Protecting Sensitive Relationships against Inference Attacks in Social Networks*

Xiangyu Liu and Xiaochun Yang

College of Information Science and Engineering,
Northeastern University, Liaoning, 110819, China
neulxy@gmail.com, yangxc@mail.neu.edu.cn

Abstract. The increasing popularity of social networks in various application domains has raised privacy concerns for the individuals involved. One popular privacy attack is identifying sensitive relationships between individuals. Simply removing all sensitive relationships before releasing the data is insufficient. It is easy for adversaries to reveal sensitive relationships by performing link inferences. Unfortunately, most of previous studies cannot protect privacy against link inference attacks. In this work, we identify two types of link inference attacks, namely, *one-step link inference* attacks and *cascaded link inference* attacks. We develop a general framework for preventing link inference attacks, which adopts a novel lineage tracing mechanism to efficiently cut off the inference paths of sensitive relationships. We also propose algorithms for preventing *one-step link inference* attacks and *cascaded link inference* attacks meanwhile retaining the data utility. Extensive experiments on real datasets show the satisfactory performance of our methods in terms of privacy protection, efficiency and practical utilities.

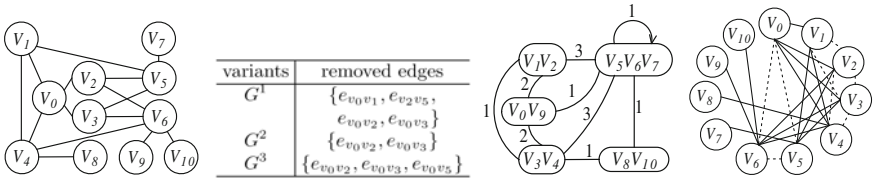
1 Introduction

In recent years, a fast growing popularity in social networks has attracted the interests of researchers from different disciplines. Exploring the properties of social networks has generated interesting knowledge discovery and data mining problems. However, social networks usually contain individuals' sensitive information. Preserving privacy in the release of social network data becomes an important concern.

One fundamental privacy issue in publishing social network data is link re-identification problem. In social network, the main entities are individuals who participate in thousands of interactions with each other. An edge (or link) refers to an interaction between two individuals involved, and there are many different kinds of interactions. In Email network, an edge connecting two people indicates that they communicate through emails. Some interactions or relationships are

* The work is partially supported by the National Natural Science Foundation of China(Nos. 61173031, 60973018) and the Fundamental Research Funds for the Central Universities(Nos. N090504004, N100604013).

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 335–350, 2012.



variants	removed edges
G^1	$\{e_{v_0v_1}, e_{v_2v_5}, e_{v_0v_2}, e_{v_0v_3}\}$
G^2	$\{e_{v_0v_2}, e_{v_0v_3}\}$
G^3	$\{e_{v_0v_2}, e_{v_0v_3}, e_{v_0v_5}\}$

(a) G (b) Variants of G (c) G_a (d) G_1

Fig. 1. Multiple versions of graph G

considered as sensitive due to the involved individuals. For instance, in telecommunication network, *Adam* and *Bob* would like to prevent the disclosure of their close relationship, which can be observed based on the frequent telephone calls between them.

1.1 Motivation

In the process of anonymizing data, sensitive relationships are always removed, i.e., they are not provided in the released data in order to protect individuals' privacy. However, it may be possible to predict or infer some of these relationships through the techniques of *link inference*, which has been widely investigated in recent years. We name inferring sensitive relationships in social network as *link inference attack*. A common sense we have known is that if two individuals have many friends in common, these two individuals are likely to know each other. In Fig.1(a), v_0 has three common neighbors with v_5, and only one common neighbor with v_8. We infer the probability of there existing a link between v_0 and v_5 is larger than that between v_0 and v_8. Link inference techniques could be adopted by adversaries to identify sensitive relationships with high probability. We empirically evaluated the re-identification power of link inference on real social networks, **Email-1** and **LiveJ-1**[10], to demonstrate that such intuition really holds. In our evaluation, for any two vertices, we considered there existed an edge between them if the count of their common neighbors was larger than a threshold δ and we studied the impact of δ on link inference. We report the statistical data of true positive instances (TP) and true negative instances (TN), and calculate corresponding rates of TP and TN, as illustrated in Table 1. When threshold $\delta = 2$, the rates of correctly inferred edges are only 9.12% and 25.49% for **Email-1** and **LiveJ-1**, respectively. For both datasets, the rates of correctly inferred edges become higher as δ increases. When $\delta = 18$, TPR in **LiveJ-1** even reaches 86.98%, and 63.61% in **Email-1**. TNR gets slightly decreased when δ increases but still keeps as high as 99%. From our evaluation results in Table 1, we clearly see that the re-identification power of link inference is strong enough to help adversaries identify sensitive relationships with high probability. Hence, link inference attacks are real in practice.

Previous work on protecting privacy of social network does not consider link inference attacks. Link inference associated information can be studied by adversaries, even though the releasing graph is anonymized through existing methods. Given graph G in Fig.1(a), according to existing methods [3,9], we obtain an anonymized graph G_a in Fig.1(c). G_a guarantees the probability of an adversary re-identifying any vertex or edge is at most $\frac{1}{k}$. For G_a, the constant k is 2. Obviously, the expected number of common neighbors between v_0 and v_5 is $1 \times \frac{6656}{14400} + 2 \times \frac{3216}{14400} + 3 \times \frac{416}{14400} + 4 \times \frac{16}{14400} = 1$, and the expected number is $1 \times \frac{252}{864} + 2 \times \frac{18}{864} = \frac{288}{864}$ between v_0 and v_8. However, the adversaries can still infer that the probability of v_0 having a link with v_5 is larger than with v_8, which is the same with previous conclusion that we draw based on original graph G.

