiments were performed on a PC with 2.13GHz Pentium Core 2 processor and 2GB memory.

4.4.1 Using VEC-Merge as Basic Operation

In this experiment, we investigate the effectiveness of performing VEC-merge as basic operation of anonymization. Although we defined VEC-merges on N-maps, the effect of an VEC-merge is to add/remove a set of edges on the underlying graph. In fact, it is straight forward to map a merge-by-union to a set of edge addition and a merge-by-intersection to a set of edge removals on the underlying graph. In this experiment, we compare the run time of using VEC-merge as the basic operation vs using single-edge addition/removal as the basic operation.

To work on the underlying graph (instead of N-map), we revised the GADED-Rand algorithm

to measure the edge confidentiality based on an N-partition. The modified algorithm works directly on the underlying graph and performs edge deletion one edge at a time. After each edge deletion, it updates the graph partition and recomputes the edge confidentiality.

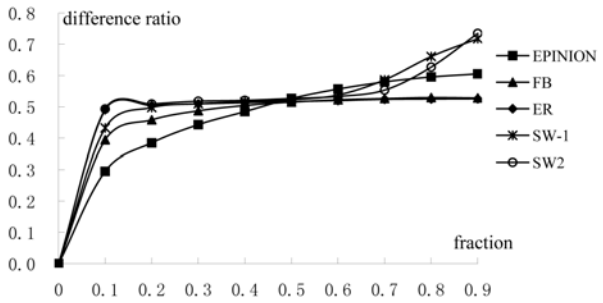
We also implemented an algorithm MonG (Merge on Graph), which performs merge-by-intersection directly in the underlying graph and checks the edge confidentiality immediately after each merge operation. Notice that this algorithm does not involve a merge plan, but it chooses the next merge-set using the same heuristics used in Algorithm 4.4.

We run GADED-Rand and MonG on all five data sets with various test settings, including various edge confidentiality requirements and different fractions of edges being designated as sensitive. Our results show that MonG runs much faster than GADED-Rand. For example, for edge confidentiality threshold $\tau = 0.7$ and with fraction of sensitive edges ranging from 0.1 to 0.9 in increment of 0.1, MonG runs on an average of 1000 times faster than GADED-Rand on EPINION and FB data sets. The results clearly show that using VEC-merge as the basic operation is superior to using single-edge addition/deletion as the basic operation, even without N-map and merge plans.

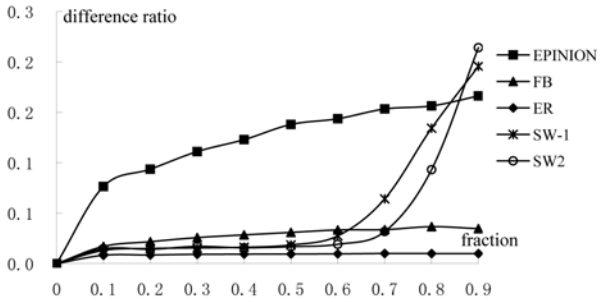
4.4.2 Benefit of N-Map

In this experiment, we investigate the performance gain of executing merge plan on the N-map vs executing the same merge plan directly on the underlying graph.

For the comparison, we implemented a version of MPCoX, called MPCoX-g, which is identical to MPCoX except it performs VEC-merges directly on the underlying graph. Therefore, it stores the underlying graph in the memory and updates the graph as an merge plan is executed.



(a) The Ratio of Number of Vertices Difference to The N-map of Original Graph



(b) The Ratio of Number of Edges Difference to The N-map of Original Graph

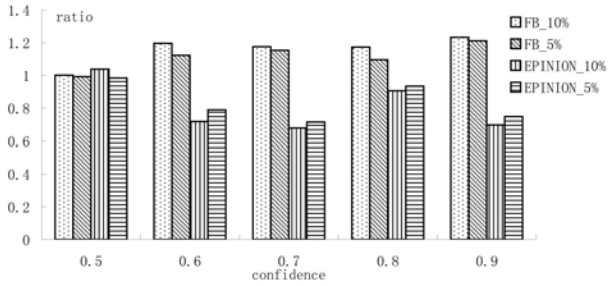
Figure 4.2: Size Comparison between N-maps Produced by MPCoX-U and N-map of Original Graph. In these figures, the X-axis is the fraction of sensitive edges in the original graph. The Y-axis is a ratio value $\frac{|V(H)|-|V(H')|}{|V(H)|}$ or $\frac{|E(H)|-|E(H')|}{|E(H)|}$. Here, the threshold of edge confidentiality is $\tau = 0.7$.

We run both MPCoX-g and MPCoX-U (i.e., MPCoX with execution type U) using all five graphs. Figure 4.2 shows the result under various test settings similar to that of Section 4.4.1. Based on the results, MPCoX-U successfully completed all test cases and always returns an N-map that is smaller than the initial N-map. In fact, during the execution of the MPCoX-U, the number of vertices and edges of the N-map decreases most of the time and never increases. On the other hand, although the number of vertices in the graph maintained by MPCoX-g remains the same, the number of edges in the graph increased significantly and rapidly, causing the graph to grow out of the available memory. In fact, MPCoX-g crashed with the out-of-memory error in many test cases on EPINION and FB data sets. These results indicate that working with the N-map can effectively improve the ability for the algorithm to handle large graphs with limited memory resource.

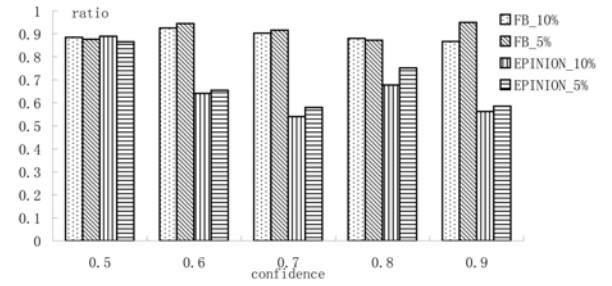
We also like to mention that the difference between MPCoX-g and MPCoX was much less significant for execution types I and H-d, because with these execution types, both N-map and underlying graph were reduced in size due to the removing of edges.

4.4.3 Effect of Our Merge Plan Construction Method

In this experiment, we study the quality of merge plans constructed by Algorithm 4.4, which is based on several heuristics. For comparison, we implemented a new algorithm, called REA, which is identical to MPCoX except that it constructs the merge plan by randomly selecting two vertices from the set S to form each merge-set. In other words, it does not take into consideration the ratio of unsatisfied degrees.



(a) The Execution Time. The Y-axis is the ratio of the execution times of MPCoX-U and REA.



(b) The Earth Move Distance. The Y-axis is the ratio of the numbers of edges added by MPCoX-U and by REA.

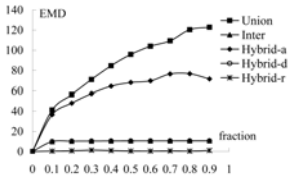
Figure 4.3: A Comparison of MPCoX-U and REA-Union on EPINION and FB Data Sets. The X-axis in these figures is the threshold of edge confidentiality τ. The fraction of sensitive edges was 0.05 and 0.1. The results of REA-Union is the average of running the algorithm 30 times.

We run both algorithms with execution type U on all data sets under various test settings of the edge confidentiality threshold and fraction of sensitive edges. Figure 4.3 shows some typical

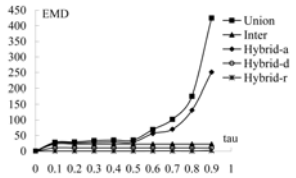
results from our experiments. As shown by the results, MPCoX outperforms REA in terms of the number of edges added, which indicates a better preservation of degree distribution. REA runs faster than MPCoX on FB data set, but slower on EPINION data set. Hence, the time performance of the two algorithms seems to be data-dependent. Similar results were also obtained on the other execution types and on other three data sets.

