# K-Isomorphism: Privacy Preserving Network Publication against Structural Attacks

James Cheng
School of Computer Engineering
Nanyang Technological University, Singapore
jcheng@acm.org

Ada Wai-Chee Fu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
adafu@cse.cuhk.edu.hk

Jia Liu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
jliu@cse.cuhk.edu.hk

## ABSTRACT

Serious concerns on privacy protection in social networks have been raised in recent years; however, research in this area is still in its infancy. The problem is challenging due to the diversity and complexity of graph data, on which an adversary can use many types of background knowledge to conduct an attack. One popular type of attacks as studied by pioneer work [2] is the use of embedding subgraphs. We follow this line of work and identify two realistic targets of attacks, namely, NodeInfo and LinkInfo. Our investigations show that $k$-isomorphism, or anonymization by forming $k$ pairwise isomorphic subgraphs, is both sufficient and necessary for the protection. The problem is shown to be NP-hard. We devise a number of techniques to enhance the anonymization efficiency while retaining the data utility. The satisfactory performance on a number of real datasets, including HEP-Th, EUemail and LiveJournal, illustrates that the high symmetry of social networks is very helpful in mitigating the difficulty of the problem.

## Categories and Subject Descriptors

H.2.8 [**Database management**]: Database Applications - Data mining

## General Terms

Algorithms, Security

## Keywords

social networks, privacy preservation, data publishing, isomorphism, structural attack

## 1. INTRODUCTION

A graph is a powerful modeling tool for representing and understanding objects and their relationships. In recent years, we observe a fast growing popularity in the use of graph data in a wide spectrum of application domains. In particular, we have seen a dramatic increase in the number, in the size, and in the variety of social networks, which can be naturally modeled as graphs. Popular social networks include collaboration networks, online communities, telecommunication networks, and many more. Social networking websites such as Friendster, MySpace, Facebook, Cyworld, and others have become very popular in recent years.

The information in social networks becomes an important data source, and sometimes it is necessary or beneficial to release such data to the public. There can be different kinds of utility for social network data. Kumar, Novak, Tomkins studied the structure and evolution of the network [16]. A considerable amount of research has been conducted on social network analysis [3, 15, 18]. There are also different kinds of querying or knowledge discovery processing on social networks [17, 14, 25, 6, 5]. [10] is a survey on link mining for datasets that resemble social networks, while [21] considers topic and role discovery in social networks.

Many real-world social networks contain sensitive information and serious privacy concerns on graph data have been raised [2, 12, 11]. However, research on preserving privacy in social networks has just begun to receive attention recently. To understand the kinds of attack we need to examine some of the potential application data from social networks, how a social network is translated into a data graph, what kind of sensitive information may be at risk and how an adversary may launch attack on individual privacy.

The first important example of a published social network dataset that has motivated the study of privacy issues is probably the Enron corpus. In 2001, Enron Corporation filed for bankruptcy. With the related legal investigation in the accounting fraud and corruption, the Federal Energy Regulatory Commission has made public a large set of email messages concerning the corporation. This dataset is valuable for researchers interested in how emails are used in an organization and better understanding of organization structure. If we represent each user as a node, and create an edge between two nodes when there exists sufficient email correspondence between the two corresponding individuals, then we arrive at a data graph, or a social network. In [30] another real dataset, consisting of political blogosphere data [1], is considered, this data graph contains over 1000 vertices and 15000 edges. The structure of the data graph will be similar to that of the Enron corpus. From such datasets one can try to understand the nature of privacy attacks.

### 1.1 Adversary knowledge

The first step to anonymization is to know what external information of a graph may be acquired by an adversary. It has been shown that simply hiding the identities of the vertices in a graph cannot stop node re-identification [2]. Recently, several methods [32, 19] have been proposed to achieve *$k$-anonymity* [23, 22, 24] based on various adversary knowledge. For example, the knowledge considered in [32] is a neighborhood of unit distance. The

$d$-neighborhood subgraph of $v$ is defined in [32] as the induced subgraph of $G$ on the set of vertices that are the connected to $v$ via paths with at most $d$ edges.

*Example 1.* In Figure 1(a) assume that the identity of the center of the 7-star in $G$ is X. Then X has 7 1-neighbors in $G$. The 1-neighborhood subgraph of X is shown in Figure 1(b). Since the identities of all vertices in $G$ are hidden, an adversary does not know which vertex in $G$ is X. However, if the adversary knows the 1-neighborhood of X, then the vertex of X in $G$ will be identified. In general, an adversary may have *partial* information about the neighborhood of a vertex, such as a neighborhood of X as shown in Figure 1(c), or Figure 1(d), because these subgraphs may represent some small groups whose information can be gathered by the adversaries. Such information also leads to the re-identification of X. □

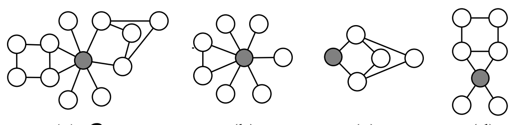

**Figure 1: Neighborhood Subgraphs as NAGs**

We call the subgraph information of an adversary an **NAG** (*Neighborhood Attack Graph*). We do not place any limitation on the NAG, so it can be any subgraph of $G$ up to the entire given graph $G$. Note that in an NAG, one vertex is always marked (shaded in Figures 1(b),(c),(d)) as the vertex under attack.

*Definition* 1 (NAG). *The information possessed by the adversary concerning a target individual $A$ is a pair $(G_a, v)$, where $G_a$ is a connected graph and $v$ is a vertex in $G_a$ that belongs to $A$. We call $(G_a, v)$ the NAG (Neighborhood Attack Graph) targeting at $A$. We also refer to $G_a$ as the NAG.*

Previous works have considered the problem of re-identification attacks, whereby an adversary may use an arbitrary subgraph to locate the vertex in a graph that belongs to an individual. It is pointed out in [2] that most vertices in real social network belong to a small uniquely identifiable subgraph; thus, it is relatively easy for an adversary to acquire subgraph background knowledge associated with a vertex to conduct an attack. In fact, the authors also show that an adversary may even carry out active attacks by maliciously planting some distinct patterns (small subgraphs) in a social network before it is anonymized and published. Thus, another advantage of modeling adversary background knowledge by subgraphs is that we can also handle such active attacks.

## 1.2 Targets of Protection

Privacy preservation is about the protection of sensitive information. From the examples of real datasets we identify two main types of sensitive information that a user may want to keep private and which may be under attack in a social network.

1. NodeInfo:
   The first type, which we call *NodeInfo*, is some information that is attached with a vertex. For example, the emails sent by an individual in the Enron dataset can be highly sensitive since some of the emails have been written only for private recipients and should not be allowed to be linked to any individual. We assume that any identifying information such as names will first be removed from NodeInfo, so that the content of NodeInfo does not help the identification of its owner.

2. LinkInfo:
   The second type, which we call *LinkInfo*, is the information about the relationships among the individuals, which may also be considered sensitive. In this case, the adversary may target at two different individuals in the network and try to find out if they are connected by some path.

We aim to provide sufficient protection for both NodeInfo and LinkInfo. We should point out that the linkage of an individual to a node in the published graph itself does not disclose any sensitive information for the NodeInfo target, because if we separate the publishing of the NodeInfo from that of the node, then attacks of the first type will not be possible.

## 1.3 Related Work

As a pioneer work on privacy in social networks, Backstrom et al [2] discuss both active and passive attacks using small subgraphs. In active attacks, an adversary maliciously plants a subgraph in the network before it is published and uses the knowledge of the planted subgraph to re-identify vertices and edges in the published network. In passive attacks, an adversary simply uses a small uniquely identifiable subgraph to infer the identity of vertices in the published network. However, they do not provide a solution to counter the attacks. Liu and Terzi [19] propose the guarantee of $k$-*anonymity* against adversary knowledge of vertex degrees, so that for every vertex $v$, there are at least $(k-1)$ other vertices that have the same vertex degrees as $v$. Zhou and Pei [32] provide a solution against 1-neighborhood NAGs. Hay et al. [12, 11] propose to protect privacy against subgraph knowledge. A graph is anonymized by random graph perturbing in [12], while [11] proposes to group vertices into partitions and publish a graph where vertices are partitions and edges show the density of edges between the partitions in the original graph. Campan and Truta [4] protect privacy by forming clusters of vertices and collapsing each cluster into a single vertex. [33] anonymizes the data graph by edge and vertex addition so that the resulting graph is $k$-automorphic. They also propose the use of generalized vertex ID's for handling dynamic data releases. All the anonymization methods in [11], [4] and [33] do not impose any restriction on the neighborhood attack graphs (NAGs) as possessed by the adversary.

For LinkInfo attacks, Zheleva and Getoor [31] propose to prevent re-identification of sensitive edges in graph data. Ying and Wu [30] address the problem by edge addition/deletion and switching. They also analyze the effect of their method by studying the spectrum of a graph. However, these works do not have a quantifiable guarantee on their protection.

Most of the above works in privacy preserving publishing of social network aim at the issue of node reidentification, which means that in the published data the adversary is not able to link any individual to a node with high confidence. Most of the solutions target at $k$-anonymity. While this target seems to be reasonable, it does not directly relate to our above targets of protection.

The first shortfall of k-anonymity in addressing our problem comes from the same issue that has been identified by [28] which proposes the technique bucketization. The issue is that if the privacy is some sensitive value linked with some individual, then $k$-anonymity is an overkill. The reason is that if the sensitive values of a set of $k$ records are simply separated from the individual records and published as a set (bucket), then privacy can be guaranteed and there is no need to change the raw data related to the other parts of the records. In the same way, we can publish the original graph structure but publish the sensitive information as a separate dataset. This is a simple solution with no distortion to the graph structure. More details are given in Section 1.4.