Table 1. Re-identification power of link inferences on real social networks

δ	Email-1				LiveJ-1			
	TP	TPR(%)	TN($\times 10^7$)	TNR(%)	TP	TPR(%)	TN($\times 10^7$)	TNR(%)
2	6425	9.12	1.242	99.96	12028	25.49	1.244	99.95
6	3849	26.80	1.248	99.94	6495	57.04	1.247	99.91
10	2568	41.93	1.248	99.93	3896	69.62	1.248	99.89
14	1722	53.76	1.248	99.93	2504	79.14	1.248	99.88
18	1117	63.61	1.249	99.92	1550	86.98	1.248	99.87

1.2 Challenges and Contributions

We address the problem of link inference attacks in social networks. The challenge of our problem lies in the complexity of link inferences in social network. Typically, the adversary can perform inference attack with multiple steps. In order to avoid privacy leakage, we need to modify the social graph. Due to the complexity of graph, a little modification (e.g., remove an edge) may incur a great impact on link inferences. How to cut off the inference paths of sensitive relationships without privacy leakage meanwhile incurring minimal information loss has proposed a great challenge.

Our contributions can be summarized as follows. (1) We discuss two types of link inference attacks, namely, *one-step link inference* attacks and *cascaded link inference* attacks, which have strong link re-identification power in real networks. (2) We propose *inference security* to protect privacy against link inference attacks. (3) We develop a general framework for preventing link inference attacks, which adopts a novel lineage tracing mechanism to efficiently cut off the inference paths of sensitive relationships. (4) We propose algorithms for preventing *one-step link inference* attacks and *cascaded link inference* attacks. (5) We design a number of techniques to make the inference preventing methods efficient meanwhile maintaining the utility. (6) Our extensive empirical studies show that our methods perform well in real datasets.

The remainder of the paper is organized as follows. Related work are summarized in Section 2. In Section 3, we give the problem definition and formalize the inference security model. We outline a general framework for preventing link

inference attacks, and propose algorithms for obtaining inference secure graphs in Section 4. We evaluate our methods in Section 5. Section 6 concludes the paper.

2 Related Work

With the increasing popularity of social networks, protecting privacy information in social networks while retaining data utility for data mining and analysis has become an interesting problem that has been studied by a number of recent works. Privacy attacks in social networks are mainly classified into two categories, including *vertex re-identification* attacks and *link re-identification* attacks.

In vertex re-identification attacks (a.k.a. identity disclosure), an adversary identifies the identities of vertices in the published network using the subgraphs associated with target individuals as background knowledge. Liu et al. [1] propose k-degree to prevent from privacy attacking using vertex degree as adversary knowledge. Zhou et al. [2] provide identity privacy through anonymizing the 1-neighborhood subgraph of each vertex. Hay and Campan [3,4] propose to protect identity privacy against subgraph knowledge through clustering vertices into super vertices, where the vertices in a super vertex are indistinguishable from each other. Zou et al. [5] propose a privacy preserving model K-Automorphism for protecting identity privacy.

In link re-identification (i.e., link disclosure) attacks, an adversary aims at identifying sensitive relationships among the individuals in social network. Zheleva et al. [6] discuss a number of privacy preserving strategies to prevent sensitive edge disclosure, in which the protection of link privacy cannot be guaranteed. Ying et al. [7] study graph randomization through adding/removing and switching edges randomly while preserving the spectrum of the network, which do not provide quantifiable privacy protection. Cormode et al. [8] propose a permutation based approach to protect the privacy of links in bipartite graphs. Bhagat et al. [9] improve the work in [8] and study graph anonymization to protect link privacy based on vertices grouping.

Different from considering identity disclosure and link disclosure problems separately, recent work [10,11] has focused on proposing general frameworks to protect both identity and link privacy. Cheng et al. [10] study how to partition and anonymize a releasing graph into disjoint k subgraphs that are isomorphic to each other for ensuring the probabilities of identifying an identity and a sensitive link both at most $\frac{1}{k}$. Besides protecting identity and link privacy, Yuan et al. [11] give a solution to satisfy different needs on privacy protecting level.

One fundamental problem underlying all of the research work mentioned above is that each of them assumes adversaries acquire privacy information using target individual's associated graph structural knowledge, and ignores that it's very possible for adversaries to infer privacy using graph inference knowledge. In this work, we try to protect sensitive relationships in a releasing graph against link inference attacks.

3 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 use $V(G)$ and $E(G)$ to refer to the vertex set and edge set of G. In this work, we also use link and relationship interchangeably to denote an edge.

Generally, link inference techniques are classified into three categories, including similarity based link inference, maximum likelihood link inference and probabilistic models based link inference. In this work, we choose common neighbor similarity based link inference (a similarity based link inference technique) as our specific link inference preventing problem.

Definition 1. *(common neighbor similarity) The common neighbor similarity of vertices u and v is defined as the count of common neighbors of u and v, denoted as $Sim_{CN}(u, v)$, which is formalized as Equation 1,*

$$Sim_{CN}(u, v) = |\Gamma(u) \cap \Gamma(v)| \tag{1}$$

where $\Gamma(u)$ refers to the neighbors of u.

We choose common neighbor similarity based link inference (LI_{CN} for short) as adversaries' graph inference knowledge for the following reasons:

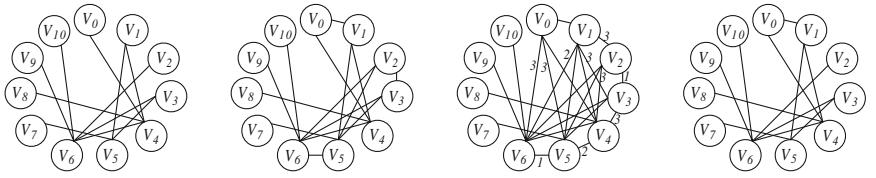
(1) As shown in Table 1, LI_{CN} is simple, effective and has strong link re-identification power, which helps adversaries identify sensitive relationships with high probability;
(2) For adversaries, LI_{CN} is easy to implement. The necessary information for adversaries to perform LI_{CN} are the common neighbor sets of the two target individuals, which are easily collected in real social networks. For instance, in Online Social Networks (OSNs), such as Facebook and MySpace, after the process of sending a friend application to a user and being one of his/her friends, you can access all his/her friends (i.e., neighbors).