4.4.4 Comparison of Merge Plan Execution Strategies

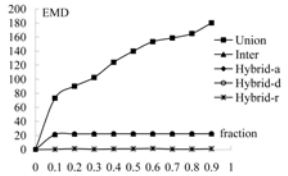
In this experiment, we study the effect of the five merge plan execution strategies on the utility. In addition to the EMD of degree distributions, we also consider clustering coefficient of a graph, and average length of shortest paths. The clustering coefficient of a graph G is $cc(G) = \frac{3N_{\Delta}(G)}{N_3(G)}$, where $N_{\Delta}(G)$ is the number of triangles and $N_3(G)$ is the number of triples in the graph. The average length of shortest paths is $alsp(G) = \frac{\sum_{(u,v)\in cp(G)} lsp(u,v)}{|cp(G)|}$, where $cp(G)$ is the set of pairs of connected vertices in graph G, and $lsp(u, v)$ is the length of the shortest path measured by the number of edges between a connected pair of vertices u and v.



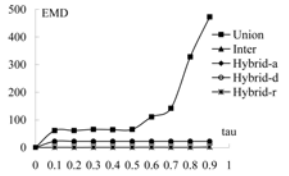
(a) EPINION with $\tau = 0.7$



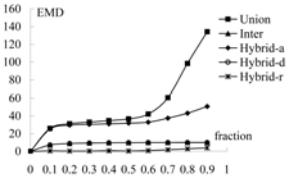
(b) EPINION with fraction being 0.5



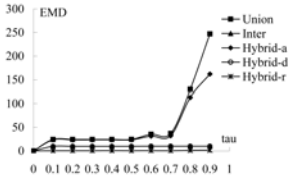
(c) FB with $\tau = 0.7$



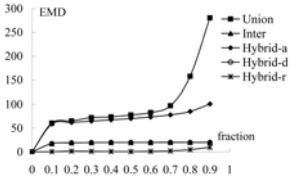
(d) FB with fraction being 0.5



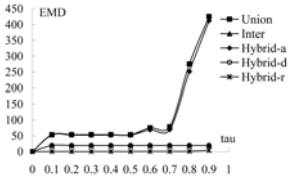
(e) SW-1 with $\tau = 0.7$



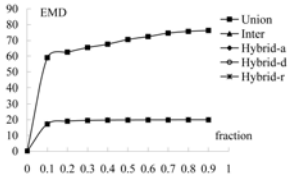
(f) SW-1 with fraction being 0.5



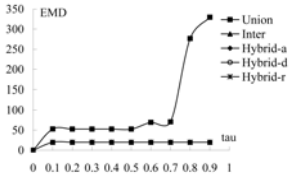
(g) SW-2 with $\tau = 0.7$



(h) SW-2 with fraction being 0.5



(i) ER with $\tau = 0.7$



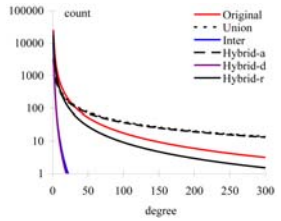
(j) ER with fraction being 0.5

Figure 4.4: Earth Move Distance. The y-axis of all subfigures is the EMD between the degree distributions of the original and the anonymized graphs. The x-axis of subfigure (a), (c), (e), (g) and (i) is the fraction of sensitive edges. The x-axis of subfigure (b), (d), (f), (h) and (i) is the edge confidentiality threshold. A greater EMD indicates worse utility.

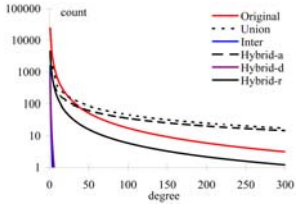
The intention of this experiment is to provide a more coherent view on how these strategies affect the quality of the anonymized graphs. In the experiment, the MPCoX with the five execution

types was executed on all five data sets under various test settings of the edge confidentiality threshold and the fraction of sensitive edges.

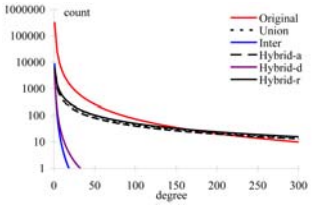
Figure 4.4 shows some results on EMD. As indicated by the results, MPCoX-U has the worst performance and MPCoX-H-r the best performance on all data sets. The performance of MPCoX-H-a is similar to MPCoX-U and that of MPCoX-H-d is similar to MPCoX-I. This is because in the experiment, merge-by-union and merge-by-intersection often affected the same number of edges, therefore, MPCoX-H-a and MPCoX-H-d ended up choosing respectively merge-by-union and merge-by-intersection most of the time.



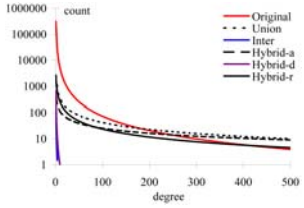
(a) EPINION with $\tau = 0.3$.



(b) EPINION with $\tau = 0.8$.



(c) FB with $\tau = 0.3$.

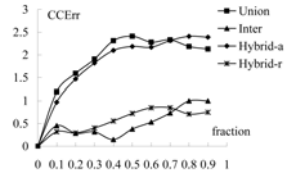


(d) FB with $\tau = 0.3$. Degree over 500 was cut off.

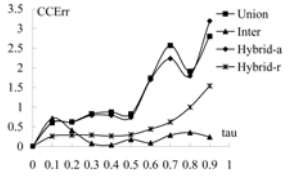
Figure 4.5: The Degree Distributions. The X-axis is the vertex degree. For readability, we ignored the degrees over 300. The Y-axis is the number of vertices of a given vertex degree. The results were measured at 70% of sensitive edges in the graph. The dark thickest solid line is the degree distribution of the original data set. The gray line is the histogram of the anonymized graph created by MPCoX-H-r. The lines of MPCoX-U and MPCoX-H-a are almost overlapped and the lines of MPCoX-I and MPCoX-H-d are very close.

Figure 4.5 shows the actual vertex degree histograms of some data sets. To improve readability, we only plotted the interpolation of the degree histogram. MPCoX-H-r has the best performance by producing degree distributions most similar to that of the original graphs. This is consistent with its performance on EMD. In Figure 4.5, the performance of MPCoX-U and MPCoX-H-a are similar to MPCoX-H-r for low threshold of edge confidentiality, but are worse than MPCoX-H-r

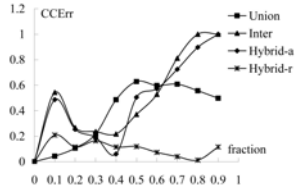
as the threshold of edge confidentiality becomes high. On the other hand, it is somewhat surprise to see MPCoX-I and MPCoX-H-d to have the worst performance on degree histograms, compared to their performance on EMD. A possible explanation is that as more and more edges are removed by MPCoX-I and MPCoX-H-d, many vertices will have the zero degree, causing significant drop of the counts at the low degree end. This may not cause big changes on EMD but may still changes the shape of the degree distribution. Based on these results, MPCoX-H-r may be a better choice for applications that are sensitive to vertex degrees.