The first shortfall may only be a matter of lower utility. The second shortfall is more serious because it is a matter of privacy breach. This problem is similar to that of $k$-anonymity in relational databases, as has been recognized in [20, 27], namely that even with $k$ records with identical quasi-identifiers, so that no individual can be linked to any record with a confidence of more than $1/k$, if all the $k$ records contain the same sensitive value, such as AIDS, then the adversary will be successful in linking an individual to AIDS. The important point is that even if a graph is $k$-anonymous or $k$-automorphic, it cannot protect the data from LinkInfo attack. The fact that there are k different ways to map each of 2 individuals in a graph does not protect the linkage of the 2 individuals if all the mappings are identical in terms of the relevant graph structure. As a simple example, a $k$-clique is a graph that is both $k$-anonymous and $k$-automorphic, however, given a $k$-clique and that 2 individuals $A$ and $B$ are in the graph, one can easily decide that the two individuals are connected by a single edge even though we cannot pinpoint which vertex each individual corresponds to.

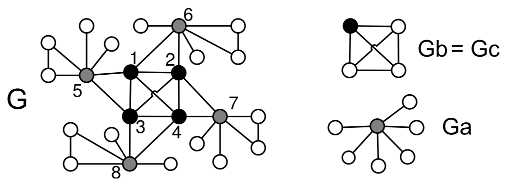

**Figure 2: k-anonymity and NAG attacks**

Figure 2 shows a slightly more complex example of LinkInfo attack on a 4-anonymous graph $G$. In fact $G$ is also 4-automorphic. The structural attacks of the adversary can be based on the NAG's $G_b$ and $G_c$ for two individuals Bob and Carol, respectively, $G_b$ and $G_c$ happen to be identical (the shaded vertex in $G_b(G_c)$ corresponds to Bob(Carol)). In $G$, 4 vertices $\{1, 2, 3, 4\}$ have matching neighborhood subgraphs and any of these can be mapped to Bob or Carol. Although the adversary cannot pin-point the vertex for either target, it is obvious that Bob and Carol must be linked by a single edge. Similarly, if the adversary has an NAG $G_a$ for Alice, and $G_c$ for Carol, the adversary can confirm that there must be a path of length 2 linking Alice and Carol. These examples show that $k$-anonymity and $k$-automorphism do not guarantee security for LinkInfo under NAG attacks.

### 1.4 Challenges and Contributions

We model a social network as a simple graph, in which vertices describe entities (e.g., persons, organizations, etc.) and edges describe the relationships between entities (e.g., friends, colleagues, business partners). A vertex in the graph has an identity (e.g., name, SID) and is also associated with some information such as a set of emails. Our task is to publish the graph in such a way that, given a specific type of *adversary knowledge* in terms of NAGs, the NodeInfo or LinkInfo of any individual can be inferred only with a probability not higher than a pre-defined threshold, whereas the information loss of the published graph with respect to the original graph is kept small.

The first half of the problem on NodeInfo security, on its own, has a simple solution. We can publish the graph structure intact with no distortion on the edges and vertices. First the content of NodeInfo for each $v$ is screened to remove the occurrences of names or other user identifying information. The processed NodeInfo of each $v$, $I(v)$, is detached from vertex $v$. We randomly partition the vertex set $V$ into groups of at least size $k$. For each group, the corresponding set of NodeInfo is published as a group. For example, if $v_1, ..., v_k$ form a group, then NodeInfo $\{I(v_1), ..., I(v_k)\}$ will be published as a group, breaking the linkage of the NodeInfo to each individual. This is illustrated in Figure 3. A similar technique has been proposed in [7] for the anonymization of bipartite graphs.

The difficulty of our problem therefore lies in the second half of the problem, where the linkage of two individuals may be under attack. Another major source of difficulty comes from the complexity of graph datasets. Not only is the anonymization problem NP-hard, but in order to tackle many of the sub-problems, subgraph isomorphism testing, an NP-complete problem by itself, is often needed many times. In general, mechanisms for graph problems tend to be highly complex.

| Vertex group | NodeInfo group |
|---|---|
| $\{a_1, a_2, a_3, ..., a_k\}$ | $\{I(a_1), I(a_2), I(a_3), ..., I(a_k)\}$ |
| $\{b_1, b_2, b_3, ..., b_k\}$ | $\{I(b_2), I(b, 2), I(b_3), ..., I(b_k)\}$ |
| ... | ... |

**Figure 3: NodeInfo Table published along with $G_k = G$**

Our contributions can be summarized as follows. (1) We identify two realistic targets of privacy attacks on social network publication, NodeInfo and LinkInfo. We point out that the popular notion of $k$-anonymity in graph data does not protect the data against LinkInfo attacks. Although some previous works have considered protection of links, there has not been any definition of a quantifiable guarantee in the protection. To our knowledge we are the first to define this problem formally. (2) We prove that this problem is NP-hard. (3) We propose a solution by $k$-isomorphism (see Section 3) anonymization and show that this is the only solution to the problem. (4) We design a number of techniques to make the anonymization process efficient while maintaining the utility. (5) We introduce a dynamic release mechanism that has a number of advantages over previous work. (6) Our empirical studies show that our method performs well in the anonymization of real datasets.

The rest of the paper is organized as follows. Section 2 gives the problem definition. Section 3 describes the main solution. Section 4 presents the anonymization algorithm. Section 5 reports the experimental results. Section 6 concludes the paper.

## 2. PRELIMINARIES AND PROBLEM DEFINITION

We model a social network as a simple graph, $G = (V, E)$, where $V$ is the set of vertices, $E$ is the set of edges. We assume that each vertex in the social network has a unique identity for an individual, which has been hidden in $G$. In addition, we assume that each vertex $v$ in $V$ is associated with some node information $I(v)$. We also use $V(G)$ and $E(G)$ to refer to the vertex set and edge set of $G$.

A graph $G' = (V', E')$ is a subgraph of a graph $G = (V, E)$, denoted by $G' \subseteq G$, if $V' \subset V$ and $(u, v) \in E'$ only if $(u, v) \in E$. We also say that $G$ is a supergraph of $G'$, denoted by $G \supseteq G'$.

*Definition* 2 (GRAPH ISOMORPHISM). *Let $G = (V, E)$ and $G' = (V', E')$ be two graphs where $|V| = |V'|$. $G$ and $G'$ are isomorphic if there exists a bijection $h$ between $V$ and $V'$, $h$: $V(G) \to V(G')$, such that $(u, v) \in E$ if and only if $(h(u), h(v)) \in E(G')$. We say that an isomorphism exists from $G$ to $G'$, and $G = G'$. We also say that edge $(u, v)$ is isomorphic to $(h(u), h(v))$.*

*Definition* 3 (SUBGRAPH ISOMORPHISM). *Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. There exists a subgraph isomorphism from $G$ to $G'$ if $G$ contains a subgraph that is isomorphic to $G'$.*

We call $G$ a *proper subgraph* of $G'$, denoted as $G \subset G'$, if $G \subseteq G'$ and $G \not\supseteq G'$. $G$ is *isomorphic* to $G'$, or $G = G'$, if $G \subseteq G'$ and $G' \subseteq G$.

The decision problem for graph isomorphism is one of a small number of interesting problems in NP which have neither been proven to be polynomial time or NP-hard. The decision problem for subgraph isomorphism is NP-Complete [9].

Adversary background knowledge refers to the information that an adversary may acquire and use to infer the identify of some vertex in a published social network. As we motivate in Section 1.1, in this paper we study the privacy preserving publishing problem with the adversary background knowledge being some subgraph that contains the vertex under attack, defined as **NAG** (*Neighborhood Attack Graph*) in Definition 1. As in the recent works of [11], [4] and [33], we do not place any limitation on the NAG, so it can be up to the entire given graph.

With the above understanding of the adversary knowledge, our problem definition is based on the notion of $k$-security.

*Definition* 4 (K-SECURITY). *Let $G = (V, E)$ be a given graph with unique node information $I(v)$ for each node(vertex) $v \in V$. Each vertex $v \in V$ is linked to a unique individual $U(v)$. Let $G_k$ be the anonymized graph of $G$. $G_k$ satisfies* $k$-security*, or $G_k$ is* $k$-secure*, with respect to $G$ if for any two target individuals $A$ and $B$ with corresponding NAGs $G_A$ and $G_B$ that are known by the adversary, the following two conditions hold*

1. *(NodeInfo Security) the adversary cannot determine from $G_k$ and $G_A(G_B)$ that A(B) is linked to $I(v)$ for any vertex v with a probability of more than $1/k$*
2. *(LinkInfo Security) the adversary cannot determine from $G_k$, $G_A$ and $G_B$ that A and B are linked by a path of a certain length with a probability of more than $1/k$.*

While $k$-security is our main objective, there is also another important objective, which is the data utility. We would like the published graph to keep the main characteristics of the original graph in order that it may be useful for data analysis. Therefore we must also consider the anonymization cost, which is a measurement of the information loss due to the anonymization. In our proposed method, anonymization may involve edge additions and deletions. One possible measure for the anonymization cost is the edit distance between $G$ and $G_k$, which is the total number of edge additions and deletions.

*Definition* 5 (EDIT DISTANCE). *The edit distance between $G$ and $G_k$ is given by $ED(G, G_k) = |(E(G) \cup E(G_k)| - |E(G) \cap E(G_k)|$*

However, edit distance is not a sound measure when both edge additions and deletions are allowed. The anonymization aims to make nodes indistinguishable as far as the neighborhood is concerned. A basic step in this process is to make sure that for k pairs of vertices, either each is linked by an edge or none is linked. If the graph is sparse, it is likely that most of the pairs will not be linked in the original graph, meaning that if edit distance is used as the utility measure, anonymization will tend to delete edges. Such tendency will result in poor utility in the published graph. Therefore we follow a principle of previous works such as [12, 30] which add and delete similar amounts of edges to maintain about the same number of edges before and after anonymization. In this way, edge deletions will not become more favorable than edge additions. To this end, our minimal anonymization cost is given by two conditions: (1) the difference between the number of edges in $G$ and the number of edges in $G_k$ is minimized. (2) Under condition (1), the edit distance $ED(G, G_k)$ is minimized.