Definition 2. *(sensitive edge) Given graph $G(V, E)$, if edge e_{uv} is defined as sensitive by data owners or two involved individuals u and v, then e_{uv} should not exist in G, and vertex pair (u, v) is denoted as sensitive pair.*

In this work, we assume *some* (not all) edges in G are defined as sensitive, and prevent the disclosures of these edges due to link inferences. As shown in Table 1, although an edge is removed, the existence of this edge could be inferred with high probability based on graph inference knowledge.

Definition 3. *(link inference) Given graph $G(V, E)$ and threshold δ, if the common neighbor similarity of vertices u and v satisfies $Sim_{CN}(u, v) \geq \delta$, it is inferred that there exists an edge between u and v.*

For two individuals with a sensitive relationship, the data owner provides a minimum threshold δ to specify his tolerance of revealing closeness of them. Threshold δ can also be personalized by these two involved individuals.



(a) G_1^1 (b) G_1^2 (c) G_3^2 (d) G_3^3

Fig. 2. Cascaded link inferences

Definition 4. *(one-step link inference) Given graph G(V,E) and threshold δ, one-step link inference on G refers to performing link inference on each unconnected vertex pair in G, such that for any unconnected vertex pair* (u, v) *with* $Sim_{CN}(u, v) \geq \delta$*, we add* e_{uv} *to* $E(G)$*.*

Given graph G in Fig.1(a) and δ=2, after one-step link inference on G, we obtain G_1 in Fig.1(d), where $\{e_{v_0v_5}, e_{v_0v_6}, e_{v_1v_2}, e_{v_1v_3}, e_{v_2v_3}, e_{v_2v_4}, e_{v_3v_4}, e_{v_5v_6}\}$ (dotted edges) are newly inferred edges.

Definition 5. *(cascaded link inference) Given graph* $G(V, E)$*, threshold* δ *and an integer i, i-times cascaded link inference (i-inference for short) on G refers to performing one-step link inference iteratively on G for i times, and we obtain the i-inference graph of G, denoted as* G_i*.*

In cascaded link inference, if an edge is inferred from the original graph, this edge is also used for future link inference. Obviously, one-step link inference is the special case of i-inference when $i = 1$, i.e. 1-inference. G_0 denotes graph G with no link inference. $E(G_i)\backslash E(G_{i-1})$(we define G_{-1}=ϕ) refers to the inferred edges in the i-th one-step link inference.

Definition 6. *(inference secure) Given graph* $G(V, E)$*, sensitive edge set S, an integer i* $(i \geq 0)$*, and threshold δ, if* $S \cap E(G_i) = \phi$*, then G is i-inference secure.*

Example 1. Given G in Fig.1(a), Fig.1(b) lists three variants of G with some edges removed. After 3-inference on G^2 in Fig.1(b), we obtain G_3^2 in Fig.2(c), where edges labeled with integer i $(i = 1, 2, 3)$ are inferred in i-th one-step link inference. For instance, $e_{v_1v_6}$ is inferred in the 2*nd* one-step link inference and belongs to $E(G_2^2)\backslash E(G_1^2)$. Let $S = \{e_{v_0v_5}, e_{v_0v_6}\}$ and $\delta = 2$, since $S \cap E(G_2^2) = \phi$ and $S \cap E(G_3^2) = S$, G^2 is 2-inference secure, not 3-inference secure. Similarly, G is 0-inference secure and G^3 is 3-inference secure.

Theorem 1. *Given graph G, sensitive edge set S, and threshold* δ*, if* $\forall e_{uv} \in S$ *satisfies* $e_{uv} \notin E(G)$ *and* $Sim_{CN}(u, v) < \delta$*, then G is 1-inference secure.*

Proof. After one-step link inference on G, we obtain G_1. For $\forall e_{uv} \in S$, we have $Sim_{CN}(u, v) < \delta$ and $e_{uv} \notin E(G)$, thus e_{uv} would not exist in $E(G_1)$ due to link inference. Hence, G is 1-inference secure.

Problem 1. (Optimal Inference Security) Given graph G, sensitive edge set S, integer i and threshold δ, find an i-inference secure graph G' with $V(G)=V(G')$ and $E(G)\cap E(G')=E(G')$, such that $|E(G)\backslash E(G')|$ is minimized.

Theorem 2. *The problem of Optimal Inference Security is NP-hard.*

Proof. The proof is by reducing the NP-complete problem of SATISFIABILITY [12]. Limited by space, we omit the details here.

In the next section, we derive a novel lineage tracing mechanism to efficiently cut off inference paths of sensitive relationships meanwhile incurring low information loss.

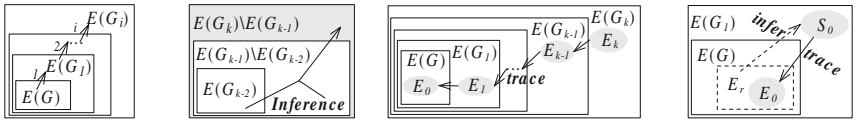
4 Preventing Link Inference Attacks

In this section, we study preventing link inference attacks in social networks. We first outline the general framework for preventing link inference attacks, then propose algorithms for preventing one-step link inference attacks and cascaded link inference attacks.

4.1 A General Framework

Given graph G that is not i-inference ($i\geq 0$) secure, i.e. $S\cap E(G_i)\neq\phi$, we provide lineage tracing mechanism to cut off the inference paths of $S\cap E(G_i)$. Informally, all edges that contribute to the inference of $S\cap E(G_i)$ are considered as the lineage of $S\cap E(G_i)$. The key is to efficiently find inference paths of $S\cap E(G_i)$.