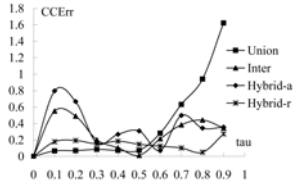
(a) EPINION with $\tau = 0.7$



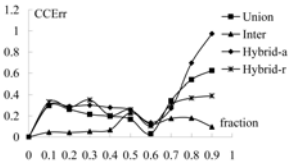
(b) EPINION with fraction being 0.5



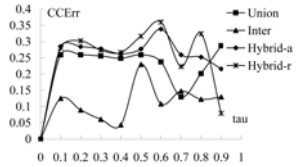
(c) FB with $\tau = 0.7$



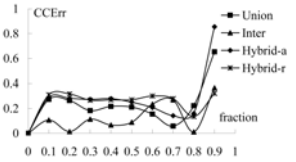
(d) FB with fraction being 0.5



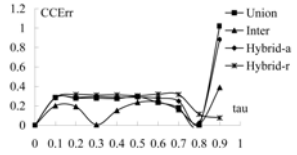
(e) SW-1 with $\tau = 0.7$



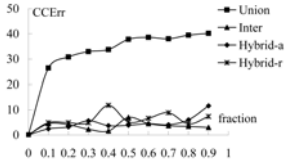
(f) SW-1 with fraction being 0.5



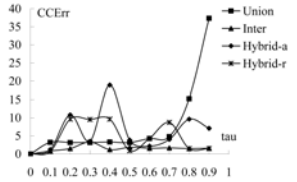
(g) SW-2 with $\tau = 0.7$



(h) SW-2 with fraction being 0.5



(i) ER with $\tau = 0.7$

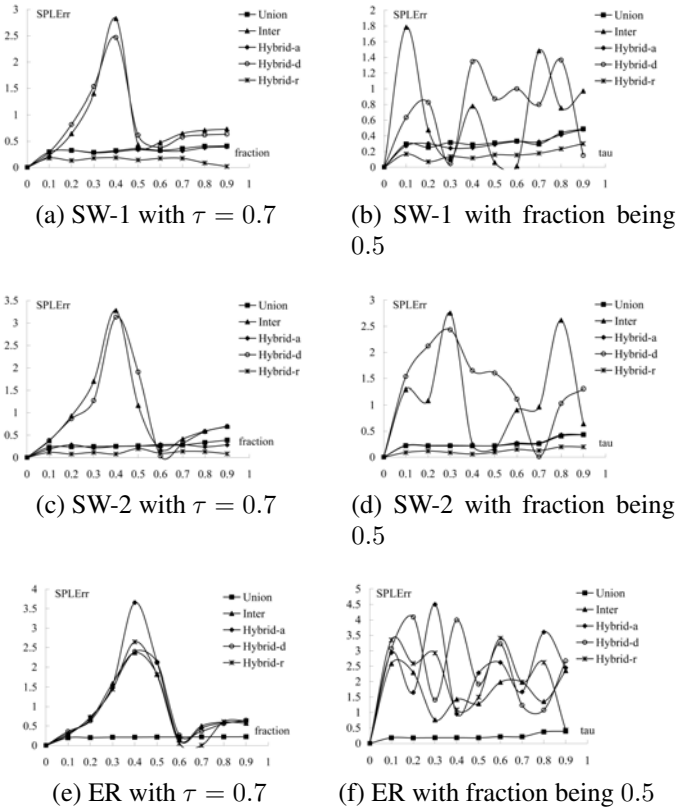


(j) ER with fraction being 0.5

Figure 4.6: Relative Error of Clustering Coefficient. The Y-axis of all subfigures is the relative difference of clustering coefficient measured between the anonymized graph and the original graph. The legend is the same as in Figure 4.4. The settings of edge confidentiality and fraction of sensitive edges are the same as in EMD measure comparison experiment. A greater relative difference value indicates worse utility.

Figure 4.6 shows the relative error of clustering coefficient, i.e. $\Delta cc(G, G') = |\frac{cc(G')-cc(G)}{cc(G)}|$, where G is the original graph, and G' is the anonymized one. The Figure does not include result

for MPCoX-H-d, because the range of values of those results are 10-100 times larger than the results of other execution types. Including those results in the Figure will make the results of other execution types too small to read. It is worth noting that among the original graphs, EPINION and SW-1 have similar clustering coefficient; FB and SW-2 have similar clustering coefficient; and ER has lower clustering coefficient than the other four data sets. According to the Figure, MPCoX-I performs the best and MPCoX-U the worst on average.



(a) SW-1 with $\tau = 0.7$ (b) SW-1 with fraction being 0.5

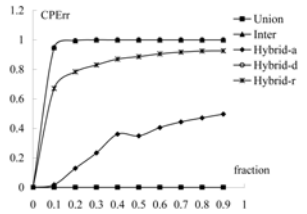
(c) SW-2 with $\tau = 0.7$ (d) SW-2 with fraction being 0.5

(e) ER with $\tau = 0.7$ (f) ER with fraction being 0.5

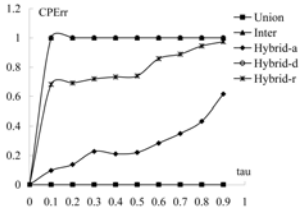
Figure 4.7: Relative Error of Average Shortest Path Length. The Y-axis of all subfigures is $\Delta lsp(G', G)$ between the anonymized graph G' and the original graph G. All other settings are the same as in Figure 4.4. A greater relative difference value indicates worse utility.

Figure 4.7 shows the relative error of average length of shortest paths in the anonymized graph, i.e. $\Delta alsp(G, G') = |\frac{alsp(G')-alsp(G)}{alsp(G)}|$, where G is the original graph, and G' is the anonymized one. Finding all-pairs shortest paths in a large graph such as the EPINION and FB data sets is extremely time-consuming. So, we use a sampling approach and the synthetic data sets. Specifically, a random sample of 50 pairs of connected vertices is taken from the graph, the shortest path

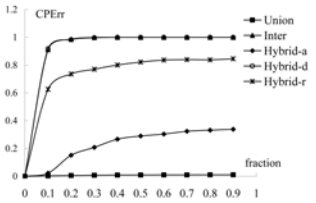
between each connected pair is found and the average of the 50 shortest paths is calculated. Our results indicate that MPCoX-H-r performs the best on SW-1 and SW-2 data sets. MPCoX-U and MPCoX-H-a have similar performance on SW-1 and SW-2 data sets. The performance of MPCoX-I and MPCoX-H-d are also similar and interesting. Specifically, for weaker privacy requirements, they both remove a small number of edges and cause many shortest paths to go through other edges and effectively increase the average length of shortest paths. However, for stronger privacy requirements, they will remove a larger number of edges and cause the graph to break up into many small pieces. As a result, the remaining connected pairs will have much shorter paths between them and the average length of the shortest paths is decreased.



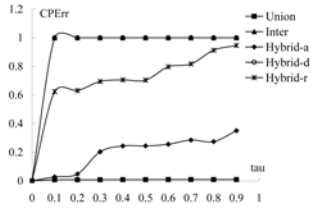
(a) EPINION with $\tau = 0.7$



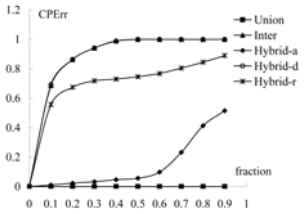
(b) EPINION with fraction being 0.5



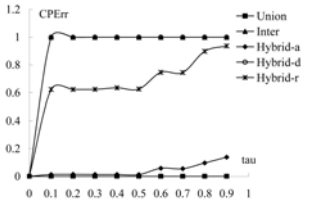
(c) FB with $\tau = 0.7$



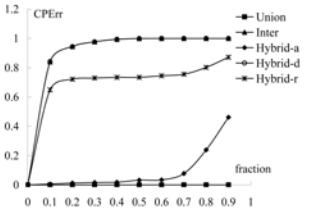
(d) FB with fraction being 0.5



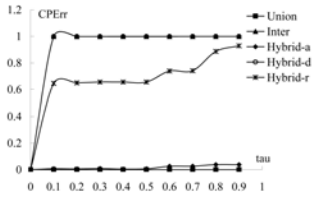
(e) SW-1 with $\tau = 0.7$



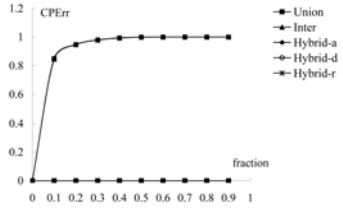
(f) SW-1 with fraction being 0.5



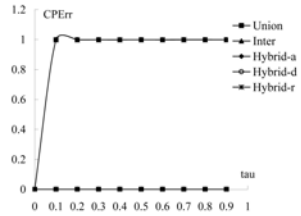
(g) SW-2 with $\tau = 0.7$



(h) SW-2 with fraction being 0.5



(i) ER with $\tau = 0.7$



(j) ER with fraction being 0.5

Figure 4.8: Relative Error of Connected Pairs. The Y-axis of all subfigures is $\Delta cp(G, G')$ measured between the anonymized graph G' and the original graph G. All other settings are the same as in Figure 4.4. A greater relative error value indicates worse utility.

