THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

TRD3/12

Sensitive Label Privacy Protection on Social Network Data

Yi Song, Panagiotis Karras, Qian Xiao and Stephane Bressan

March 2012

T e c h n i c a l R e p o r t

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

OOI Beng Chin
Dean of School

Sensitive Label Privacy Protection on Social Network Data

Yi Song*, Panagiotis Karras◊, Qian Xiao*, and Stéphane Bressan*

*School of Computing
National University of Singapore
{songyi, xiaoqian, steph}@nus.edu.sg
◊Rutgers Business School
Rutgers University
karras@business.rutgers.edu

Abstract. The publication of social network data presents opportunities for data mining and analytics for strategic public, commercial and academic applications. Yet the publication of social network data entails a privacy threat for their users. Sensitive information should be protected. The challenge is to devise methods to publish these data in a form that affords utility without compromising privacy. Previous research has proposed various privacy models with the corresponding protection mechanisms. These early privacy models are mostly concerned with identity and link disclosure. The social networks are modeled as graphs in which users are nodes. The threat definitions and the protection mechanisms leverage structural properties of the graph. This paper is motivated by the recognition of the need for a finer grain and more personalized privacy. We propose a privacy protection scheme that not only prevents the disclosure of identity of users but also the disclosure of selected features in users' profiles. An individual user can select which features of her profile she wishes to conceal. The social networks are modeled as graphs in which users are nodes and features are labels. Labels are denoted either as sensitive or as non-sensitive. We treat node labels *both* as background knowledge an adversary may possess, *and* as sensitive information that has to be protected. We present privacy protection algorithms that allow for graph data to be published in a form such that an adversary who possesses information about a node's neighborhood cannot safely infer its identity and its sensitive labels. To this aim, the algorithms transform the original graph into a graph in which nodes are sufficiently indistinguishable. The algorithms are designed to do so while losing as little information and while preserving as much utility as possible. We evaluate empirically the extent to which the algorithms preserve the original graph's structure and properties. We show that our solution is effective, efficient and scalable while offering stronger privacy guarantees than those in previous research.

1 Introduction

Online social networks create opportunities for users to share information, keep in touch with peers, and maintain professional circles. However such data also

arouses the interest of professionals and researchers from various fields including epidemiology, marketing and sociology among many other examples. Thus, making the social network data available can serve public, commercial and academic interests.

Yet the publication of social network data entails a privacy threat for their users. Sensitive information about users of the social networks should be protected. The challenge is to devise methods to publish social network data in a form that affords utility without compromising privacy.

Previous research has proposed various privacy models with the corresponding protection mechanisms that prevent both inadvertent private information leakage and attacks by malicious adversaries. These early privacy models are mostly concerned with identity and link disclosure. The social networks are modeled as graphs in which users are nodes and the social connections are edges. The threat definitions and the protection mechanisms leverage structural properties of the graph. This paper is motivated by the recognition of the need for a finer grain and more personalized privacy.

Users entrust social networks such as Facebook and LinkedIn with a wealth of personal information such as their age, address, current location or political orientation. We refer to these details and messages as features in the user's profile. We propose a privacy protection scheme that not only prevents the disclosure of identity of users but also the disclosure of selected features in users' profiles. An individual user can select which features of her profile she wishes to conceal.

The social networks are modeled as graphs in which users are nodes and features are labels[1]. Labels are denoted either as sensitive or as non-sensitive. Figure 1 is a labeled graph representing a small subset of such a social network. Each node in the graph represents a user, and the edge between two nodes represents the fact that the two persons are friends. Labels annotated to the nodes show the locations of users. Each letter represents a city name as a label for each node. Some individuals do not mind their residence being known by the others, but some do, for various reasons. In such case, the privacy of their labels should be protected at data release. Therefore the locations are either sensitive (labels are in italic in Figure 1) or non-sensitive. Table 1 records the neighborhood information for each node that has sensitive label.

The privacy issue arises from the disclosure of sensitive labels. One might suggest that such labels should be simply deleted. Still, such a solution would present an incomplete view of the network and may hide interesting statistical information that does not threaten privacy. A more sophisticated approach consists in releasing information about sensitive labels, while ensuring that the identities of users are protected from privacy threats. We consider such threats as *neighborhood attack*, in which an adversary finds out sensitive information based on *prior knowledge* of the number of neighbors of a target node and the labels of these neighbors. In the example, if an adversary knows that a user has

[1] Although modeling features in the profile as attribute-value pairs would be closer to the actual social network structure, it is without loss of generality that we consider atomic labels.