Fig.3(a) describes the process of obtaining G_i, where the arrow line labeled with integer k (k=1,...,i) refers to the k-th one-step link inference. Clearly, edges in $E(G_k)\setminus E(G_{k-1})$ are due to the combinational impact of $E(G_{k-1})\backslash E(G_{k-2})$ and $E(G_{k-2})$, as depicted in Fig.3(b). Intuitively, we can prevent inferring edges in $E(G_k)\backslash E(G_{k-1})$ through removing edges in $E(G_{k-1})\backslash E(G_{k-2})$. We formalize this intuition in Theorem 3.



(a) Obtain G_i (b) Union infer (c) Lineage tracing (d) Prevent S_0

Fig. 3. Cutting off inference paths through lineage tracing

Theorem 3. *Given graph $G(V,E)$, edges in $E(G_k)\backslash E(G_{k-1})$ can be prevented from inferring through removing edges in $E(G_{k-1})\backslash E(G_{k-2})$.*

Proof. (Proof by Contradiction.) Assume to the contrary that there exists an edge e in $E(G_k)\backslash E(G_{k-1})$ that cannot be prevented from inferring through removing edges in $E(G_{k-1})\backslash E(G_{k-2})$, i.e. edge e can still be inferred after removing $E(G_{k-1})\backslash E(G_{k-2})$ from G_{k-1}. Thus, e can be inferred based on $E(G_{k-2})$ through one-step link inference and e is obviously in $E(G_{k-1})\backslash E(G_{k-2})$, contradicting our assumption that e is in $E(G_k)\backslash E(G_{k-1})$.

As shown in Fig.3(c), given graph G and edge set $E_k \subseteq E(G_k)\backslash E(G_{k-1})$, Theorem 3 inspires us that we can prevent inferring E_k by removing edges in $E(G_{k-1})\backslash E(G_{k-2})$, noted as E_{k-1}. Similarly, we prevent inferring E_{k-1} by removing $E_{k-2} \subseteq E(G_{k-2})\backslash E(G_{k-3})$. We perform these lineage tracing and removing operations iteratively until we remove E_0 in $E(G)$. After above operations, edges in E_k would not exist in $E(G_k)$.

Clearly, for graph G that is $(k-1)$-inference secure but not k-inference secure (i.e., $S \cap E(G_{k-1}) = \phi$ and $S \cap E(G_k) \neq \phi$), we can prevent inferring $S \cap E(G_k)$ using lineage tracing and removing operations described in Fig.3(c) and obtain k-inference security for G. Given graph G and an integer i ($i \geq 0$), if we want to obtain i-inference security for G, we can firstly obtain 0-inference secure graph (i.e. remove sensitive edges directly from $E(G)$), and then obtain k-inference ($k=1,\ldots,i$) secure graph based on $(k-1)$-inference secure graph iteratively.

Based on the framework, we propose algorithms for preventing one-step link inference attacks and cascaded link inference attacks in the following subsections, where we will elaborate the technical details of lineage tracing and removing.

4.2 Preventing One-Step Link Inference Attacks

Algorithm 1 protects input graph G against one-step link inference attacks (i.e., to obtain 1-inference secure graph), the process of which is outlined in Fig.3(d). The fact that the graph produced by Algorithm 1 is 1-inference secure is a straightforward result of Theorem 1. The key idea is to make Sim_{CN} of each sensitive edge less than δ.

Algorithm 1 firstly removes sensitive edges from $E(G)$ to obtain 0-inference security (Line 1). Then S_0 is initialized with sensitive edges with $Sim_{CN} \geq \delta$ (Line 2). All edges present in S_0 would be in $E(G_1)$. Removable edge set E_r that contribute to the inference of S_0 is generated (Lines 3-5). In practice, we can obtain 1-inference secure graph through removing $E_0 \subseteq E_r$ from $E(G)$ and E_0 should contain as less edges as possible. In procedure **Remove_Edge**, we provide different strategies for removing edges in E_r (Line 7). While removing edges in E_r, sensitive edges already with $Sim_{CN} < \delta$ should be excluded from S_0 (Line 8). In order to minimize the changes on graph properties due to edge removing, for a removed edge e, the procedure **Find_Add_Edge** finds a new edge e' to add to $E(G)$ (Line 9). Hence, the condition of $E(G) \cap E(G') = E(G')$ in Problem 1 is relaxed to $E(G) \cap E(G') \approx E(G')$. Algorithm 1 performs the procedures **Remove_Edge** and **Find_Add_Edge** repeatedly until graph G is 1-inference secure. We present the details of Algorithm 1 in the followings.

Algorithm 1: Preventing One-Step Link Inference Attacks

Input: Graph $G(V, E)$, sensitive edge set S and threshold δ
Output: 1-inference secure graph G

1 $E(G) \leftarrow E(G) \backslash S$; /* Remove sensitive edges in G */
2 $S_0 \leftarrow \{e_{uv} | \forall e_{uv} \in S \& Sim_{CN}(u, v) \geq \delta\}$; /* Initialize S_0 */
3 **for** *each* $e_{uv} \in S_0$ **do** /* Generate E_r */
4 **for** *each* $w \in \Gamma(u) \cap \Gamma(v)$ **do**
5 Add e_{uw}, e_{wv} to E_r;
6 **repeat**
7 $e \leftarrow$ **Remove_Edge**(S_0, E_r) ; /* Remove edge e in E_r */
8 update S_0;
9 $e' \leftarrow$ **Find_Add_Edge**(e) ; /* Based on e, find a new edge e' */
10 **if** *new edge* e' *for* e *is found* **then**
11 $E(G) \leftarrow E(G) \cup \{e'\}$;
12 **until** S_0 *is empty* ;
13 **return** G;

4.2.1 Generate Removable Edge Set

When we generate removable edge set in Algorithm 1 (Lines 3-5), we only consider edges that contribute to the Sim_{CN} of sensitive edges in S_0. As shown in Algorithm 1, S_0 only contain sensitive edges with $Sim_{CN} \geq \delta$.