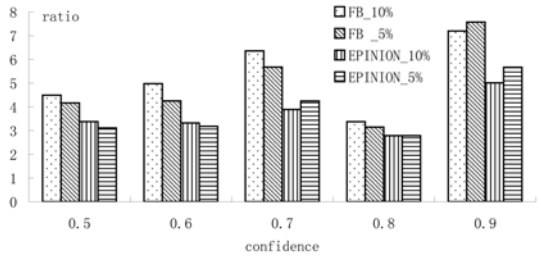
Figure 4.8 shows the relative error of connected pairs. MPCoX-U has the best performance on

all data sets, because adding edges does not change the number of connected pairs in a connected graph. Except FB data set, other four data sets are connected graphs originally. MPCoX-I and MPCoX-H-d performs similar on this measurement. This is because both methods remove similar numbers of edges in various tests. The performance of MPCoX-H-a and MPCoX-H-r are worse than MPCoX-U but better than MPCoX-I and MPCoX-H-d. This is because MPCoX-H-a and MPCoX-H-r usually remove less number of edges than MPCoX-I and MPCoX-H-d for the same privacy requirements.

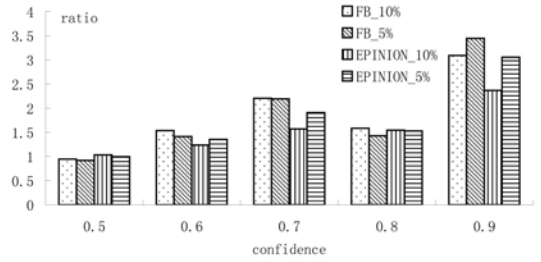
To account for the randomness in the results, the utility loss reported in this section is the average of 30 runs of each experiment. The standard deviations of the actual results were quite small and depend on the settings of the tests. For instance, the standard deviation of EMD for MPCoX-I running on FB data set with various testing conditions ranged between 0.0022 and 0.013, and that of clustering coefficient for MPCoX-U running on EPINION data set with various testing conditions ranged between 0.001 and 0.031. The highest standard deviation is of average shortest path length for MPCoX-U running on SW-1 data set, which ranged between 0.4 and 0.7.

4.4.5 Postponing the Checking of Edge Confidentiality

In this experiment, we investigate the advantages of postponing the checking of edge confidentiality until a merge plan is executed. We compare MPCoX with MPCoX-np, a version of MPCoX that uses the same heuristics used by Algorithm 4.4 to determine the merge-sets and the same execution strategy used by MPCoX to execute the VEC-merges, but checks the edge confidentiality immediately after each VEC-merge. We run MPCoX and MPCoX-np with different execution types on FB and EPINION data sets under various test settings.



(a) Execution Time. The Y-axis is the ratio of execution times of MPCoX-U-np and MPCoX-U.



(b) The Earth Move Distance. The Y-axis is the ratio of the numbers of edges added by MPCoX-U-np and MPCoX-U.

Figure 4.9: Performance of MPCoX-U-np vs MPCoX-U on EPINION and FB Data Sets. The X-axis is the threshold of edge confidentiality. The fractions of sensitive edges are 5% and 10%.

Figure 4.9 shows the performance of MPCoX vs MPCoX-np using execution type U. For the execution time, MPCoX-U is several times faster than MPCoX-U-np. For the utility loss, MPCoX-U needs to add less number of edges than MPCoX-U-np to satisfy given privacy requirements. Hence, checking edge confidentiality after the completion of a merge plan is more efficient than checking edge confidentiality after each merge. Similar results were also obtained for other execution types.

4.5 Conclusion

In this chapter, we present a new edge anonymization method, which uses a novel neighbor-set equivalence relation in the edge confidentiality as the privacy measure. We show that the neighbor-set equivalence is not only much easier to compute but it is also no weaker than the graph automorphism wrt the edge confidentiality. We introduce a number of techniques to improve the performance and reduce utility loss. These techniques include a partition map that succinctly rep-

resents the neighbor-set partition of the graph, two types of basic operations that merge vertices in the partition map and cause a set of edges to be add to or removed from the underlying graph, and a method that constructs and executes plans of sequences of the merge operations with a delayed check of edge confidentiality. We evaluate our method through extensive experiments using two real-world large social graphs and three random graphs with several graph utility measures. The results show that the techniques used by our method effectively improve graph utility and significantly reduce computational overhead.

Chapter 5: ANALYSIS OF PRIVACY DISCLOSURE OF SOCIAL GRAPHS UNDER INFORMATION EXTRACTION ATTACK

In this chapter, we will introduce a new information re-association attack model in 5.1, and the measurement of the attack effectiveness in social networks. Section 5.2 introduces an implementation of the attack model. Section 5.3 presents the experimental study about this attack.

5.1 Information Extraction Attack Model

We will define a model for any information extraction attack on a user of a social network. The model finds private information of the user from text documents (including web pages). Intuitively, the information about person is modeled as a set of typed features, some of which are publicly visible and others are hidden (and private). The information extraction attack takes as input the some visible features the victim and returns as output a set of features extracted from a set of documents. Each document that is related (or describes some aspect of) with the person will contain some of the person's features. From the perspective of the attack, only the information that can be extracted from a document will be considered a feature. The effectiveness of the attack can be measured by the amount of hidden features that are enclosed in the returned features. We now formalize these ideas.

Definition 14. (**Feature Set**) Given a document D and an information extraction method m. A feature w of type m in D is a piece of information that can be extracted from D by m. The feature set of type m (of D) is a multiset (or bag) (S, ρ), where $S = \{w | w \text{ is a feature of type } m \text{ in } D\}$ is a set of features and $\rho : S \to \mathbb{N}$ is the multiplicity mapping, such that $\rho(w)$ is the number of occurrences of w.

In the case where the multiplicity of every feature is one, we say that the mapping ρ is a trivial, and simply write (S, ρ) as S. Note that a feature needs not to be some text in a document. It can also have some non-text representation.

Features of the same type may not be completely distinct or the same. We use $sim_m(w_1, w_2)$ to denote feature similarity to measure the difference between features w_1 and w_2 of the same type m. Given two sets of features of the same type (possibly extracted from different text documents), we are interested in the similarity of the information described by them. Intuitively, if the two sets are similar enough (as defined by some threshold), we may consider them as describing the same person. There may be many ways to define the similarity of feature sets. The feature sets similarity can be defined as a function of features similarity. The features similarity and function for feature sets similarity can be chosen by users who implements the attack model in terms of specific features adopted. In this chapter, we use Eq. 5.1 to calculate feature sets similarity.

Definition 15. (**Feature Sets Similarity**) Let (S_1, ρ_1) and (S_2, ρ_2) be two feature sets of type m. Given the feature similarity sim_m of type m, the similarity between (S_1, ρ_1) and (S_2, ρ_2) is

$$sim_m((S_1, \rho_1), (S_2, \rho_2)) = \max\{\frac{\sum_{w_i \in S_1} \max_{w_j \in S_2}\{sim_m(w_i, w_j)\}}{|S_1|}, \frac{\sum_{w_j \in S_2} \max_{w_i \in S_1}\{sim_m(w_i, w_j)\}}{|S_2|}\}. \quad (5.1)$$

Intuitively, we pair each feature of a feature set with the most similar feature of the other feature set, and summarize the total similarities of those pairs. The result is normalized within the range of $[0, 1]$.

Using different types of information extraction methods, different feature sets can be extracted from a document form a feature vector.

Definition 16. (**Feature Vector**) Given a document D and a set $\mathcal{M} = \{m_1, \ldots, m_n\}$ of information extraction methods. The feature vector of D is $v_D =< (S_1, \rho_1), ..., (S_n, \rho_n) >$, where (S_i, ρ_i) is the feature set of type m_i extracted from D.

The concept of feature vector is sufficiently general to represent information that can be extracted from any text document that describes a person.

In this attack model, we consider several types of feature vectors. First, there is a person

vector for the victim, which contains all features in the person's profile. Some of these features are publicly visible and others are private. Second, the input to the attack method is a query vector, which contains some publicly visible features of the victim as the starting point to identify relevant documents. Third, the attack method may obtain a document vector from each document, which contains features extracted from the document. Finally, the attack method returns a discovered vector, which contains features extracted from a set of documents and are believed to belong to the victim. In a person vector, query vector or discovered vector, the mapping ρ on every feature set is trivial.

We use the similarity of feature sets to define a similarity measure of feature vectors. Again, this framework allows other measurement alternatives. In this chapter, we use the following measure for its simplicity and generality.