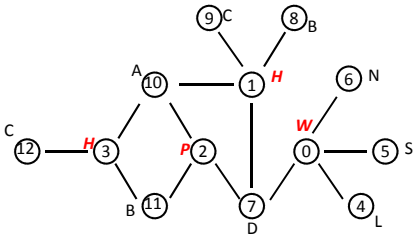


Fig. 1. Example of the labeled graph representing a social network

vertexID	Sensitive Label	Neighbor Labels
0	*W*	D, L, S, N
1	*H*	A, B, C, D
2	*P*	A, B, D
3	*H*	A, B, C

W: Warsaw, H: Helsinki, P: Prague
D: Dublin, S:Stockholm, N: Nice, A: Alexandria,
B: Berlin, C: Copenhagen, L: Lisbon

Table 1. Neighbor labels

three friends and that these friends are in A (Alexandria), B (Berlin) and C (Copenhagen), respectively, then she can infer that the user is in H (Helsinki).

We present privacy protection algorithms that allow for graph data to be published in a form such that an adversary cannot safely infer the identity and sensitive labels of users. We consider the case in which the adversary possesses both structural knowledge and label information.

The algorithms that we propose transform the original graph into a graph in which any node with a sensitive label is indistinguishable from at least $\ell-1$ other nodes. The probability to infer that any node has a certain sensitive label (we call such nodes *sensitive nodes*) is no larger than $1/\ell$. For this purpose we design ℓ-diversity-like model, where we treat node labels as *both* part of an adversary's background knowledge *and* as sensitive information that has to be protected.

The algorithms are designed to provide privacy protection while losing as little information and while preserving as much utility as possible. In view of the tradeoff between data privacy and utility [19], we evaluate empirically the extent to which the algorithms preserve the original graph's structure and properties such as density, degree distribution and clustering coefficient. We show that our solution is effective, efficient and scalable while offering stronger privacy guarantees than those in previous research, and that our algorithms scale well as data size grows.

The rest of the paper is organized as follows. Section 2 reviews previous works in the area. Then we define our problem in Section 3 and propose solutions in Section 4. Experiments and result analysis are described in Section 5. We conclude this work in Section 6.

2 Related Work

The first necessary anonymization technique in both the contexts of micro- and network data consists in removing identification. This nave technique has quickly been recognized as failing to protect privacy. For microdata, Sweeney et al. propose k-anonymity [20] to circumvent possible identity disclosure in naively anonymized microdata. ℓ-diversity is proposed in [16] in order to further prevent attribute disclosure.

Similarly for network data, Backstrom et al., in [2], show that naive anonymization is insufficient as the structure of the released graph may reveal the identity of the individuals corresponding to the nodes. Hay et al. [12] emphasize this problem and quantify the risk of re-identification by adversaries with external information that is formalized into structural queries (node refinement queries, subgraph knowledge queries). Recognizing the problem, several works [6, 14, 21, 23–25, 27, 31] propose techniques that can be applied to the naive anonymized graph, further modifying the graph in order to provide certain privacy guarantee. Liu et al. [14] suggest to perturb the graph by adding edges to render graph's degree sequence to be *k-anonymity*, that is, for each node, there are at least $k-1$ other nodes having the same degree. The published graph is *k-degree-anonymous*. Wu et al. [23], having the same consideration on re-identification risk, suggest an approach to modify the graph to be *k-symmetry*, which guarantees that each node in the graph is structurally indistinguishable from other $k-1$ nodes. Similarly, Zou et al. [31] propose to modify the graph to be *k-automorphic* by adding nodes, deleting edges and adding edges. The graph after perturbation is able to resist any kind of structural attack. Instead of preventing re-identification attacks, Zhang et al. [27] focus on the attack of link disclosure and raise the concept of τ*-confidence* which limits the adversary's confidence of edge existence between two nodes to a threshold. To achieve such constraints they use edge swapping and deletion operations. Cheng et al. [6] try to give a comprehensive solution to provide protection against all kinds of attacks, including identity disclosure, attributes disclosure and link disclosure. They propose *k-isomorphism*, a property with which the graph consists of k pairwise isomorphic subgraphs. However, the partitioning of the graph brings along significant distortion resulting in severe change of graph properties. They also perturb the graphs by adding nodes, adding edges and deleting edges. Ying et al. in [25] study on the randomization approaches for link disclosure, while in [24], they study the effects of random edge addition, deletion and switching on graph spectrum properties. They quantify the anonymity level based on a-posteriori belief, which Bonchi et al. [11] believe is not adequate. Bonchi et al. suggest using entropy-based quantification, a global quantification. Based on such measurement, they perturb the graphs randomly. Tai et al. [21] propose k^2-degree anonymity to protect against re-identification by adversaries with background knowledge of degrees of two nodes connected by an edge. Campan et al. [4, 5] and Cormode et al. [8] handle the problem of re-identification via clustering and generalization (e.g. making super nodes and super edges). Most of these works focus on graph structure, rather than labels on nodes.