*Definition* 6 (ANONYMIZATION COST). *An anonymization from $G$ to $G_k$ has minimal anonymization cost if*

**(1)** $||E(G_k)| - |E(G)||$ *is minimized ;*

**(2)** *under condition (1), $ED(G, G_k)$ is minimized*

*Definition* 7 (PROBLEM DEFINITION). *The problem of privacy preservation in graph publication by $k$-security is defined as follows:* given a network graph $G = (V, E)$ with unique $I(v)$ for each $v \in V$, and a positive integer $k$, derive an anonymized graph $G_k = (V_k, E_k)$ to be published, such that (1) $V_k = V$ ; (2) $G_k$ is $k$-secure with respect to $G$; and (3) the anonymization from $G$ to $G_k$ has minimal anonymization cost. *We call this problem $k$-Secure-PPNP (or $k$-Secure Privacy Preserving Network Publication).*

THEOREM 1 (NP-HARDNESS). *The problem of $k$-Secure Privacy Preserving Network Publication is NP-hard.*

PROOF. The proof is by reducing the NP-complete problem of PARTITION INTO TRIANGLES. The details are given in the Appendix. □

COROLLARY 1. *The NP-Hardness for $K$-Secure-PPNP remains to hold if the minimal anonymization cost requirement is replaced by minimum edit distance $ED(G, G_k)$ in the problem definition.*

PROOF. Prove by simply removing the condition of $||E(G_k)| - |E(G)||$ in the proof for Theorem 1 in the Appendix. □

Though the problem is NP-hard, typically it is not possible to relax the privacy requirement. In the next section, we derive a necessary and sufficient solution for $k$-security when we aim at keeping the partitioning of the given graph to a minimum.

## 3. K-ISOMORPHISM

In this section, we propose a framework solution for the problem of privacy preservation in a graph for $k$-security. For simplicity we first assume that for the given graph $G = \{V, E\}$, $|V|$ is a multiple of $k$. This assumption can be easily waived by adding no more than $k - 1$ dummy vertices in the graph. The solution relies on the concept of graph isomorphism.

*Definition* 8 (K-ISOMORPHISM). *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 $i \neq j$.*

The solution is as follows. Given a graph $G = \{V, E\}$. Derive a graph $G_k = \{V_k, E_k\}$ such that $V_k = V$, and $G_k$ is $k$-isomorphic, that is $G_k = \{g_1, ..., g_k\}$ with pairwise isomorphic $g_i$ and $g_j$, $i \neq j$. $G_k$ is the published graph. For each $v \in V$, NodeInfo $I(v)$ is attached to $v$ in the published graph.

THEOREM 2 (SOUNDNESS). *A k-isomorphic graph $G_k = \{g_1, ..., g_k\}$ is k-secure.*

PROOF. Since the graphs $g_1, ..., g_k$ are pairwise isomorphic, for any NAG of an adversary for a target individual $Alice$, whenever the NAG is contained in any $g_i$, there are at least $k$ different vertices $v_1, ..., v_k$ that can be mapped to $Alice$ and they are not distinguishable. Hence NodeInfo security is guaranteed.

If the adversary aims to attack the linkage of 2 individuals $Alice$ and $Bob$, in the worst case, the adversary can find matching vertices for both $Alice$ and $Bob$ in one of the subgraphs $g_i$. However, by $k$-isomorphism, the same is true for each subgraph. There are $k$ different vertices $a_1, ...., a_k$ that can be mapped to $Alice$, and $k$ different vertices $b_1, ..., b_k$ that can be mapped to $Bob$, where $a_i \in g_i$ and $b_i \in g_i$, for $1 \le i \le k$.

If $a_1$ and $b_1$ are linked by a path of length $p$ in $g_1$, $a_i$ and $b_i$ are linked by a similar path in $g_i$, for all $i$. For $Alice$ to be the owner of $a_1$ and $Bob$ to be the owner of $b_1$, the probability is $\frac{1}{k} \times \frac{1}{k}$. The probability that $Alice$ is linked to $Bob$ by a path of length $p$ is hence the probability that their vertices are in the same $g_i$, and it is given by $k \times \frac{1}{k} \times \frac{1}{k} = \frac{1}{k}$. Therefore the condition for LinkInfo security holds and $G_k$ is $k$-secure. □

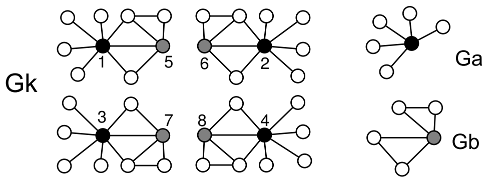

**Figure 4: k-isomorphism and k-security**

*Example 2.* An example is shown in Figure 4. Here the adversary attacks with two NAGs $G_a$ and $G_b$ for two target individuals, $Alice$ and $Bob$, respectively. Four vertices $\{1, 2, 3, 4\}$ are found to be linkable to $Alice$, while $\{5, 6, 7, 8\}$ are linkable to $Bob$. The adversary can only determine that $Alice$ and $Bob$ are linked by an edge with a probability of $1/4$. □

Given Theorem 2, the following theorem says that if the partitioning of the graph $G$ in the anonymization is limited, then partitioning into $k$ isomorphic subgraphs is the only solution for $k$-security.

THEOREM 3 (NECESSITY). *Let a published graph $G_k$ be $k$-secure, if $G_k$ is made up of no more than $k$ disjoint connected subgraphs, then $G_k$ is $k$-isomorphic.*

PROOF. For the proof we shall need the notion of graph automorphism: Given a graph $G = (V, E)$, an automorphism is a function $f$ from $V$ to $V$, such that $(u, v) \in E$ iff $(f(u), f(v)) \in E$. That is $f$ is a graph isomorphism from $G$ to itself.

Suppose the graph $G_k$ is a collection of no more than $k$ disjoint subgraphs, let there be $l$ disjoint subgraphs, $l \le k$. Let $g_L$ be a biggest disjoint subgraph with a maximum number of vertices. In the worst case the adversary may identify 2 targets Alice and Bob by means of two NAGs $(G_A, v_A)$ and $(G_B, v_B)$, where $G_A = G_B = g_L$. Then it is possible that the vertices for Alice and Bob are in $g_L$ and therefore they are linked by some path of length $p$ in $g_L$. If there exist one or more automorphisms in $g_L$, then there will be more than one vertex that can be mapped to Alice and Bob. However, with each automorphism, Alice is linked to Bob via a path of the same length $p$. Hence with or without any automorphism in $g_L$, if $g_L$ is the only disjoint connected subgraph in $G_k$ that is isomorphic to $G_A$(=$G_B$), then the adversary can confirm the linkage between Alice and Bob.

From the above, there must be two or more disjoint subgraphs that are isomorphic to $G_A(G_B)$. Suppose there are $m$ such subgraphs, each containing a maximum number of vertices. The adversary can confirm that these are the only disjoint subgraphs where Alice and Bob can be mapped to. The probability that they are both in one of the $m$ biggest subgraphs is given by $\frac{1}{m} \times \frac{1}{m}$. The probability that Alice and Bob are linked by a path with length $p$ is given by $(m) \times \frac{1}{m} \times \frac{1}{m} = \frac{1}{m}$. In order for the probability to be bounded by $1/k$, $m \ge k$. Since we are allowed at most $k$ partitioned subgraphs, $m = k$. Hence there are exactly $k$ disjoint connected subgraphs that are isomorphic to each other. □

COROLLARY 2. *If $G_k$ is $k$-secure and is made up of more than $k$ disjoint connected subgraphs, let $g_L$ be any disjoint connected subgraph in $G_k$ with the greatest number of vertices, then $G_k$ must contain at least $k - 1$ other disjoint connected subgraphs isomorphic to $g_L$.*

PROOF. The corollary follows readily from arguments similar to that in the proof of Theorem 3. □

Enforcing $k$-isomorphism could mean that we have reduced the information of the given graph $G$ to $1/k$ the original size. In fact, since all the graphs $g_i$ are isomorphic we may as well publish just one of the subgraphs $g_i$ in $G_k$, if graph structure is the only interested information. However, if there is individual information $I(v)$ attached to each vertex $v$, then the subgraphs are not totally the same. In considering the utility of $G_k$, it helps to compare with conventional data analysis. Many real life applications give rise to very large graphs in terms of thousands or even millions of vertices. With such a large population, analysis will typically be based on a statistical study by sampling. It is noted that each of the $k$ isomorphic subgraphs in $G_k$ can be seen as a sample of the population and the sample size is very large, consisting of $1/k$ of the population, where $k$ is very small compared with the graph size. In fact, the anonymization effort can introduce normalization to the data to avoid inaccuracies due to overfitting. Hence there is good reason to believe that the anonymized graph maintains good utilities.

## 4. ALGORITHM

Our solution as presented in the previous section involves the generation and publishing of a graph $G_k$ that consists of $k$ isomorphic subgraphs, let us call these subgraphs **i-graphs**. Here we consider how to arrive at the i-graphs from the given graph $G$. We would preserve the set of vertices by partitioning the graph of $G$ into $k$ subgraphs with the same number of vertices. Figure 5 shows an example where $k$ is 4, so that the given graph $G$ is partitioned into 4 subgraphs $g_1, g_2, g_3, g_4$.

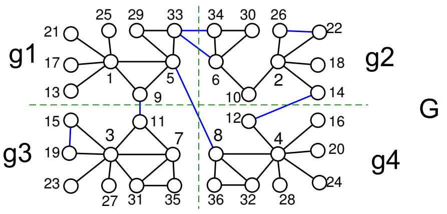

**Figure 5: Given graph G and partitioning**

After the partitioning, the subgraphs are augmented by edge addition and deletion to ensure pairwise graph isomorphism. In Figure 5, edges are added or deleted so that we obtain the graph $G_k$ as shown in Figure 4. Let the isomorphic function from $g_i$ to $g_j$ be $h_{ij}$. In this example, $h_{12}(1) = 2$, and $h_{21}(2) = 1$; $h_{34}(7) = 8$, $h_{24}(22) = 24$ ...