Definition 17. (**Similarity of Feature Vectors**) Let $v =< (S_1, \rho_1), \ldots, (S_n, \rho_n) >$ and $v' =< (S'_1, \rho'_1), \ldots, (S'_n, \rho'_n) >$ be two feature vectors in feature space $\mathcal{M} = \{m_1, \ldots, m_n\}$. The similarity of v and v' is defined as

$$sim(v, v') = \sum_{i=1}^{n} a_i \cdot sim_{m_i}((S_i, \rho_i), (S'_i, \rho'_i)) \tag{5.2}$$

where $0 \leq a_i \leq 1$ is a weight constant.

Two feature vectors that describe the same person can be combined into one feature vector using the feature vector sum operation. Note that the feature vector sum operation can only be applied on two feature vectors extracted from the same methods. Let two feature vectors be $u =< u_1, ..., u_n >$ and $v =< v_1, ..., v_n >$, the sum of the two vectors is $w =< w_1, ..., w_n >$. For any $w_i = (S_w, \rho_w) \in w$, $S_w = S_u \cup S_v$, where $u_i = (S_u, \rho_u)$, $v_i = (S_v, \rho_v)$, and for each $s \in S_w$, $\rho_w(s) = \rho_u(s) + \rho_v(s)$ if $s \notin S_u$ (or $s \notin S_v$) , $\rho_u(s) = 0$ (or $\rho_v(s) = 0$).

The effectiveness of information extraction attack is measured by the precision and recall [52] of the discovered vector as compared to the person vector. The precision measures the correctness of the discovered previously unknown features, and the recall measures the completeness of the

discovered correct features.

Let S be a feature set in a discovered vector, S' be the feature set in the person vector that is of the same type as S. The precision of a feature set is $prec(S) = \frac{|S \cap S'|}{|S|}$, if $S \neq \emptyset$ and 0, otherwise. The recall of a feature set is $recall(S) = \frac{|S \cap S'|}{|S'|}$, if $S' \neq \emptyset$ and 0, otherwise.

Definition 18. (**Attack Effectiveness**) Let $u =< u_1, \ldots, u_n >$ be a discovered vector returned by an information extraction attack started by a query vector $w =< w_1, ..., w_n >$, and $v =< v_1, \ldots, v_n >$ be the person vector of a victim. The precision of the attack on the victim is

$$prec = \frac{\sum_{i=1, w_i \neq v_i}^{n} prec(u_i)}{|\{w_i \in w | w_i \neq v_i\}|} \tag{5.3}$$

and the recall of the attack on the victim is

$$recall = \frac{\sum_{i=1, w_i \neq v_i}^{n} recall(u_i)}{|\{w_i \in w | w_i \neq v_i\}|} \tag{5.4}$$

, where $prec$ and $recall$ equal to 0 if $|\{w_i \in w | w_i \neq v_i\}| = 0$.

Example 7. The information about Mike Jones on Facebook can be represented by the person vector v_{mike} =<{ “Mike Jones” }, { “ABC University”, “XYZ Inc” }, { (_, “New York”, “NY”, _), (_, “Austin”, “TX”, _) }, { “male” }>, where all multiplicity mappings are trivial. The types of the feature sets from left to right are person name, organization, location, and gender. Based on the person vector v_{mike}, we can define a query vector $q_{Mike} =< \{“MikeJones”\}, \{“ABC\ University”\}, \emptyset, \emptyset >$. If an attack returns the discovered vector v'_{mike} =<∅,∅,{(_, “New York”, “NY”, _)}, {“male”}>, the attack precision will be 0.67 and the attack recall will be 0.5.

5.2 Web-Based Private Information Extraction

The framework of the information extraction attack we describe in this section is shown in Figure 5.1. This Web-based Private Information Extraction (or WebPIE for short) consists of four components: query submission, feature extraction, clustering and selection, and refinement.

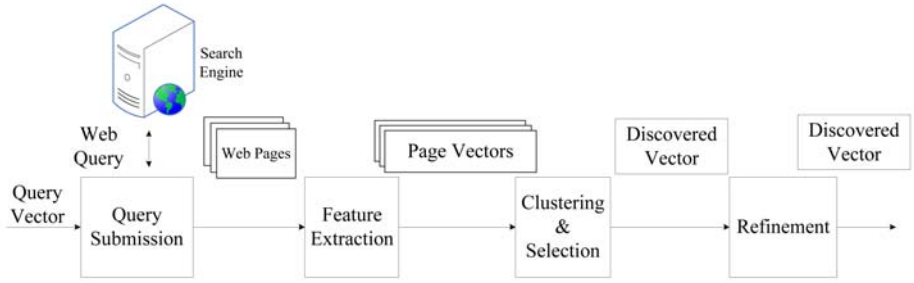


Figure 5.1: Framework of Information Extraction Attack

5.2.1 Web Query Submission

The attack starts with a query vector. The features in the query vector is chosen by the attacker from the publicly visible information of a social network about the victim. The query vector is used by the component to construct an initial web query for a search engine. Alternatively, the initial web query may also be manually submitted to the component by the attacker.

Example 8. Given the query vector of Example 7, the web query to search engine will be "Mike Jones ABC University".

The component takes either a query vector (or a user entered web query) and returns a set of web pages that are obtained from general search engines. Its tasks include generating the web query from a query vector, submitting the web query to search engines, and retrieving web pages according to search results.

5.2.2 Feature Extraction

The feature extraction component takes the set of web pages returned by the query submission component and returns a set of page vectors. An page vector is a feature vector extracted from a web page. First, each web page is decomposed into segments of text according to the web query and a user-specified window size. A feature vector will be extracted from each segment, and called segment vector.

The following types of features are extracted from each web page, among them only the named entity types are included in person vectors, the other features are provided to facilitate the work in

subsequent components.

- *Web Page URL*. This is the unique link to the web page. This feature is returned by query submission component. The extraction method for it is trivial.

- *Web Page Title*. This is a set of words located in a web page within the <title> element in the <head> element. Given a web page, the title can be obtained by parsing the web page into an HTML parse tree and locating the HEAD element in the tree structure.

- *Web Page Summary*: This is a summary of the web page. The content is typically enclosed in the <meta> tag of <head> element. Similar to title feature extraction, the content is extracted from the <meta> tag with description property in the parsed web page. Then, part-of-speech (POS) tags is added to each word via POS Tagger [11]. The summary is a set of nouns selected from the tagged content.

- *Compound Key Word (CKW)*. A compound word is a sequence of nouns appearing together in a text. Each compound word in a text can be assigned an importance score. A compound key word is a compound word whose importance score is higher than a threshold. This feature can be exacted by using the method in [64].

- *Named Entity (NE)*. A named entity is an information item [44] that has a unique textural representation. Well-known NEs include names of person, organization, location and numeric patterns for time, date, currency and percentage. Each of these is treated as a feature type and can be extracted using methods described in [44]. Before extracting the location, organization and personal names the NE recognizers [6] is applied and the recognized features are normalized according to their semantics. The normalization is needed because some NEs may have different syntactic form with the identical meaning. For example, “Texas” or “TX” can both refers to the State of Texas.

The Web page URL, title, and summary describe the information about a Web page, while CKWs and NEs describe the information about a segment. Consequently, the feature extraction methods

are divided into page methods and segment methods. After a web page is segmented, segment methods are applied to extract segment vectors from all segments, and page methods are applied to extract a feature vector about the Web page. These vectors are then integrated into one page vector by using feature set sum operation. The set of page vectors is then returned.

Example 9. A feature vector for Mike Jones from a web page including Web page URL, title, and summary can be v_1 =<{("http://ABC.edu")}, {("ABC news")}, {("Sports events"), ("Basketball season")}, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$}>. A segment vector from the same web page can be v_2 =<$\emptyset$, $\emptyset$, $\emptyset$, {("Basketball")}, {("New York")}, {("ABC University"),("XYZ University")}, $\emptyset$}>. Another segment vector can be v_3 =<$\emptyset$, $\emptyset$, $\emptyset$, {("Players")}, {("New York")}, $\emptyset$, {("Mike",1),("David",2)}>. The segment vectors include CKWs, and named entities, i.e. locations, organizations, and persons. Then a page vector is v =<("http://ABC.edu")}, {("ABC news")}, {("Sports events"), ("Basketball season")}, {("Basketball"), ("Players")}, {("New York",2)}, {("ABC University"),("XYZ University")}, {("Mike"),("David")}> integrated by iteratively performing feature set sum operations on the feature vectors.