Definition 7. *(removable edge set) Given graph G, sensitive edge set S, and threshold δ, we use removable edge set E_r to denote the set of all edges that connect between vertices of sensitive pairs with $Sim_{CN} \geq \delta$ and their common neighbors.*

Example 2. Given graph G in Fig.1(a), let $S = \{e_{v_0v_5}, e_{v_0v_6}\}$ and $\delta = 2$. Since $Sim_{CN}(v_0, v_5)$ and $Sim_{CN}(v_0, v_6)$ both equal to $3 > \delta$, we obtain removable edge set $E_r = \{e_{v_0v_1}, e_{v_0v_2}, e_{v_0v_3}, e_{v_0v_4}, e_{v_1v_5}, e_{v_2v_5}, e_{v_3v_5}, e_{v_2v_6}, e_{v_3v_6}, e_{v_4v_6}\}$.

Note that there always exists a feasible solution to obtain a 1-inference secure graph. In the worst case, all edges in E_r can be removed, i.e. remove $E_0 = E_r$ from $E(G)$. In this way, Sim_{CN} of all sensitive pairs equal to 0; thus, any inference security requirement is satisfied. However, we want to remove *minimum* edges in E_r to obtain a 1-inference secure graph. In the next subsection, we introduce our heuristic strategy of removing edges in E_r.

4.2.2 Remove Edges in E_r

Taking S_0 and E_r as input, **Remove_Edge** removes an edge in E_r and return it. We provide different strategies of removing edges in E_r.

A naive strategy of **Remove_Edge** is that for sensitive edge $e_{uv} \in S_0$ with $Sim_{CN}(u, v) \geq \delta$, we randomly remove $(Sim_{CN}(u, v) - \delta + 1)$ edges in E_r. The removed edges must meet the following conditions: (1) These edges connect between u (or v) and vertices in $\Gamma(u) \cap \Gamma(v)$; (2) None of these edges connect to the same common neighbor. Removing such edges incurs $Sim_{CN}(u, v) < \delta$.

After performing such removing operation on each sensitive edge in S_0, G is 1-inference secure. Since $S_0 \subseteq S$ and $\delta \geq 1$, the computational complexity of naive strategy is $\mathcal{O}(|S|Sim_{max})$, where Sim_{max} denotes the maximal Sim_{CN} of the sensitive edges in S.

In order to prevent inferring S_0 with minimum edges removed in E_r, we design a heuristic function IC(Inference Contribution) to evaluate the contribution of each edge in E_r for the inference of S_0, which is formalized as Equation 2.

$$IC(e_{uv}) = countif_{m \in \Gamma(v)}(e_{um} \in S_0) + countif_{n \in \Gamma(u)}(e_{nv} \in S_0) \tag{2}$$

Clearly, removing the edge with larger IC value results in more decrease on Sim_{CN} of sensitive edges in S_0. A heuristic strategy of **Remove_Edge** is to always remove the edge in E_r with the largest IC. Obviously, heuristic strategy remove at most $|S|Sim_{max}$ edges in E_r. Since removing an edge introduces $|E_r|$ operations to search the edge with the largest IC in E_r and there are at most $2|S|Sim_{max}$ edges in E_r, the computational complexity of the heuristic strategy is $\mathcal{O}(2|S|^2 Sim^2_{max})$.

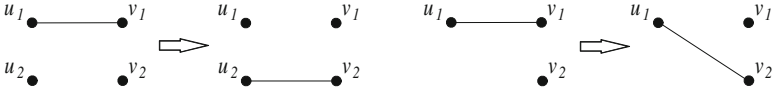
Considering Example 2, for graph G, we obtain G^1 and G^2 in Fig.1(b) with naive and heuristic strategy, respectively. As shown in Fig.2, G^1 and G^2 are both 1-inference secure. However, G^1 incurs more edges removed than G^2.

4.2.3 Add Edges for Preserving Graph Properties

Although existing research work [7] studies graph randomizing techniques meanwhile preserving graph spectrum, it is not realizable to adopt the methods in [7] when the amount of edge modifications is large. We design two efficient strategies on adding new edges for preserving graph properties.

Definition 8. *(Non Inference Contributing Edge) Given graph G, sensitive edge set S, and threshold δ, for an edge $e \notin E(G)$, if adding e to $E(G)$ does not change S_0=$\{e_{uv} | \forall e_{uv} \in S \& Sim_{CN}(u,v) \geq \delta\}$, then edge e is a non inference contributing edge, denoted as nic-edge for simplicity.*

Notice that the procedure **Find_Add_Edge** only consider nic-edges as candidates to ensure new edges would not incur changes of S_0.



(a) Deletion/Addition (b) Switch

Fig. 4. Add new edges for preserving graph properties

Deletion/Addition. The first edge adding strategy is `Deletion/Addition`, which is illustrated in Fig.4(a). When edge $e_{u_1v_1}$ is removed from $E(G)$, we add new edge $e_{u_2v_2}(e_{u_2v_2} \notin E(G))$ to $E(G)$, which is with the largest Structural

Similarity (SS) to $e_{u_1v_1}$. Structural Similarity (SS) is formalized as Equation 3. For a removed edge $e_{u_1v_1}$, there exist at most $\frac{|V(G)|(|V(G)|-1)}{2}-|E(G)|$ new edge candidates.

$$SS(e_{u_1v_1}, e_{u_2v_2}) = \frac{|(\Gamma(u_1) \cup \Gamma(v_1)) \cap (\Gamma(u_2) \cup \Gamma(v_2))|}{|(\Gamma(u_1) \cup \Gamma(v_1)) \cup (\Gamma(u_2) \cup \Gamma(v_2))|} \tag{3}$$