Obviously the quality of the anonymization depends heavily on the graph partitioning. We introduce a simple baseline algorithm that helps us form the $k$ partitions. This will be refined later for better anonymization quality. First we need some definitions.

*Definition* 9 (EMBEDDINGS AND FREQUENCY). *Given two graphs $g$ and $G$, the set of embeddings of $g$ in $G$, denoted by* *embed$(g, G)$, is defined as embed$(g, G) = \{g' : g' \subseteq G, g' = g\}$. The* frequency *of $g$ in $G$, denoted by freq$(g, G)$, is defined as freq$(g, G) = |embed(g, G)|$. For simplicity, we use embed$(g)$ and freq$(g)$ when $G$ is clear in the context.*

*Definition* 10 (VERTEX-DISJOINT EMBEDDINGS). *Let $g$ be a connected subgraph in a graph G. A set of embeddings $b_1, ..., b_k$ of $g$ are vertex-disjoint embeddings if $\forall b_i, b_j, V(b_i) \cap V(b_j) = \emptyset$. We use "VD-embedding" as a shorthand for* "vertex-disjoint embedding*".*

*Example 3.* Figure 9 shows a graph $G$ and a subgraph $g$, there are 4 embeddings of $g$ in $G$, $embed(g, G) = \{g_1, g_2, g_3, g_4\}$. The frequency of $g$ in $G$ is given by $freq(g, G) = 4$. The embeddings are overlapping, so the maximum set of VD-embeddings for $g$ has a size of 3 only, namely, $\{g_1, g_4, g_3\}$. □

**Algorithm 1 Baseline Graph Synthesis**

```
Input: A graph $G$ and an integer $k$.
Output: An anonymized graph, $G_k = \{g_1, ..., g_k\}$, of $G$.
VM: Vertex Mapping for $g_1, ..., g_k$.
1. $\forall i, 1 \le i \le k : g_i \leftarrow \phi$; VM $\leftarrow \phi$;
2. while $G$ is not empty
3.     select a graph $g$ with $k$ VD-embeddings $b_1, ..., b_k$ in $G$;
4.     for each embedding $b_i$ due to Line 3
5.         remove $b_i$ from $G$;
6.         insert $b_i$ into $g_i$;
7.     append the new vertex mappings to VM;
8. add/delete edges in each $g_i$ for pairwise $k$-isomorphism;
9. return $G_k$;
```

Algorithm 1 also creates a Vertex Mapping VM, which will be used in the final step of edge addition and deletion as discussed in Section 4.1.3. VM is a table with $k$ columns, $c_1, ..., c_k$, with $c_i$ for subgraph $g_i$, where VM$[c, r]$ is the table entry at column $c$ and row $r$. Each tuple in the table corresponds to one possible vertex mapping so that the value for $h_{ij}(\text{VM}[i, r]) = \text{VM}[j, r]$ for all $1 \le i, j \le k$, and $i \ne j$. The vertex mapping VM for the example in Figures 4 and 5 is shown in Figure 6. Here vertex $5 = \text{VM}[1, 2]$, vertex $7 = \text{VM}[3, 2]$, $h_{13}(5) = 7$.

| $g_1$ | $g_2$ | $g_3$ | $g_4$ |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| ... | ... | ... | ... |
| 33 | 34 | 35 | 36 |

**Figure 6: Vertex Mapping VM for** $G_k = \{g_1, g_2, g_3, g_k\}$

Note that there always exist choices for the selection of graph $g$ at Line 3 of Algorithm 1 so that the Baseline graph synthesis algorithm terminates with an anonymized graph that is $k$-isomorphic. This is because a trivial selection is a graph with a single node. However, this will result in poor utility for $G_k$. Therefore in Section 4.1 we shall refine the algorithm and introduce better mechanisms for selecting such subgraphs.

## 4.1 Refined Algorithm

While the Baseline Algorithm 1 in the above is a sound solution, there is no specific guideline on how to select the graph $g$ in Line 3 for the graph synthesis. In order to find good candidates to be inserted into the $k$ i-graphs, here we propose to consider frequent subgraphs that are large. Considering frequent subgraphs has been shown to be a sound strategy in related works such as [32] and [33]. In our case, frequent subgraphs have a high potential to generate VD-embeddings and large connected subgraphs minimize the edge augmentation needed for graph isomorphism.

However, the discovery of frequent subgraphs is costly, especially when considering large subgraphs. Also considering large subgraphs may not be useful since they are less likely to be frequent. For better performance we set a threshold $maxPAGsize$ on the size of the maximum subgraphs to be considered, where the size is in terms of the number of edges in the subgraph.

*Definition* 11 (PAG). *Given a graph G, and a size threshold $maxPAGsize$, any connected subgraph $g$ of G, with $|E(g)| \le maxPAGSize$, is a Potential Anonymization SubGraph, or PAG.*

Our empirical studies show that the average degree $d$ of $G$ is a good value for $maxPAGsize$. The intuitive explanation for this phenomenon is that many vertices in G have this degree $d$ and each forms a PAG with their $d$ 1-neighbors. Using such a threshold would be a good basis for locating frequent subgraphs. Algorithm 2 outlines the refined algorithm.

At Line 2 of Algorithm 2, we compute a set of PAGs, $\mathcal{M}$, which is by traversing the given $G$ from each vertex in a depth-first manner and enumerating all connected subgraphs of size *maxPAGsize*. If the vertex set of these PAGs cannot cover all vertices in $G$, then we also enumerate some PAGs of sizes less than *maxPAGsize*, which are in fact those isolated connected components in $G$ with less than *maxPAGsize* edges. The graph traversal also determines the embeddings for each PAG.

For PAGs with at least $k$ VD-embeddings, we can extract the vertices of $k$ such embeddings to be transferred from $G$ to the $g_i$'s in $G_k$ (Lines 3-7). These embeddings ensure that very few or no edge augmentation will be necessary for the anonymization with respect to the embeddings. Since removing some embeddings from $G$ may affect the formation of VD-embeddings for the remaining PAGs in $G$, we select PAGs in a greedy manner. PAGs of bigger sizes will be considered earlier. Also we give priority to VD-embeddings that contain vertices with the highest degrees. Such embeddings may incur greater distortion and there is a better chance to reduce overall distortion if treated earlier.

We need to anonymize a PAG $g \in \mathcal{M}$ if $g$ has less than $k$ VD-embeddings. By anonymizing $g$, we refer to the process by which $k$ VD-embeddings of $g$ are formed so that they can be removed from $G$, their vertex mapping is entered into VM, and the corresponding vertices inserted into the i-graphs. We discuss how to anonymize $g$ (Line 10) in Section 4.1.1. When graph $G$ becomes empty, all vertices have been transferred to $G_k$. The final step is to add edges to $G_k$ at Line 11, which will be discussed in Section 4.1.3.

### *4.1.1 To anonymize a PAG*

In this subsection, we discuss the step at Line 10 of Algorithm 2, that is, how to anonymize a PAG $g$. There are insufficient VD-embeddings for $g$, to create more VD-embeddings for $g$, we can find certain subgraphs of $g$ and add edges linking new vertices to the subgraphs. To reduce the information loss, we want the selected subgraphs to be as large as possible so that less graph augmentation is needed. We describe a simple algorithm that anonymizes $g$ as follows.

We enumerate $g$'s size-$i$ subgraphs one by one by decrementing $i$ from $(|g|-1)$ to 0 (size-0 subgraphs are single-vertex subgraphs). For each $g' \subset g$ enumerated, we search for all embeddings of $g'$ in $G_k$. For each embedding $g'_{emb}$ of $g'$, if it is vertex-disjoint with all the current VD-embeddings of $g$, we add edges and vertices to

**Algorithm 2 Frequency Based Graph Synthesis**

```
Input: A graph $G$ and an integer $k$.
Output: An anonymized graph, $G_k = \{g_1, ..., g_k\}$, of $G$.
VM: Vertex Mapping for $g_1, ..., g_k$.
1. $\forall i, 1 \leq i \leq k : g_i \leftarrow \phi$; VM $\leftarrow \phi$;
2. $\mathcal{M} \leftarrow$ a set of PAGs that covers $G$;
   determine $embed(g, G)$ for each $g \in \mathcal{M}$;
3. while there exists $g \in \mathcal{M}$ with at least $k$ VD-embeddings in $G$
4.     select $k$ VD-embeddings $b_1, ..., b_i$ of $g$;
5.     for each $b_i$
6.         remove $b_i$ from $G$; add $V(b_i)$ to $g_i$;
7.         update $\mathcal{M}$; update VM;
8. while $G$ is not empty
9.     select $g \in \mathcal{M}$;
10.    anonymize $g$;   /* Details in Section 4.1.1 */
   /* At this point, $E(g_i) = \phi$ for each $g_i$ */
11. add edges in each $g_i$ for pairwise $k$-isomorphism;
12. return $G_k$;
```

$g'_{emb}$ to make it into a VD-embedding of $g$. The process continues until the number of VD-embeddings of $g$ in $G_k$ reaches $k$.

The algorithm described above, however, suffers deficiency in the following two aspects. First, it requires the searching of all embeddings of the subgraphs of $g$ in $G_k$. This process is expensive because it may involve a huge number of subgraph isomorphism tests. Second, most PAGs actually share with each other a large number of common subgraphs (as also pointed out in maximal frequent subgraph mining [13]). Thus, many subgraphs may be repeatedly processed and much computing resource is wasted.

To address the first deficiency, we take advantage of the fact that the embeddings of the PAGs have already been located in $G$. Thus, we can locate the embeddings of the subgraphs within each embedding of the PAGs, which is significantly faster than searching in the big graph $G$.

To address the second deficiency, we use a hashtable, $\mathcal{H}$, to keep every subgraph $g'$ that has been processed, along with the embeddings $embed(g')$ that have been uncovered. Later when we anonymize another PAG $g$ and enumerate a subgraph $g'$, we can first check if $g'$ is used. If $g'$ is found in $\mathcal{H}$, then we can readily use the embeddings of $g'$ to anonymize $g$. The hash function for $\mathcal{H}$ is based on graph properties such as degree distribution.