5.2.3 Clustering and Selection

This component takes a set of page vectors from the feature extraction component and returns one discovered vector. We assume that each web page describes a single person. However, the web pages obtained by query submission component may describe multiple persons due to the namesake issue [8, 26, 42, 53, 64]. Thus, the first task of this component is to identify document vectors that describes the victim. The second task is to integrate identified page vectors into a discovered vector.

Due to the limited features in the query vector, page vectors that are consistent with the query vector may not all describe the same person. To determine whether two page vectors describe the same person is a difficult problem without additional information. We solve this problem by a heuristic, that is, web pages that describe the same person will have much duplicate information, and those describing different persons will contain some significantly different information. Based

on this heuristic, we find clusters of page vectors using the hierarchical agglomerative clustering algorithms [22].

Once the set of document vectors are clustered, one cluster can be chosen based on one of the following strategies to form the discovered vector.

- Choose the cluster that is the most similar to the query vector.

  Here the similarity of two clusters is defined as

  $$sim(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{v_k \in C_i} \sum_{v_l \in C_j} sim(v_k, v_l)$$

  , where $sim(v_k, v_l)$ is the similarity of feature vectors v_k and v_l. If only one cluster has the maximum similarity, choose it. If more than one cluster has the maximal similarity, choose the largest one among them.

- Choose the largest cluster.

  If there is only one largest cluster, choose it. Otherwise, choose the largest cluster that is the most similar to the query vector.

- Choose the largest cluster and the top-k other similar clusters.

  First, choose the largest cluster. Then, choose k other clusters that are most similar to the first chosen cluster.

After clusters are selected, feature vector sum operations are iteratively applied on all page vectors in the clusters to obtain the discovered vector.

Example 10. A discovered vector for Mike Jones can be <{("New York", 5), ("Text", 2)}, {("ABC University", 4), ("XYZ University", 2)}, {("Mike", 3),("David", 1)}>

5.2.4 Refinement

The goal of feature refinement is to remove false features in the discovered vector returned from the feature clustering and selection component, because feature extraction methods may produce errors, such as named entity extraction methods may produces NE recognition errors [44]. For example, an extraction method notices the person entity "Mike Jones", but give it a wrong label as an organization entity; Or recognize a location entity like "Text" in Example 10 which is nonsense indeed.

We propose two feature refinement strategies based on the heuristic that true features are unique, while false features could vary a lot. So true features should have high multiplicity in the feature set compressed from similar segment vectors, while false features has low multiplicity.

1. Removing feature w with $\rho(w)$ less than a certain multiplicity threshold. This strategy probably removes all features from the person vector given a threshold.

2. Removing feature w with relatively lower $\rho(w)$ than other features in a feature set. This strategy avoids the problem resulted by the first strategy. It calculates the range of fraction of each feature's multiplicity in a feature set, and removes the features with lower multiplicity fraction than a value within the range.

Example 11. A discovered vector in Example 10 will be refined as <{("New York", 5)}, {("ABC University", 4)}, {("Mike", 3)}> if we use the first strategy.

5.3 Experiments

In this section, we experimental study the privacy disclosure under information extraction attack. We use the FB data set which was crawled from Facebook.com in 2011. This data set contains 10,000 users profiles. Each profile has user name, gender, current resident city, home town, employer, and graduate colleges. We carry out two sets of experiments on FB data set. The first set of experiments is to verify the motivation of our study in Section 5.3.1. The second set of experiments is to show the attack effectiveness via our implemented WebPIE system.

People in different organizations could have different quantity of online information, which must impact the volume of discovered information. We categorize users in terms of network information, because most of users in the data set have this type of information. The network information on Facebook usually refers to the graduation colleges or organizations people participate in. Considering most profiles have college as network information, we group the data set according to the ranking of colleges. Users from colleges in Top 50 rankings belong to one group. Users from other colleges belong to the other group.

5.3.1 Overlap of Personal Information in Multiple Social Networks

We carry out two experiments to verify that users place the same type of information in their profiles on different social network sites. We analyze three types of information, i.e. graduation colleges, location and employer, because they are very common in profiles.

In the first experiment, we randomly obtain 5 samples with each having 30 profiles from FB data set. We get the user name from the profile to build a web query and search the user's information in Google.com. For each FB user, we read top 20 web pages except ones from Facebook.com among the search results, and find web pages pertaining to the FB user. We use some information, such as pictures, graduation schools or employers, and perform some reasoning to manually determine whether a web page pertains to the FB user or not. As a result, we find on average 37.3% users place at least one of the three types of information on some other social websites, such as LinkedIn.com, MySpace.com, Hi5.com and so on. The standard deviation of this evaluation on 5 samples is 8.2%.

In the second experiment, we choose from Top 50 colleges the users whose network names on Facebook are Harvard, Columbia, Cornell, Yale, or UPenn. For each network, we also select 30 users. We use the same method as the first experiment to find personal information on the search web pages. The results show that on average 69% users place at least one of the three types of information on some other websites. The standard deviation of this experiment is 5%.

We contrast our experiments with the similar studies in [45,46]. Patriquin [46] shows that 64%

of Facebook users are also present on MySpace. Narayanan and Shmatikov show that 24% of the names associated with Twitter accounts occur in Flickr, while 5% of the names associated with Flickr accounts occur in Twitter. Their studies reinforce our motivation that overlap of personal information may exist among multiple social network sites because memberships overlap among them. But they do not examine the specific information among them.

5.3.2 Attack Effectiveness of a WebPIE System

We implement a WebPIE system. In query submission component, we develop an HTTP client to send web queries to and receive search results from google[1]. In feature extraction component, we choose location, organization and personal names as NE features. We used OpenNLP[2] to extract named entities. In clustering and selection component, we implemented the hierarchical agglomerative clustering algorithm and three selection strategies, but only reported the result of choosing the largest cluster and the top-k other similar clusters, where $k = 2$, to avoid too much noises. In the refinement component, we implemented both strategies, and only reported the result of employing absolute threshold.

We perform several experiments to measure the effectiveness of the attack on two groups of users. One group of users are from TOP 50 ranking schools, called top school (TS) data set. The other group of users are uniformly sampled from the entire data set, shortly called any school (AS) data set. For each group, we get 5 samples with each having 30 users. In Section 5.3.2, we study the difference of attack effectiveness on the two groups. In Section 5.3.2, we construct web queries with different features to study the impact of web queries on the attack effectiveness.

Effectiveness of Attack on Different Groups of Users

In this experiment, we use personal name and network information to construct a web query. The person vector includes location and organization feature sets. The location is the state of current city in FB set. The organization contains the high school, college, graduate school and employer.

[1] http://www.google.com
[2] http://opennlp.apache.org/

Table 5.1: Attack Effectiveness on TS and AS Data Set: "Avg" is short for average; "Std" is shorted for standard deviation.

	TS		AS	
	Recall	Precision	Recall	Precision
Avg	0.42	0.38	0.25	0.20
Std	0.10	0.11	0.10	0.08
Avg (Location)	0.48	0.35	0.31	0.26
Std (Location)	0.12	0.11	0.11	0.10
Avg (Organization)	0.36	0.40	0.18	0.13
Std (Organization)	0.09	0.10	0.09	0.07

Different person may have different number of features in the organization feature set. Table 5.1 reports the results of this experiment. The first two rows are the results measured on the person vector and discovered vector. We measure the location and organization separately to analyze the effectiveness on different type of features.

In a summary, the attack is more effective on TS data set than AS data set, which is expected because more TS users place their information on multiple websites as showed in Section 5.3.1. Thus, personal information in TS is easier to be extracted from the Web. In general, the recall is slightly better than the precision because person vector has not too many features. There is only one location feature, and at most four organization features in the person vector. So it is easier to achieve completeness than correctness.