Switch. Fig.4(b) describes another efficient new edge adding strategy, named as `Switch`. Different from `Deletion/Addition`, in `Switch`, when edge $e_{u_1v_1}$ is removed, we add new edge that connects between one vertex in $\{u_1, v_1\}$ and other vertices in $V(G)$. For instance, as shown in Fig.4(b), edge $e_{u_1v_1}$ is removed and new edge $e_{u_1v_2}$ is added to $E(G)$. When adopting `Switch`, Structural Similarity $SS(e_{u_1v_1}, e_{u_1v_2})$ is simplified into $SS(e_{u_1v_1}, e_{u_1v_2})=\frac{|\Gamma(u_1)\cup(\Gamma(v_1)\cap\Gamma(v_2))|}{|\Gamma(u_1)\cup(\Gamma(v_1)\cup\Gamma(v_2))|}$. Obviously, for a removed edge $e_{u_1v_1}$, there exist at most $(2(|V(G)|-2))$ new edge candidates in `Switch`, which is less than `Deletion/Addition`.

Although `Switch` achieves a better efficiency than `Deletion/Addition` through considering less new edge candidates, it performs well in preserving graph properties, which will be shown in the experimental section.

4.3 Avoiding Cascaded Link Inference Attacks

Based on the general framework, we now propose the algorithm of avoiding cascaded link inference attacks (i.e., transform a graph to be i-inference secure), which is shown in Algorithm 2.

Algorithm 2: Avoiding Cascaded Link Inference Attacks

Input: Graph $G(V, E)$, threshold δ, sensitive edge set S and an integer i
Output: i-inference secure graph G

```
1 E(G) ← E(G)\S ;                          /* Make G 0-inference secure */
2 for k = 1 to i do    /* Obtain k-inference secure graph iteratively */
3     S_0 ← E(G_k) ∩ S ;                  /* Obtain sensitive edges in G_k */
4     for j=k−1 to 0 do                   /* Lineage tracing and removing */
5         E_r ← φ, E_s ← φ;
6         generate E_r for S_0 using edges in E(G_j)\E(G_{j−1});
7         remove E_s⊆E_r to prevent inferring S_0;
8         S_0 ← E_s;
9 return G;
```

Algorithm 2 firstly make G 0-inference secure (Line 1), then obtains k-inference ($1 \leq k \leq i$) secure graph iteratively (Lines 2-8). When transform a graph from $(k-1)$-inference secure into k-inference secure (Lines 3-8), we firstly obtain sensitive edges inferred in G_k (Line 3), i.e. $E(G_k) \cap S$. Then, we perform lineage tracing and removing operations iteratively to cut off the inference paths of $E(G_k) \cap S$ (Lines 4-8). For each iteration with j, S_0 contains the removed edges

in $E(G_{j+1})\backslash E(G_j)$, and we remove edges in $E(G_j)\backslash E(G_{j-1})$ to prevent inferring S_0. We adopt the general idea of Algorithm 1 to prevent inferring S_0, and make some modifications and extensions to original method. Firstly, when we generate E_r that contribute to the inference of S_0, we only consider edges in $E(G_j)\backslash E(G_{j-1})$ (Lines 6). Secondly, Sim_{CN} of edges in S_0 are calculated on $E(G_j)$, and edges in $E(G_k)\backslash E(G_j)$ are neglected (Line 7).

Now, we analyze the computational complexity of Algorithm 2. When tracing lineage in $E(G_j)\backslash E(G_{j-1})$, E_r and E_s contain at most $2|S|Sim_{max}^{k-j}$ and $|S|Sim_{max}^{k-j}$ edges respectively, where Sim_{max} denotes the maximal Sim_{CN} of the edges in $E(G_k)$. Algorithm 2 with naive edge removing strategy removes $\Sigma_{k=1}^{i}\Sigma_{j=k-1}^{0}|S|Sim_{max}^{k-j} \leq |S|Sim_{max}^{i}$ edges for obtaining i-inference security. Hence, the computational complexity of Algorithm 2 with naive edge removing strategy is $\mathcal{O}(|S|Sim_{max}^{i})$, where Sim_{max} is the maximal Sim_{CN} of the edges in $E(G_i)$. Similarly, the computational complexity is $\mathcal{O}(2|S|^2 Sim_{max}^{2i})$ for Algorithm 2 with heuristic edge removing strategy.

5 Experimental Evaluation

In this section, we provide extensive experiments to evaluate our methods. We use two real network datasets, **Email-1** and **LiveJ-1**, which are also used in [10]. There are 5000 vertices and 11047 edges in **Email-1** with average degree 4.42, 5000 vertices and 17847 edges in **LiveJ-1** with average degree 7.14, respectively. Table 2 lists the statistics of vertex pairs with Sim_{CN}=1,...,10 in each dataset.

Table 2. Statistics of Sim_{CN} in networks

Dataset	Common Neighbor Similarity (Sim_{CN})									
	1	2	3	4	5	6	7	8	9	10
Email-1	164144	31146	13068	7307	4558	3075	2153	1723	1285	1014
LiveJ-1	284567	21158	7510	4363	2771	2028	1575	1220	967	792

We implement four versions of Algorithm 1 for preventing one-step link inference attacks, which are *Naive*(`N`, `NA`), *LIP*(`H`, `NA`), *D/A-LIP*(`H`, `D/A`) and *S-LIP*(`H`, `S`), where `N` and `H` refer to remove edges with naive and heuristic strategy respectively, `NA` refers to no edge addition, and `D/A` and `S` refer to edge addition with `Deletion/Addition` and `Switch` strategy respectively. We also implement Algorithm 2 for avoiding cascaded link inference attacks, named as $CLIP$, which adopts heuristic edge removing strategy. All the programs are implemented in Java. The experiments are performed on a 2.33GHz Intel Core 2 Duo CPU with 4GB DRAM running the Windows XP operating system.