The algorithm of anonymizing a PAG $g$ is outlined in Algorithm 3. First, we extract a set of VD-embeddings for $g$ (Line 1). Then, we enumerate $g$'s subgraphs, from the largest to the smallest (Line 2), but terminate whenever we have $k$ VD-embeddings of $g$ (Line 14-17). For each $g' \subset g$ enumerated, we first find all the embeddings of $g'$, $embed(g')$, from the embeddings of the PAGs that are supergraphs of $g'$ (Lines 4-6). Note that $\mathcal{M}$ is the set of PAGs computed in Algorithm 2. We keep $embed(g')$ in a hashtable $\mathcal{H}$. Thus, if a subgraph $g'$ is in $\mathcal{H}$ already, we obtain $embed(g')$ directly from $\mathcal{H}$. Then, Lines 10-13 find an embedding $g'_{embed}$ of $g'$ which is vertex-disjoint with all VD-embeddings of $g$ in $D$. We create a new VD-embedding of $g$ from $g'_{embed}$ by adding vertex-disjoint edge(s) and node(s).

### 4.1.2 An Example

The following example illustrates how we partition a graph $G$ for $k$-security.

*Example 4.* Figure 7 shows a graph $G$ and its anonymized graph $G_k$. For clarity in the explanation of our algorithms we choose a $G$ that consists of multiple components $G$ = { (a), (b), (c), (d), (e), (f), (r), (s) }. We use a $maxPAGsize$ of 5 for illustration. The value of $k$ for $k$-security is 2.

**Algorithm 3 Anonymize-PAG**

```
Input: A PAG $g$ to be anonymized, $G, G_k$, VM, $\mathcal{M}$.
Output: Updated $G_k = \{g_1, ..., g_k\}, G, \mathcal{M}$, VM.
$\mathcal{H}$: global hashtable to store processed PAGs and embeddings.
1. $\mathcal{D} \leftarrow$ set of VD-embeddings of $g$ in current $G$ ;
2. for each $g' \subset g$ in size-descending order
3.     if $g'$ is not in $\mathcal{H}$
4.         $embed(g') \leftarrow \emptyset$;
5.         for each $g'' \in \mathcal{M}$, where $g'' \supset g'$ and $g'' \neq g$
6.             search in the embeddings of $g''$ for the embeddings
               of $g'$, and add these embeddings of $g'$ to $embed(g')$;
7.         insert $g'$ and $embed(g')$ into $\mathcal{H}$;
8.     else
9.         obtain $embed(g')$ from $\mathcal{H}$;
10.    for each $g'_{embed} \in embed(g')$
11.        if $g'_{embed}$ is vertex-disjoint from all graphs in $\mathcal{D}$
12.            create a new VD-embedding $b$ of $g$ from $g'_{embed}$;
13.            delete $b$ from $G$; insert $b$ into $\mathcal{D}$;
14.            if $|\mathcal{D}| = k$
15.                insert the vertices of the $k$ graphs in $\mathcal{D}$ into $g_1, ..., g_k$;
16                 update VM; update $\mathcal{M}$;
17.                return
```

In the first step of Algorithm 2 we compute the set of PAGs in $G$. The PAGs with 5 edges will be { (a), (c), (d), (f), (r) }. However these PAGs do not cover the entire graph $G$, in particular, the component (s) is not covered. Hence we add (s) to the set of PAGs. We also determine the embeddings of the PAGs, the results are shown in Figure 8. In this table, we use (e)x4 to denote the 4 embeddings of graph (a) in (e). For each PAG $P$, we also measure the highest degree in each embedding in $embed(P)$, and count the number of graphs in $embed(P)$ with this degree. Each PAG $P$ and $embed(P)$ are entered into the hashtable $\mathcal{H}$. The VD-embeddings are also determined. Among the PAGs, (a), (f) and (s) have sufficient number of VD-embeddings (|VD-embed|).

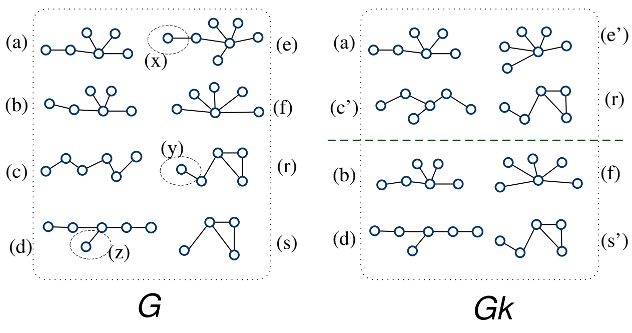

**Figure 7: Graph $G$ and Anonymized Graph $G_k$**

| PAG | embed() | #embed | \|VD-embed\| | MaxDeg (count) |
|---|---|---|---|---|
| (a) | (a),(b), (e)x4 | 6 | 3 | 5 (1) |
| (c) | (c) | 1 | 1 | 2 (1) |
| (d) | (d) | 1 | 1 | 3 (1) |
| (f) | (e), (f) | 2 | 2 | 5 (2) |
| (r) | (r) | 1 | 1 | 3 (1) |
| (s) | (r), (s) | 2 | 2 | 3 (2) |

**Figure 8: Properties of PAGs in Figure 7**

According to Lines 3 to 7 in Algorithm 2, we select the PAGs (a), (f) and (s) for anonymization. (f) will be processed first since it has more VD-embeddings with a maximum degree of 5 (MaxDeg count of 2). There are exactly 2 VD-embeddings for (f), namely,

**Algorithm 4 i-Graph Formation**

```
Input: $G = (V, E)$ $(E = \{e_1, ..., e_{|E|}\})$, VM.
Output: $G_k = \{g_1, ..., g_k\}$.
$\mathcal{CE}$ stores the number of edges in $E$ crossing 2 i-graphs in $G_k$.
 1. $V(G_k) \leftarrow V$; $E(G_k) \leftarrow \emptyset$; $\mathcal{CE} \leftarrow 0$;
        $\forall i, 1 \le i \le |E|$: $Add[i].cnt \leftarrow k$; $Add[i]$.VM $\leftarrow \emptyset$;
              $Processed[i] \leftarrow False, Marked[i] \leftarrow False$;
 2. for each edge $e_j = \{v_A, v_B\} \in E$
 3.     if not $Processed[j]$
 4.         if $v_A$ and $v_B$ appear in different columns in VM
 5.             $\mathcal{CE} \leftarrow \mathcal{CE} + 1$; /* increment number of cross edges */
 6.         else if $v_A =$ VM$[c, a]$ and $v_B =$ VM$[c, b]$
 7.             $Marked[j] \leftarrow True$; $Processed[j] \leftarrow True$;
 8.             $Add[j]$.VM $\leftarrow \{a, b\}$;
 9.             for each $e' = \{$VM$[i, a]$, VM$[i, b]\}$ /* isomorphic edges */
10.                 if $e' = e_r \in E$
11.                     $Add[j].cnt \leftarrow Add[j].cnt - 1$;
12.                     $Processed[r] \leftarrow True$;
13. retain only entries $Add[i]$ where $Marked[i] = True$;
        /* Let there be $n$ retained entries in $Add[]$ */
14. sort the retained entries $Add[]$ by $Add[].cnt$ in increasing order;
15. determine cut point $x$ in the sorted $Add[]$ to minimize
        $|\sum_{1 \le i \le x} Add[i].cnt - (\sum_{x < i \le n}(k - Add[i].cnt) + \mathcal{CE})|$
16. for each $1 \le i \le x$
17.     add all isomorphic edges determined by $Add[i]$.VM to $G_k$;
18. return $G_k$;
```

(e)$-$(x) and (f), they are removed from $G$ and their vertex sets are added to subgraphs $g_1$ and $g_2$ of $G_k$, respectively. $G$ becomes { (a), (b), (c), (d), (r), (s), (x) }. (x) is added to the PAG set $\mathcal{M}$. Next we process PAG (a), and remove (a) and (b) from $G$ while adding vertex set $V$(a) to $g_1$ and $V$(b) to $g_2$. For PAG (s), (r)-(y) and (s) are removed from $G$ and their vertices entered into $g_1$ and $g_2$ respectively. $G$ becomes { (c), (d), (x), (y) }.

Since (d) has a higher degree vertex, it is selected as the next $g$ in Line 9 of Algorithm 2. By Line 2 of Algorithm 3, we select some subgraph of (d) and look for its embeddings. Suppose the subgraph of (d)$-$(z) is selected as $g'$. It is not in hashtable $\mathcal{H}$. From Line 4-7 in Algorithm 3, we find that $g'$ is a subgraph of (c) ((c) acts as $g''$ here). Hence we get the embedding (c) and executing Lines 11-12, augment it to (c') to form a VD-embedding for (d). Vertex sets $V$(c') and $V$(d) are added to $g_1$ and $g_2$ respectively, while (c) and (d) are removed from $G$.

Only (x) and (y) remain in $G$ and (x) remains in $\mathcal{M}$, with VD-embeddings of (x) and (y). They are removed from $G$ to $g_1$ and $g_2$, at which point $G$ becomes empty and the partitioning of the vertex set is complete. With Line 11, we add edges to $g_1$ and $g_2$, resulting in the graph $G_k$ as shown in Figure 7. □

### 4.1.3 *From Vertex Partitions to i-Graphs*

After the vertices of $G$ are partitioned so that the vertices of the i-graphs in $G_k$ are decided, and VM is formed, we consider the final step of addition of edges to $G_k$ at Line 11 of Algorithm 2. This step is shown in Algorithm 4, here the input graph $G$ is the original graph. Intuitively, for each set of $k$ potential isomorphic edges $\{$VM$[i, r_1]$, VM$[i, r_2]\}$ for $1 \le i \le k$ and any pair of $r_1, r_2$, we have the choice of adding all of them to the $g_i$'s or making sure they do not appear in all $g_i$'s. It reduces the edit distance to choose addition when there are fewer missing edges in $G$ than existing edges, and deletion (not adding) otherwise. However, we must also keep the number of additions and deletions similar, so it is necessary to find a balancing point.