Web Query Impact on Attack Effectiveness

In this experiment, we use different features to construct web queries, and compare the attack effectiveness. We choose users from TS data set because it is easier to the better performance on attack effectiveness. The selected users must have current city, and 2 or more networks in their profiles. We construct the various types of queries using only name (NAME) or combining the name with either one of the following features: (1) 1 network (1N); (2) 2 networks (2N); (3) all networks (AN); (4) current city (LOC). As a matter of fact, we construct the query based on many combinations of the network features. Here, we only report the results of multiple network features

Table 5.2: Web Query Impact on Attack Effectiveness: "Avg" is short for average; "Std" is shorted for standard deviation.

	Avg		Std	
Query	Recall	Precision	Recall	Precision
NAME	0.2	0.15	0.04	0.05
1N	0.41	0.34	0.09	0.06
2N	0.30	0.16	0.10	0.05
AN	0.23	0.13	0.09	0.05
LOC	0.10	0.08	0.02	0.03

following the order in their profiles, which is already able to support our conclusion.

From Table 5.2, we can see that placing more features in the query does not necessarily improve the effectiveness although it can intuitively improve the quality of clustering and selection component. The reason is that the search engine can't return accurate results when the query contains many words.

Table 5.2 also illustrates that using query with location feature achieves bad performance. As mentioned before, the network feature usually refers to organization. Intuitively, there is more possible that persons have the same name in one city than ones in one organization. So it is more difficult to disambiguate the person from other persons with the same name using location than using network.

5.4 Conclusion

In this chapter, we describe a privacy attack that utilizes web search, NER and name disambiguation techniques to automate the discovery of private information of a social network user. From the experiment, we find that among the Facebook users that discloses location, education, and employer information to public, 64% of them put the same information in other social networks, e.g. LinkedIn, MySpace, or Hi5. Our information extraction attack can extract personal information for social network users. The attack effectiveness is impacted by the ways of web query construction and depends on the quantity of victim's online information. Compared to previously studied information re-association attacks [29, 45, 67], this new attack places more challenge on privacy

policy settings of social network sites.

Chapter 6: CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

In this dissertation, we present two studies about privacy protection in social graphs.

The first study in Chapter 3 proposes a privacy requirement for edge anonymity based on graph partition. The privacy measure does not depend on a specific equivalence relation to partition the graph. Thus, it can be applied to evaluate a wide range of edge anonymization methods against vertex re-identification attack. We develop two methods to protect edge privacy. The first method in Chapter 3 considers an equivalence relation on vertex degree. Our experiments, based on three real-world social networks and several utility measures, show that these algorithms can effectively protect sensitive edges and can produce anonymized graphs that have acceptable utility. The second work uses a strong equivalence relation based on neighbor-set to partition the graph. The new edge anonymization method can protect edges against any vertex re-identification attack based on structural information. Besides, the method achieves great performance in large scale social graphs.

The second study in Chapter 5 designs a new information re-association attack which extracts personal information from the Web. The attack does not utilize sophisticated information extraction techniques, but still is able to discover personal information of some social network users. The experiments show that the effectiveness of the attack depends on the information quantity of social network users on the Web, and the way to construct the web queries to launch the attack.

6.2 Future Works

In this research area, one direction is exploring new privacy attacks. The other direction is developing new protection methods against these attacks.

6.2.1 Privacy Attacks Exploration

This section introduces potential vertex re-identification and information re-association attacks.

Vertex Re-Identification Attacks

Most existing studies only consider the vertex re-identification attacks on a simple graph that is a graph without labels on vertices and edges. In a real world, published graph can have labels on vertices or edges to represent different types of information. For example, the vertices on Facebook could have birthday, location, interest information; the edges in the "Web of trust" graph from Epinion.com can represent trust or distrust relation. Protecting labeled graphs is more difficult than simple graph because an adversary may have more opportunities to re-identify vertices using various labels.

Many existing studies except [69] do not consider periodical publication of social graphs. Some applications require a graph to be periodically published because vertices and edges can change over time. For instance, a fundamental open question [15] in the analysis of social networks is to understand the interplay between similarity of user behaviors and their social relations. Since user relations and behaviors vary from time to time, people have to study the question in multiple graph snapshots.

Information Extraction Attack

According to the experiment result in Section 5.3.1, personal information overlaps heavily among social network sites for some users. Thus, we can limit the search range to social network sites. Many social network sites provide people search engine, and even development API (Application Programming Interface). So instead of general web search engine, we can consider utilizing the specific people search engines in the query submission component.

6.2.2 Privacy Preservation Methods

Even though our edge anonymization method can be easily extended to protect vertex identity, the privacy measure we proposed does not evaluate the protection degree for vertex identity. It is necessary to propose a more general privacy measure for the entire graph. The new measure may lead to designing another preservation approach.

As mentioned in [69], the anonymization methods for "one-time" release of social graphs can not guarantee the privacy. Also, the anonymization methods for simple graph can't protect the privacy of labeled graph. Considering the wide application of social graphs, we should consider the two cases in the future.

Because many anonymization methods do not consider the specific application of social graphs, the measure of utility loss may not be practical. Also, it is difficult to find a reasonable utility measure to evaluate the loss of all kinds of utilities. So, another research direction could be focusing on a particular application, and designing graph anonymization methods to achieve good utility just for the application.

BIBLIOGRAPHY

[1] C. Aggarwal and P. Yu. A condensation approach to privacy preserving data mining. In *International Conference on Extending Database Technology*, pages 183–199, 2004.

[2] Charu C. Aggarwal. On k-anonymity and the curse of dimensionality. In *International Conference on Very Large Data Bases*, pages 901–909, 2005.

[3] Gagan Aggarwal, Tomas Feder, Krishnaram Kenthapadi, Rajeev Motwani, Rina Panigrahy, Dilys Thomas, and An Zhu. Approximation algorithms for k-anonymity. *Journal of Privacy Technology*, 2005.

[4] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *ACM SIGMOD International Conference on Management of Data*, pages 439–450. ACM, 2000.

[5] Yong-Yeol Ahn, Seungyeop Han, Haewoon Kwak, Sue Moon, and Hawoong Jeong. Analysis of topological characteristics of huge online social networking services. In *International World Wide Web Conference*, 2007.

[6] Apache.org. http://incubator.apache.org/opennlp/documentation/manual/opennlp.htm, 2010.

[7] Lars Backstorm, Cynthia Dwork, and Jon Kleinberg. Wherefore art thou r3579x? anonymized social networks, hidden patterns, and structural steganography. In *International World Wide Web Conference*, 2007.

[8] Ron Bekkerman and Andrew McCallum. Disambiguating web appearances of people in a social network. In *International World Wide Web Conference*, 2005.

[9] Smriti Bhagat, Graham Cormode, Balachander Krishnamurthy, and Divesh Srivastava. Class-based graph anonymization for social network data. In *International Conference on Very Large Data Bases*, 2009.

[10] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *INTERNATIONAL JOURNAL OF MATHEMATICAL MODELS AND METHODS IN APPLIED SCIENCES*, 1:300–307, 2007.

[11] Eugene Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 18 (4):33–44, 1997.

[12] James Cheng, Ada Wai-Chee Fu, and Jia Liu. K-isomorphism: Privacy preserving network publication against structural attacks. In *SIGMOD*, 2010.

[13] Sangdui Choi, editor. *Introduction to Systems Biology*. Humana Press Inc., 2007.

[14] COA-Dataset. http://www.cs.helsinki.fi/u/tsaparas/MACN2006/data-code.html.

[15] David Crandall, Dan Cosley, Daniel Huttenlocher, Jon Kleinberg, and Siddharth Suri. Feedback effects between similarity and social influence in online communities. In *International Conference on Knowledge Discovery and Data Mining*, 2008.

[16] L. da F. Costa, F. A. Rodrigues, G. Travieso, and P. R. Villas Boas. Characterization of complex networks: A survey of measurements. *Advances in Physics*, 56(1):167–242, 2007.

[17] Sudipto Das, Omer Egecioglu, and Amr El Abbadi. Anonimos: An lp based approach for anonymizing weighted social network graphs. *IEEE TRANSACTIONS OF KNOWLEDGE AND DATA ENGINEERING*, 2010.

[18] Jeremy Day, Yizhou Huang, Edward Knapp, and Ian Goldberg. Spectre: Spot-checked private ecash tolling at roadside. In *Workshop on Privacy in the Electronic Society*, 2011.