5.1 Performance of Link Inference Preventing v.s. Sim_{CN}, δ

We design two set of experiments to evaluate the impacts of Sim_{CN} and δ on the performance of one-step link inference preventing algorithms. Firstly, for

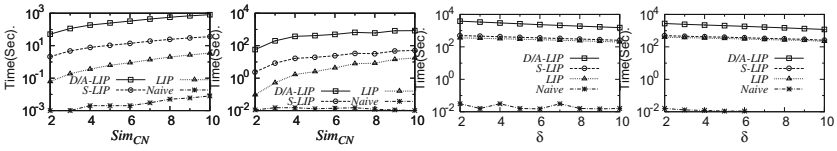
evaluating the impact of Sim_{CN}, we set δ=2 and generate S through randomly sampling 200 unconnected vertex pairs with Sim_{CN}=2,...,10 in **Email-1** and **LiveJ-1**, respectively. Secondly, for evaluating the impact of δ, we generate S with unconnected vertex pairs with $Sim_{CN} \geq 15$ and obtain 1193 and 521 sensitive pairs in **Email-1** and **LiveJ-1** respectively, and set δ=2,...,10.

5.1.1 Runtime and Information Loss

We use $\mathcal{I}.\mathcal{L}. = \frac{number\ of\ removed\ edges}{|E(G)|}$ to evaluate information loss of inference preventing algorithms. Fig.5 and Fig.6 show the results of runtime and information loss. Due to the same edge removing strategy for *LIP*, *D/A-LIP* and *S-LIP*, we only plot *LIP* in Fig.6 as representative. Generally, as δ is constant and Sim_{CN} increases, both runtime and information loss get higher as shown in Figs.5(a), 5(b) and Figs.6(a), 6(b), respectively. The reason is intuitively that preventing the inferences of sensitive edges with larger Sim_{CN} incurs more edges to remove. An exception arises in Fig.6(b) when Sim_{CN}=8, where the algorithms incurs less information loss than Sim_{CN}=7, which seems unexpected. However, on closer inspection, we find that the average IC of edges in E_r is 2.911 when Sim_{CN}=8, which is much higher than 2.294(Sim_{CN}=7) and 2.443(Sim_{CN}=9). Such observation shows that the IC value is an important factor affecting the performance of the link inference preventing algorithms. When S is constant and δ gets increased, the algorithms require lower runtime and incur less information loss, which are depicted in Figs.5(c),5(d) and Figs.6(c), 6(d), respectively, since higher δ refers to less inference secure requirement. In Fig.5, the runtime of the algorithms can be summarized as *Naive*$\ll$*LIP*$<$*S-LIP*$\ll$*D/A-LIP*. With our heuristic edge removing strategy, the information loss of *LIP* is much lower than *Naive*, as shown in Fig.6.

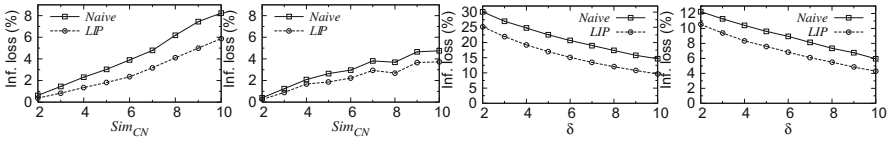
5.1.2 Data Utilities

We examine two graph structural properties, *Transitivity* and *Average Path Length* (see [13] for details), on 1000 vertices randomly sampled in vertices of S and their neighbors. We use `Change ratio` $= |P_o - P_s|/|P_o|$ to evaluate the property change ratio, where P_o and P_s refer to the property values of the original graph G and the inference secure version of G. Fig.7 and Fig.8 show the change ratios of *Transitivity* and *Average Path Length*, respectively. Generally, *S-LIP* and *D/A-LIP* are most effective in terms of preserving graph properties, and their change ratios are around 2% for **Email-1** and 7% for **LiveJ-1**. The curves of *Naive* and *LIP* in Fig.7 and Fig.8 behave similarly to the ones in Fig.6. Although considering less adding edge candidates, in practice, *S-LIP* performs as well as *D/A-LIP* but with higher efficiency. We notice that *S-LIP* and *D/A-LIP* do not preserve graph properties of **LiveJ-1** as well as **Email-1**. Such observation could be explained by the statistic data in Table 2, where except for $Sim_{CN} = 1$, the numbers of vertex pairs in **Email-1** with Sim_{CN}=2,...,10 are much higher than in **LiveJ-1**. Hence, the newly added edges in **Email-1** are with higher SS values and preserve graph properties better.



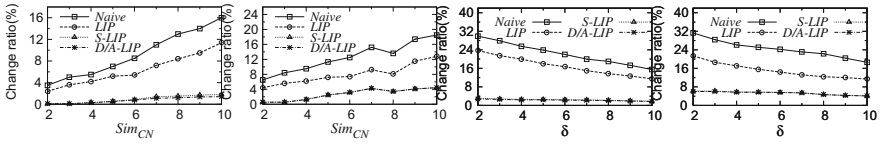
(a) Email-1 (b) LiveJ-1 (c) Email-1 (d) LiveJ-1

Fig. 5. Runtime of preventing one-step link inference



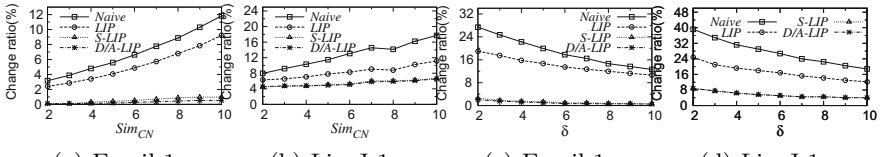
(a) Email-1 (b) LiveJ-1 (c) Email-1 (d) LiveJ-1