With Algorithm 4, for each pair of rows $a$ and $b$ in VM, if ( VM$[i, a]$, VM$[i, b]$ ) corresponds to some edge $e$ in $E$, then the entry of $Add[j]$ for either $e$ or exactly one of the edges in $E$ isomorphic to $e$ will be filled so that $Add[j].vm = \{a, b\}$ and $Add[j].cnt$ is $k$ minus the number of edges in $E$ isomorphic to $e$, also $Marked[j]$ is set to True. $Processed[]$ helps to avoid processing an edge which has been considered during the processing of some other edge. After the sorting at Line 14, the $Add[e]$ entries are in increasing order of $Add[e].cnt$, which is the number of edges to be added if all edges isomorphic to $e$ should exist in $G_k$. The cut point $x$ determines a point in the sorted list where all entries above the point correspond to edge addition, and those below will involve edge deletion.

LEMMA 1. *In the anonymization of graph $G = (V, E)$, given a vertex mapping VM for the graphs $G_k = g_1, ..., g_k$ in Algorithm 4. Let $\mathcal{C} = max\{0, \mathcal{CE} - \sum_{1<i<n} Add(i).cnt\}$ at Line 15. A graph $G_k = (V, E_k)$ conforming to VM with a minimum edit distance $ED(G_k, G)$ under the condition of $||E_k| - |E|| \le max\{k, \mathcal{C}\}$ can be determined in $\mathcal{O}(k|E|)$ time.*

PROOF. Algorithm 4 generates a $G_k$ that conforms to VM. At Line 15 of the algorithm, if the number of cross edges, $\mathcal{CE}$, is greater than the sum of added edges in $G_k$, then only edge additions in the $g_i$ are used for attaining graph isomorphism, then $||E_k| - |E|| = \mathcal{C}$. Otherwise, $||E_k| - |E||$ must be at most $k$, a greater value is not possible since the algorithm could have shifted the cut point $c$ up or down the array $Add[]$ to obtain a smaller $||E_k| - |E||$. Note that the sorting at Line 11 can be performed by scanning $Add[]$ once, collecting entries for the $k$ different values of $Add[].cnt$. The processing time of the algorithm is given by $\mathcal{O}(k|E|)$. □

## 4.2 Locating Vertex Disjoint Embeddings

At Line 3 of Algorithm 2, the more detailed steps involve first enumerating the embeddings of a graph $g$ and then forming VD-embeddings from these embeddings. When a graph $g$ has multiple embeddings that are vertex-disjoint, it is non-trivial to determine which embeddings we should pick to obtain the set of VD-embeddings. We illustrate this problem by the following example.

*Example 5.* Figure 9 shows a graph $G$ and four embeddings of a subgraph $g$ in $G$. Now let us consider only VD-embeddings. If we first include $g_2$, then we only get one VD-embedding of $g$. However, if we select $g_1$, $g_3$ and $g_4$ first, then we have three VD-embeddings of $g$, as highlighted by the three dashed circles.

This example shows that the selection of the VD-embeddings may affect the computational efficiency and the information preservation. For example, let $k = 3$, if we select $g_1$, $g_3$ and $g_4$, we do not need to create any new embedding. If we select $g_2$, then we need to create two new VD-embeddings for $g$, which increases both computational cost and information loss. □

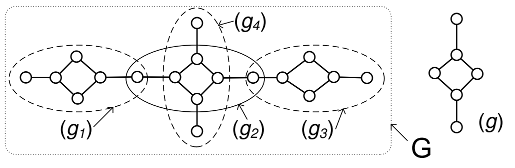

**Figure 9: Four Embeddings of a subgraph $g$**

Example 5 reveals that we have an optimization problem at hand, namely, in the selection of VD-embeddings from the set of all embeddings, the number of VD-embeddings should be maximized. We formally transform this problem into a *maximum independent set* problem as follows.

*Definition* 12 (MAXIMUM INDEPENDENT SET $MIS(g, G)$). *Let G be a graph and g be a subgraph in G. We define a* maximum independent set *problem with respect to G and g, denoted as $MIS(g, G)$, as follows.*

- *Input: A graph, $G_I = (V, E)$, where $V(G_I) = embed(g, G)$ and $E(G_I) = \{(g_i, g_j) : g_i, g_j \in embed(g, G), g_i \neq g_j$, and $\exists v$ such that $v \in V(g_i)$ and $v \in V(g_j)\}$.*
- *Output: A set, O, where $O \subseteq V(G_I)$, such that $\forall g_i, g_j \in O$, $(g_i, g_j) \notin E(G_I)$, and $|O|$ is maximized.*

*Example 6.* Consider the input graph $G_I$ to the independent set problem of the graph embeddings in Figure 9. The set of embeddings, $\{g_1, g_2, g_3, g_4\}$, defines $V(G_I)$. Since $g_2$ shares vertices with $g_1$, $g_3$ and $g_4$, there is an edge between $g_2$ and each of $g_1$, $g_3$ and $g_4$. The output, i.e., the maximum independent set, is $O = \{g_1, g_3, g_4\}$. □

The following lemma formally establishes the connection between the maximum independent set problem and our problem of maximizing the number of VD-embeddings.

LEMMA 2. *Let G be a graph and g be a subgraph in G. Let N be the maximum number of VD-embeddings of g in G, and O be the output of $MIS(g, G)$. Then, $N = |O|$.*

PROOF. Let $G_I$ be the input of $MIS(g, G)$. By the construction of $G_I$, for any $g_i, g_j \in V(G_I)$, $(g_i, g_j) \notin E(G_I)$ implies that $g_i$ and $g_j$ are disjoint. Therefore, the embeddings of $g$ in any independent set for $G_I$ are disjoint, and hence the maximum independent set defines the largest set of VD-embeddings of $g$. □

For better efficiency, there are polynomial-time approximation algorithms [8] which can be used to find a sub-optimal maximum independent set, whose size is in general greater than $\lceil n/(d + 1)\rceil$ for an input graph with $n$ vertices and average degree $d$, such mechanisms can greatly speed up maximum independent set computations.

## 4.3 Analysis and Discussion

Next we analyze the correctness, the performance in time complexity and data quality, and some properties of our algorithms. First we show that Algorithm 2 achieves $k$-security.

THEOREM 4. *Let G be the input graph of Algorithm 2 and $G_k$ be the anonymized graph which is the output of Algorithm 2. Then, $G_k$ is k-secure.*

PROOF. The resulting graph $G_k$ is made up of $k$ pairwise isomorphic disjoint subgraphs $g_1, ..., g_k$. From Theorem 2 $G_k$ is $k$-secure. □

Next we consider the computation cost, which is closely related to the number of PAGs being processed.

LEMMA 3 (COMPUTATION COST). *Given an input graph G, Algorithm 2 anonymizes at most $\mathcal{O}(|V(G)|)$ PAGs.*

PROOF. In anonymizing each PAG we add at least one vertex to each $g_i$. We can at most add $\frac{|V(G)|}{k} = \mathcal{O}(|V(G)|)$ vertices to $g_i$, hence at most $\mathcal{O}(|V(G)|)$ PAGs are anonymized. □

According to Lemma 3, the worst-case complexity of Algorithm 2 is $\mathcal{O}(|V(G)| \times A + B + C)$, where $\mathcal{O}(A)$ is the cost of anonymizing one PAG by Algorithm 3, $\mathcal{O}(B)$ is the cost of computing all PAGs in Line 2 of Algorithm 2, and $\mathcal{O}(C)$ is the cost of adding edges by Algorithm 4 to complete the formation of i-graphs.

Let $z = maxPAGsize$, and $f = freq(g)$ for a PAG $g$, $O(A) = O(2^f + cf2^z)$, where $O(2^f)$ is the cost of computing $MIS(g, G)$, and $O(2^z)$ is cost for a subgraph isomorphism (Lines 5-6 of Algorithm 2), and $c$ is the number of supergraphs $g''$ of $g'$ in $\mathcal{M}$. The factor $\mathcal{O}(B)$ includes the cost of enumerating all the embeddings of the PAGs by the depth-first exploration of $G$. We use hashing to determine to which PAG an enumerated embedding belongs, which takes $\mathcal{O}(1)$ subgraph isomorphism tests. Therefore, $\mathcal{O}(B) = \mathcal{O}(N \times 2^z)$, where $N$ is the total number of all embeddings of all the PAGs and $O(2^z)$ is the cost of one subgraph isomorphism test for a graph $g$ of size $z$. From Lemma 1, $\mathcal{O}(C) = \mathcal{O}(k|E(G)|)$.

In general, $\mathcal{O}(|V(G)| \times A)$ dominates the total cost since each $\mathcal{O}(A)$ involves a computation of maximum independent set for locating VD-embeddings at Line 3 of Algorithm 2 and a number of subgraph isomorphism tests (Lines 5-6 of Algorithm 3). However, since $embed(g')$ can be re-used, the average number of subgraph isomorphism tests is small. We emphasize that in our problem the subgraph isomorphism tests operate on inputs of very small sizes.

The annoymization scheme in [32] shows good performance on different social networks because many vertices share similar neighborhoods. In most social networks most people belong to small groups of closely connected friends or acquaintances, which are well modeled by the PAGs, we believe that it is easy to find PAGs sharing similar structures and our anonymization algorithm also will have good performance.

In the worst case, we add or delete no more than $k|E(G)|/2$ edges to $G$. The worst case scenario is also the addition of $(k - 1)|E(G)|$ edges for the anonymization in [32]. However, as pointed out in [32], recent studies such as [26] and [29] report that high symmetry is found in real networks, and such symmetry enables our algorithm to minimize changes to the graph.

Finally, we should mention that our method also protects again node re-identification, supporting $k$-anonymity as studied in previous works, and $G_k$ is $k$-automorphic.