[19] John R. Douceur. The sybil attack. In *Proceedings of the International Workshop on Peer–To–Peer Systems*, 2001.

[20] Epinions-Dataset. http://www.trustlet.org/wiki/Downloaded_Epinions_dataset.

[21] P. Erdos and A. Renyi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5:17–61, 1960.

[22] Alberto Fernandez and Sergio Gomez. Solving non-uniqueness in agglomerative hierarchical clustering using multidendrograms. *Journal of Classification*, 25:43–65, 2008.

[23] Philip W. L. Fong. Preventing sybil attacks by privilege attenuation: A design principle for social network systems. In *IEEE Symposium on Security and Privacy*, 2011.

[24] Mark Granovetter. The impact of social structure on economic outcomes. *The Journal of Economic Perspectives*, 19(1):33–50, 2005.

[25] Ralph Gross and Alessandro Acquisti. Information revelation and privacy in online social networks. In *Proceedings of Workshop on Privacy in the Electronic Society (WPES)*, 2005.

[26] Xianpei Han and Jun Zhao. Web personal name disambiguation based on reference entity tables mined from the web. In *Proceeding of the eleventh international workshop on Web information and data management*, 2009.

[27] Michael Hay, Gerome Miklau, David Jensen, Don Towsley, and Philipp Weis. Resisting structural re-identification in anonymized social networks. In *International Conference on Very Large Data Bases*, 2008.

[28] YihChun Hu and Helen J. Wang. A framework for location privacy in wireless networks. In *ACM SIGCOMM Asia Workshop*, 2005.

[29] Apu Kapadia Huina Mao, Xin Shuai. Loose tweets: An analysis of privacy leaks on twitter. In *Workshop on Privacy in the Electronic Society*, 2011.

[30] KDDCUP-Dataset. http://www.cs.cornell.edu/projects/kddcup/datasets.html.

[31] Peter Klerks. The network paradigm applied to criminal organisations: Theoretical nitpicking or a relevant doctrine for investigators? recent developments in the netherlands. *INSNA Connections*, 3:53–65, 2001.

[32] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *International Conference on Knowledge Discovery and Data Mining*, 2006.

[33] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: Efficient fulldomain k-anonymity. In *ACM SIGMOD International Conference on Management of Data*, 2005.

[34] Jure Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The dynamics of viral marketing. *ACM Transactions on the Web*, 1:228–237, 2007.

[35] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *International Conference on Knowledge Discovery and Data Mining*, 2008.

[36] Ninghui Li and Tiancheng Li. t-closeness: Privacy beyond k-anonymity and l-diversity. In *IEEE International Conference on Data Engineering*, 2007.

[37] Kun Liu and Evimaria Terzi. Towards identity anonymization on graphs. In *ACM SIGMOD International Conference on Management of Data*, pages 93–106, 2008.

[38] Lian Liu, Jie Wang, Jinze Liu, and Jun Zhang. Privacy preservation in social networks with sensitive edge weights. In *Proceedings of the Ninth SIAM International Conference on Data Mining*, 2009.

[39] Francois Lorrain and Harrison C. White. Structural equivalence of individuals in social networks. *Journal of Mathematical Sociology*, 1:49–80, 1971.

[40] A. Lubiw. Some NP-complete problems similar to graph isomorphism. *SIAM Journal on Computing*, 10:11–21, 1981.

[41] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. ℓ-diversity: Privacy beyond k-anonymity. In *IEEE International Conference on Data Engineering*, 2006.

[42] Gideon S. Mann and David Yarowsky. Unsupervised personal name disambiguation. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL*, 2003.

[43] Shrirang Mare, Jacob Sorber, Minho Shin, Cory Cornelius, and David Kotz. Adapt-lite: Privacy-aware, secure, and efficient mhealth sensing. In *Workshop on Privacy in the Electronic Society*, 2011.

[44] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30:3–26, 2007.

[45] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy*, 2009.

[46] Alex Patriquin. Connecting the social graph: member overlap at opensocial and facebook. http://blog.compete.com/2007/11/12/connecting-the-social-graph-member-overlap-at-opensocial-and-facebook/, 2007.

[47] Alfredo Rial and George Danezis. Privacy-preserving smart metering. In *Workshop on Privacy in the Electronic Society*, 2011.

[48] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 2:99–121, 2000.

[49] P. Samarati and L. Sweeney. Protecting privacy when disclosing information:k-anonymity and its enforcement through generalization and suppression. In *Proc. of the IEEE Symposium on Research in Security and Privac*, 1998.

[50] John P. Scott. *Social Network Analysis: A Handbook*. Sage Publications Ltd, 2000.

[51] Lisa Singh and Justin Zhan. Measuring topological anonymity in social networks. In *IEEE International Conference on Granular Computing*, 2007.

[52] Amit Singhal. Modern information retrieval: A brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–43, 2001.

[53] Elena Smirnova, Konstantin Avrachenkov, and Brigitte Trousse. Using web graph structure for person name disambiguation. In *In Third Web People Search Evaluation Forum (WePS-3)*, 2010.

[54] Ke Wang, Philip S. Yu, and Sourav Chakraborty. Bottom-up generalization: A data mining solution to privacy protection. In *IEEE International Conference on Data Mining*, 2004.

[55] Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic spreading in real networks: An eigenvalue viewpoint. In *2003 International Symposium of Reliable Distributed System*, 2003.

[56] Duncan J. Watts and Steven Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[57] Gilbert Wondracek, Thorsten Holz, Engin Kirda, and Christopher Kruegel. A practical attack to de-anonymize social network users. In *IEEE Symposium on Security and Privacy*, 2010.

[58] Raymond Chi-Wing Wong, Jiuyong Li, Ada Wai-Chee Fu, and Ke Wang. (α, k)-anonymity: An enhanced k-anonymity model for privacy-preserving data publishing. In *International Conference on Knowledge Discovery and Data Mining*, 2006.

[59] Wentao Wu, Yanghua Xiao, Wei Wang, Zhenying He, and Zhihui Wang. K-symmetry model for identity anonymization in social networks. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 111–122, 2010.

[60] Xiaokui Xiao and Yufei Tao. Personalized privacy preservation. In *ACM SIGMOD International Conference on Management of Data*, pages 229–240, 2006.

[61] Xiaokui Xiao and Yufei Tao. m-invariance: Towards privacy preserving re-publication of dynamic datasets. In *ACM SIGMOD International Conference on Management of Data*, 2007.

[62] Akira Yamada, Tiffany Hyun-Jin Kim, and Adrian Perrig. Exploiting privacy policy conflicts in online social networks. Technical report, Carnegie Mellon University, 2012.

[63] Xiaowei Ying and Xintao Wu. Randomizing social networks: a spectrum preserving approach. In *SIAM International Conference on Data Mining*, 2008.

[64] Minoru Yoshida, Masaki Ikeda, Shingo Ono, Issei Sato, and Hiroshi Nakagawa. Person name disambiguation by bootstrapping. In *ACM SIGIR International Conference on Information Retrieval*, 2010.

[65] Lijie Zhang and Weining Zhang. Edge anonymity in social network graphs. In *IEEE International Conference on Social Computing*, 2009.

[66] Elena Zheleva and Lise Getoor. Preserving the privacy of sensitive relationships in graph data. In *ACM International Workshop on Privacy, Security, and Trust in KDD (PinKDD)*, 2007.

[67] Elena Zheleva and Lise Getoor. To join or not to join: The illusion of privacy in social networs with mixed public and private user profiles. In *International World Wide Web Conference*, 2009.

[68] Bin Zhou and Jian Pei. Preserving privacy in social networks against neighborhood attacks. In *IEEE International Conference on Data Engineering*, 2008.

[69] Lei Zou, Lei Chen, and M. Tamer Ozsu. K-automorphism: A general framework for privacy preserving network publication. In *International Conference on Very Large Data Bases*, 2009.

VITA

Lijie Zhang was born in Chongqing, China. She graduated from the College of Computer Science and Technology at Beijing University of Posts and Telecommunications (BUPT) in 2002, received her B.S. degree and enrolled into the masters program in computer science at BUPT. In 2005, she received her M. Sc. degree. In the same year, she started pursuing her Ph.D. degree in the Department of Computer Science at University of Texas at San Antonio (UTSA).