Fig. 6. Information loss of preventing one-step link inference



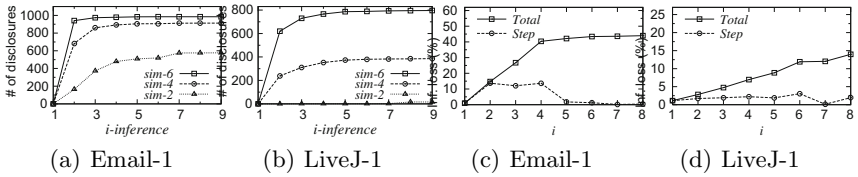
(a) Email-1 (b) LiveJ-1 (c) Email-1 (d) LiveJ-1

Fig. 7. Change ratio of Transitivity



(a) Email-1 (b) LiveJ-1 (c) Email-1 (d) LiveJ-1

Fig. 8. Change ratio of Average Path Length



(a) Email-1 (b) LiveJ-1 (c) Email-1 (d) LiveJ-1

Fig. 9. Re-identification power and information loss in $CLIP$

5.2 Re-identification Power and Information Loss in *CLIP*

Firstly, we evaluate the re-identification power of cascaded link inference. We set δ=8, and generate S through randomly sampling 1000 unconnected vertex pairs with Sim_{CN}=2,4,6 in each dataset. We count the number of disclosures in the i-inference (i=1,. . .,9) graph and show the results in Fig.9(a) and Fig.9(b). Overall, the number of disclosures gets higher as i increases and tends to stabilize after i=4. Such observation can be explained by the community theories of real social networks. The social networks consist of a large amount of communities, and the probability of there existing a link between two vertices within a community is much higher than two vertices belong to different communities. Hence, after performing one-step link inference several times, cascaded link inference has disclosed the sensitive edges connecting vertices within a community rather than other ones bridging different communities, such that i-inference (i>4) do not lead to an observable increase in disclosures in any dataset. Specially, for sensitive pairs with Sim_{CN}=6, 4-inference causes 100% and 80% of these edges disclosed in **Email-1** and **LiveJ-1**, respectively. Hence, cascaded link inference is indeed a privacy threat for sensitive relationships in real networks.

Secondly, we examine the information loss of *CLIP*. We set δ=3, and generate S through randomly sampling 200 unconnected vertex pairs with Sim_{CN}=4 in each dataset. *Total* and *Step* in Figs. 9(c), 9(d) refer to the information loss for obtaining i-inference secure graph and transforming a (i−1)-inference secure graph into i-inference secure, respectively. As depicted in Fig.9(c) and Fig.9(d), *Total* gets higher as i increases from 1 to 8 meanwhile *Step* behaves unsteadily. An interesting observation of *Step*, namely *Sharp Drop*, arises in **Email-1** when i=5 and in **LiveJ-1** when i=7, where *Step* decreases sharply. Such observation could be interpreted as follows. Before *Sharp Drop* arises, edges in G are gradually removed by *CLIP*. When most of the inference paths of S have been cut off, *CLIP* would remove much less edges in G than before to obtain an i-inference secure graph, which is a *Sharp Drop*. However, inference paths could be reconstructed by cascaded link inference and *Step* may increase after *Sharp Drop*, as shown in Fig.9(d) when i=8. Overall, studying the inference paths of sensitive links is the key to avoid cascaded link inference attacks.

6 Conclusion

In this paper, we discuss an important privacy problem in social networks, namely *link inference attacks*. We formalize *inference security* and develop a general framework for obtaining inference secure graphs. We propose efficient algorithms for preventing *one-step link inference* attacks and *cascaded link inference* attacks. An extensive empirical study on real datasets indicates that link inference attacks are real in practice, and our methods perform well in terms of privacy protection and efficiency meanwhile maintaining graph properties.

References

1. Liu, K., Terzi, E.: Towards identity anonymization on graphs. In: SIGMOD, pp. 93–106 (2008)
2. Zhou, B., Pei, J.: Preserving privacy in social networks against neighborhood attacks. In: ICDE, pp. 506–515 (2008)
3. Hay, M., Miklau, G., Jensen, D., Towsley, D.: Resisting structural re-identification in anonymized social networks. In: VLDB, pp. 102–114 (2008)
4. Campan, A., Truta, T.M.: A clustering approach for data and structural anonymity in social networks. In: PinKDD (2008)
5. Zou, L., Chen, L., Ozsu, M.T.: K-automorphism: A general framework for privacy preserving network publication. VLDB Endowment 2(1), 946–957 (2009)
6. Zheleva, E., Getoor, L.: Preserving the Privacy of Sensitive Relationships in Graph Data. In: Bonchi, F., Malin, B., Saygın, Y. (eds.) PInKDD 2007. LNCS, vol. 4890, pp. 153–171. Springer, Heidelberg (2008)
7. Ying, X., Wu, X.: Randomizing social networks: a spectrum preserving approach. In: SDM, pp. 739–750 (2008)
8. Cormode, G., Srivastava, D., Yu, T., Zhang, Q.: Anonymizing bipartite graph data using safe groupings. VLDB Endowment 1(1), 833–844 (2008)
9. Bhagat, S., Cormode, G., Krishnamurthy, B., Srivastava, D.: Class-based graph anonymization for social network data. VLDB Endowment 2(1), 766–777 (2009)
10. Cheng, J., Fu, A.W.-C., Liu, J.: K-Isomorphism: Privacy preserving network publication against structural attacks. In: SIGMOD, pp. 459–470 (2010)
11. Yuan, M., Chen, L., Yu, P.S.: Personalized privacy protection in social networks. VLDB Endowment 4(2), 141–150 (2010)
12. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC, pp. 151–158 (1971)
13. Costa, L.D.F., Rodrigues, F.A., Travieso, G., Boas, P.R.V.: Characterization of complex networks: A Survey of measurements. Advances in Physics 56(1), 167–242 (2007)