LEMMA 4. *A k-isomorphic graph $G_k$ is k-automorphic.*

PROOF. Let $G_k = \{g_1, ..., g_k\}$ and $h_{ij}$ be the isomorphism function from $g_i$ to $g_j$. Then $\{h_{12}, h_{23}, ..., h_{k1}\}$ forms an automorphic function $F_1$. $\{h_{13}, h_{24}, ..., h_{k2}\}$ forms an automorphic function $F_2$. ... $\{h_{1k}, h_{21}, ..., h_{k(k-1)}\}$ forms an automorphic function $F_{k-1}$. For each vertex $v$ in $G_k$, $F_i(v) \neq F_j(v)$ for $i \neq j$. From Definition 3.1 in [33], $G_k$ is $k$-automorphic. □

## 4.4 Dynamic Releases

So far we focus on a single data release for a social network. In general the data may be evolving and published dynamically. As pointed out in [33], if we keep the same vertex ID for the node of each individual over multiple graph releases for better utility, it is possible that the adversary can succeed in re-identification by intersecting the anonymization vertex groups for a target individual over the releases.

For example, given 2 releases, $R1$ and $R2$. Suppose that in $R1$, the published graph is $G_k^1 = \{g_1^1, ..., g_k^1\}$. Vertex ID $w$ appears in a partition in $G_k^1$. From the adversary's NAG$_w^1$, $w$ is one of $k$ vertices (from different $g_j^1$'s) that match a target individual $o_w$. Let us call this set of $k$ vertex IDs $W_1$. In Release $R2$, the published graph is $G_k^2 = \{g_1^2, ..., g_k^2\}$. Vertex ID $w$ appears in a partition in $G_k^2$, and from adversary's NAG$_w^2$, $w$ is one of $k$ vertices (from different $g_j^2$'s) that match $o_w$. Let us call this set of $k$ vertex IDs $W_2$. All other vertex IDs in $W_2$, do not appear in $W_1$. Hence $w$ is the only vertex that maps to both NAG$^1_w$ and NAG$^2_w$. This results in node-reidentification of $o_w$, and both NodeInfo and LinkInfo are jeopardized.

Note that vertex ID is different from individual ID in that it only serves the purpose of tracing the node in multiple releases. [33] is the only work known to us that deals with dynamic releases of graph data. They propose a method to generalize certain vertex IDs when necessary.

Our method also makes use of generalized vertex IDs. After we form a $k$-isomorphic graph $G_k$, we have a vertex mapping table $VM$ consisting of tuples of the form $VM[r] = \{v_{r1}, v_{r2}, ..., v_{rk}\}$, where $v_{ri}$ is the vertex ID for a node that is in subgraph $g_i$ and is mapped to the other $v_{rj}$'s via the isomorphic functions, $h_{ij}(v_{ri}) = v_{rj}$. With multiple releases, instead of releasing the graph with vertex IDs of $v_{ri}$, for each tuple $VM[r]$ in the table $VM$, we form a compound vertex ID of $\{v_{r1}, v_{r2}, ..., v_{rk}\}$, this compound ID will replace each of the vertex IDs of $v_{r1}$, $v_{r2}$, ..., $v_{rk}$ in $G_k$, and the resulting graph $G'_k$ is published. We refer to the original vertex IDs of $v_{ri}$ as *simple vertex IDs*.

There is no need of special handling for vertex addition or deletion. For vertex deletion, the simple vertex ID of the deleted vertex simply will not appear in the new data release. For vertex addition, the simple vertex ID of the new vertex will be part of a compound ID like any other vertex.

With compound vertex IDs, intersecting the anonymization vertex groups for a target individual over the releases will not identify any node for any individual since in each release there are $k$ vertices with the same compound vertex ID.

THEOREM 5. *Given a series of network graph releases. Assume adversary knowledge of NAG's for multiple data releases, and also of individuals joining or leaving the network at each release. If anonymized graphs are published based on our anonymization by $k$-isomorphism and the above compound vertex ID mechanism, then each published graph is $k$-secure.*

Though the compound vertex method in [33] also solves the above problem, our method has a number of advantages. Firstly, no record of past releases need to be maintained and processed in each release. The processing is very simple. There is no distortion in the way of including vertex IDs of deleted vertices. Finally, the compound vertex ID size is given by $k$, where compound vertex ID size refers to the number of simple vertex IDs in the compound vertex ID. This compares favorably to a bound of $2k - 1 + \delta$ for the generalized vertex ID size in [33], where $\delta$ is the number of vertices in the first release divided by that in the current release.

## 5. EXPERIMENTAL EVALUATION

In our experiments, we have used 5 datasets from 3 different applications: Hep-Th, EUemail and LiveJournal.

The **HEP-Th** database presents information on papers in theoretical high-energy physics. The data in this dataset were derived from the abstract and citation files provided for the 2003 KDD Cup competition. The original datasets are from arXiv. We extract a subset of the authors and considered the authors linked if they wrote at least one paper together. There are 5618 vertices and 11786 edges in this data graph. **EUemail** is a communication network dataset generated using data from a large European research institution (http://snap.stanford.edu/data/email-EuAll.html), for a period from October 2003 to May 2005. The nodes in the network are email addresses and edges represent email communication between two nodes. There are 265214 nodes, from which we randomly extracted 5000 and 10,000 vertices to form 2 datasets, Email-1 and Email-2, respectively. **LiveJournal** is an online journaling community[1]. The nodes are users and edges represent relationship of friend-lists between users. The entire dataset contains 4847571 nodes, from which we randomly extracted 5000 and then 20,000 vertices to form 2 datasets LiveJ-1 and LiveJ-2.

In previous works such as [11, 19, 30, 33], it is recognized that the utility of graph data depends on the preservation of some graph properties and their empirical studies are based on such properties. Here we use the measurements as adopted by such previous works to capture the utilities of the anonymized graphs. (1) The first measurement, $Degree$, is the distribution of the degrees of all vertices in the graph. (2) The second measurement, $Pathlength$, is a distribution of the lengths of the shortest paths between 500 randomly sampled pairs of vertices in the network. (3) $Transitivity$, also known as the *clustering coefficient*, is also a distribution, for each vertex, we measure the proportion of all possible neighbor pairs that are connected.

All the programs are implemented in C++. The experiments are performed on a Linux workstation with a 2.8Ghz CPU and 4 Gigabyte memory.

### 5.1 Runtime and Data Utilities

Figure 10 shows some graph characteristics and experimental results for the five sets of data. Though Email-1 and Email-2 are extracted from the same dataset, their average degrees are quite different, this is because the original dataset is very large, with 265214 nodes, and extracting different parts of the graph gives rise to different characteristics. The same holds for LiveJ-1 and LiveJ-2.

| | HEP-Th | Email-1 | Email-2 | LiveJ-1 | LiveJ-2 |
|---|---|---|---|---|---|
| Avg degree | 4.20 | 3.96 | 2.67 | 7.14 | 2.65 |
| Number of vertices | 5618 | 5000 | 10000 | 5000 | 20000 |
| Runtime (sec) | 456 | 3798 | 8814 | 7503 | 2559 |
| $\|\|E_k\| - \|E\|\|$ | 76 | 1051 | 1511 | 1377 | 205 |

**Figure 10: Results for different datasets (k=10)**

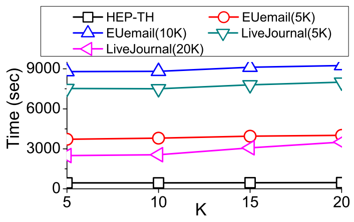

**Figure 11: Runtime for different k values**

Our results show that the dataset of LiveJ-2 with 20000 nodes requires less runtime than LiveJ-1 with 5000 nodes, which seems unexpected. However, on closer inspection, we find that the average degree of LiveJ-1 is much higher, hence even though it has less nodes, it contains many more possible embeddings for subgraphs than LiveJ-1. This shows that the more important factor affecting the performance will be the vertex degrees rather than the number of vertices. Similarly, the effect of $k$ is not very significant as shown in Figure 11, since $k$ does not affect the graph nature related to the costly operations of subgraph isomorphism or MIS. The reason why HEP-Th has lower runtime is because it has higher symmetry in the data graph, which helps to generate a large number of VD-embeddings for graph partitioning.

Figures 12 to 14 show the results of the experiments with respect to the three measurements. We compare the properties of

[1]http://www.livejournal.com/.

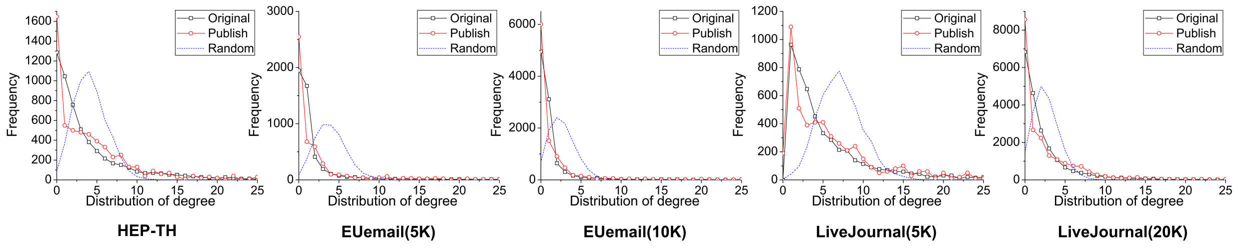

**Figure 12: Distributions of Degrees (k=10)**

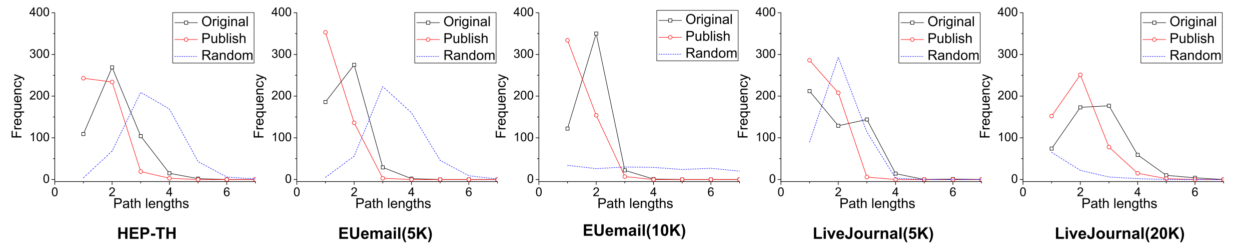

**Figure 13: Distributions of Shortest Path Lengths (k=10)**

the anonymized graph $G_k = (V_k, E_k)$ from our algorithm to the original data graph $G = (V, E)$ for $k = 10$. As in [11], we also consider a random graph as a baseline case. The random graph has been generated by fixing the number of vertices to the same number in the dataset at hand, and also setting the average degree to be the same as the original graph for the dataset. Overall, the anonymized graphs are able to preserve the essential graph information. In most cases the curves for $G_k$ and $G$ are aligned, while the random graph behaves very differently. We have repeated the experiments with $k = 5, 10, 15, 20$ and the utility qualities are very similar.

There is some unusual pattern in Figure 13 in that in the results on EUemail-1(10K) and LiveJ-2(20K), the distributions for the random graphs are quite different from that in other cases. Here the average degrees of the random graphs are very small and the numbers of vertices are large, hence when we sample 500 pairs of nodes to test for the shortest path length, many pairs cannot find a short path.

## 5.2 Dynamic Releases

Since our multiple release mechanism follows closely the single release method, the performance is also similar. Here we compare the performance of our dynamic release mechanism to that in [33], which we refer to as RDVM (Retaining Deleted Vertices for Multi-release). The reason for this comparison is that the mechanism for multiple release in [33] can also be used in our case.

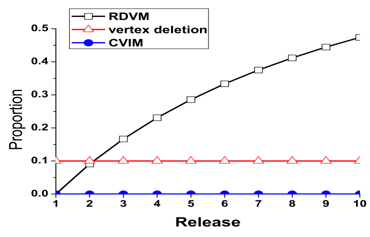

**Figure 15: Phantom vertex ID's over multiple releases**

The comparison is shown in Figure 15, where we plot the proportion of fictitious vertex ID's among all vertex ID's in each release, this result applies to any dataset under the following assumptions: there are 10 data releases, in each data release 10% of the vertices in the first release are deleted, while the same number of new vertices are added. Since we do not modify the set of simple vertices from the current dataset, the distortion is always zero. In the figure our result is labeled CVIM (Compound Vertex ID's for Multi-release). RDVM suffers from high distortion to the data in the amount of fictitious vertices. After 10 releases, the proportion of vertex ID's for deletion vertices[2] climbs to almost 50% of all vertex ID's.

## 6. CONCLUSIONS

We have identified a new problem of enforcing $k$-security for protecting sensitive information concerning the nodes and links in a published network dataset. Our investigation leads to a solution based on $k$-isomorphism. In conclusion, the selection of anonymization algorithm mainly depends on the adversary knowledge and the targets of protection. If the target is only NodeInfo protection against structural attack, a solution as shown in Section 1.4 will suffice. If the adversary may possess other information, then new mechanisms could be needed. Nevertheless, we believe that NodeInfo and LinkInfo are two basic and likely sources of sensitive information in network datasets, and they call for special efforts for their protection.

## 7. ACKNOWLEDGEMENTS

We thank the anonymous referees and Pierangela Samarati for their valuable comments. This research is supported by the RGC Research Direct Grant of the Chinese University of Hong Kong Project 2050421 and 2150472 and Chinese University of Hong Kong Postdoctoral Fellowship Grant 2008-2009.

## 8. REFERENCES

[1] L. Adamic and N. Glance. The political blogosphere and the 2004 us election : divided they blog. In *In Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem*, 2005.

[2] L. Backstrom, C. Dwork, and J. M. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, pages 181–190, 2007.

[2]We clarify that here we have not counted the dummy vertex insertions for graph partitioning in [33]. In Algorithm 5 in [33], VD-embeddings are created by introducing a dummy vertex for each overlapping vertex in two embeddings. Here we only consider dummy vertex ID's they have used for deleted individuals.

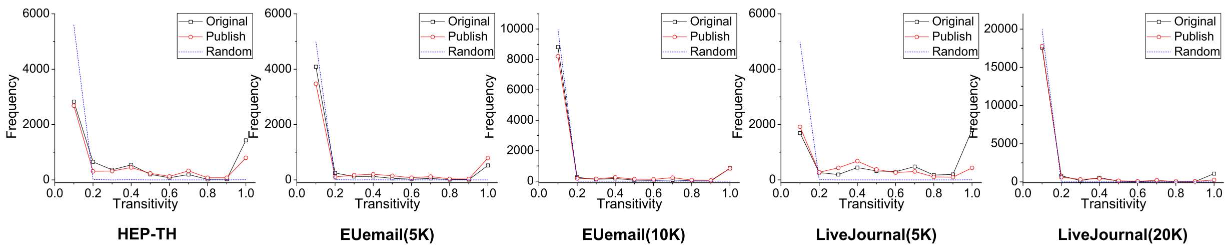

**Figure 14: Distributions of Cluster Coefficients (Transitivity) (k=10)**

[3] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54, 2006.

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

[5] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks by h*-graph. In *To appear in SIGMOD*, 2010.

[6] J. Cheng, Y. Ke, W. Ng, and J. X. Yu. Context-aware object connection discovery in large graphs. *Proceedings of the International Conference on Data Engineering (ICDE)*, 2009.

[7] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. *PVLDB*, 1(1):833–844, 2008.

[8] A. Dharwadker. The independent set algorithm. *http://www.geocities.com/dharwadker/independent_set/*, 2006.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[10] L. Getor and C. Diehl. Link mining: a survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

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

[12] M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava. Anonymizing social networks. *Technical report No. 07-19, Computer Science Department, University of Massachusetts Amherst*, 2007.

[13] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. In *KDD*, pages 581–586, 2004.

[14] Y. Koren, S. C. North, and C. Volinsky. Measuring and extracting proximity in networks. In *KDD*, pages 245–255, 2006.

[15] G. Kossinets, J. M. Kleinberg, and D. J. Watts. The structure of information pathways in a social communication network. In *KDD*, pages 435–443, 2008.

[16] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *KDD*, pages 611–617, 2006.

[17] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. In *the SIAM International Conference on Data Mining*, 2004.

[18] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In *KDD*, pages 462–470, 2008.

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

[20] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, page 24, 2006.

[21] A. McCallum, A. Corrada-Emmanuel, and X. Wang. Topic and role discovery in social networks. In *Proc. of International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[22] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(6):1010–1027, 2001.

[23] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, SRI International, 1998.

[24] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[25] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

[26] H. Wang, G. Yan, and Y. Xiao. Symmetry in world trade networks. *Journal of Systems Science and Complexity*, 22(2):280–290, 2008.

[27] R. Wong, J. Li, A. Fu, and K. Wang. (alpha, k)-anonymity: An enhanced k-anonymity model for privacy-preserving data publishing. In *KDD*, Aug 2006.

[28] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *VLDB*, 2006.

[29] Y. Xiao, M. Xiong, W. Wang, and H. Wang. Emergence of symmetry in complex networks. *Physical Review E*, 77(6), 2008.

[30] X. Ying and X. Wu. Randomizing social networks: a spectrum preserving approach. In *SDM*, pages 739–750, 2008.

[31] E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *PinKDD*, pages 153–171, 2007.

[32] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *ICDE*, pages 506–515, 2008.

[33] L. Zou, L. Chen, and M. T. Ozsu. K-automorphism: A general framework for privacy preserving network publication. In *VLDB*, 2009.

## 9. APPENDIX

PROOF. (Theorem 1 : NP-Hardness of $k$-Secure-PPNP) Given a problem of PARTITION INTO TRIANGLES [9] as follows

INSTANCE: A graph $G = (V, E)$, with $|V| = 3k$ for a positive integer $k$.

QUESTION: Is there a partition of $V$ into $k$ disjoint sets $V_1, ..., V_q$ of three vertices each such that, for each $V_i = \{v_{i[1]}, v_{i[2]}, v_{i[3]}\}$, the three edges $\{v_{i[1]}, v_{i[2]}\}$, $\{v_{i[2]}, v_{i[3]}\}$, $\{v_{i[1]}, v_{i[3]}\}$, all belong to $E$ ?

We transform the above problem to a decision problem of $k$-security: The graph $G = (V, E)$ from the above instance is given as the input graph, the problem is whether there is an anonymization of the graph to a graph $G_k$ by edge augmentation with an anonymization cost where $||E(G_k)| - |E(G)|| \leq |E| - |V|$ and $ED(G, G_k)$ of at most $|E| - |V|$ so that the published data is $k$-secure.

The transformation takes polynomial time. Next we show that this is indeed a transformation. If there is a partition into triangles for $G$, then from Theorem 2, the partitioned graph is a $G_k$ that satisfies $k$-security and $||E(G_k)| - |E(G)|| = ED(G, G_k) = |E| - 3k = |E| - |V|$.

Conversely suppose there is no partition into triangles for $G$. Then it is not possible to partition the graph into $k$ isomorphic components with the given cost constraints. From Theorem 3, $G_k$ must contain more than $k$ disjoint connected subgraphs, otherwise, there is no solution for $k$-security, since we cannot form $k$ disjoint isomorphic connected subgraphs. Hence $G_k$ contains $m$ disjoint connected subgraphs, where $m > k$. From Corollary 2, the biggest disjoint subgraph in $G_k$ must contain at most 2 vertices. Let there be $t$ disjoint subgraphs with 2 vertices and one edge, the remaining disjoint subgraphs are single vertex subgraphs. The number of edges in $G_k$ is given by $|E(G_k)| = t$ where $t \leq |V|/2$. Hence $|E(G_k)| \leq |V|/2$. Then $||E(G_k)| - |E|| = ED(G, G_k) = |E| - |E(G_k)| \geq |E| - |V|/2 > |E| - |V|$. This violates the anonymization cost requirement of the $k$-security problem. Hence the solution for the $k$-security problem is also negative. $\square$