Some works are based on graph models other than simple graph. Zheleva and Getoor [28] consider graphs containing both sensitive and non-sensitive edges. They assume that the adversaries predict sensitive edges based on the observed non-sensitive edges. Liu et al. [15], Sudipto et al. [9] and Li et al. [13] all take edge weights into considerations for privacy preservation. Bhagat et al. [3] anonymize the labeled bipartite graphs by grouping similar nodes or edges.

To our knowledge, Zhou and Pei [29, 30] and Yuan et al. [26] were the first to consider modeling social networks as labeled graphs, similarly to what we consider in this paper. To prevent re-identification attacks by adversaries with immediate neighborhood structural knowledge, Zhou and Pei [29] propose a method that groups nodes and anonymizes the neighborhoods of nodes in the same group by generalizing node labels and adding edges. They enforce a *k-anonymity* privacy constraint on the graph, each node of which is guaranteed to have the same immediate neighborhood structure with other $k-1$ nodes. In [30], they improve the privacy guarantee provided by *k-anonymity* with the idea of ℓ-diversity, to protect labels on nodes as well. Yuan et al. [26] try to be more practical by considering users' different privacy concerns. They divide privacy requirements into three levels, and suggest methods to generalize labels and modify structure corresponding to every privacy demand. Nevertheless, neither Zhou and Pei, nor Yuan et al. consider labels as a part of the background knowledge. However, in case adversaries hold label information, the methods of [29, 30, 26] cannot achieve the same privacy guarantee. Moreover, as with the context of microdata, a graph that satisfies a k-anonymity privacy guarantee may still leak sensitive information regarding its labels [16].

Differential privacy [10] is a popular privacy paradigm. It finds applications in interactive query systems. A graph satisfying some version of differential privacy would provide identity protection, but still suffer from the same sensitive label privacy problem that we identify; the way to model the difference between sensitive and non-sensitive labels in that model is not straightforward. Thus, we propose on an ℓ-diversity-like model, while we treat node labels *both* as part of an adversary's background knowledge, *and* as sensitive information that has to be protected. Furthermore, recent research has established that, even in the case of microdata, differential privacy does not provide the kind of sensitive value protection that ℓ-diversity can afford [7].

3 Problem Definition

We model a network as $G(V, E, L^s, L, \Gamma)$, where V is a set of nodes, E is s set of edges, L^s is a set of sensitive labels, and L is a set of non-sensitive labels. Γ maps nodes to their labels, $\Gamma : V \rightarrow L^s \cup L$. Then we propose a privacy model, *ℓ-sensitive-label-diversity*; in this model, we treat node labels *both* as part of an adversary's background knowledge, *and* as sensitive information that has to be protected. These concepts are clarified by the following definitions:

Definition 1. *The **neighborhood information** of node v comprises the degree of v and the labels of v's neighbors.*

Definition 2. *(ℓ-**sensitive-label-diversity**) For each node v that associates with a sensitive label, there must be at least ℓ − 1 other nodes with the same neighborhood information, but attached with different sensitive labels.*

In Example 1, nodes 0, 1, 2, and 3 have sensitive labels. The neighborhood information of node 0, includes its degree, which is 4, and the labels on nodes 4, 5, 6, and 7, which are L, S, N, and D, respectively. For node 2, the neighborhood information includes degree 3 and the labels on nodes 7, 10, and 11, which are D, A, and B. The graph in Figure 2 satisfies 2-*sensitive-label-diversity*; that is because, in this graph, nodes 0 and 3 are indistinguishable, having six neighbors with label A, B, {C,L}, D, S, N separately; likewise, nodes 1 and 2 are indistinguishable, as they both have four neighbors with labels A, B, C, D separately (Table 2).

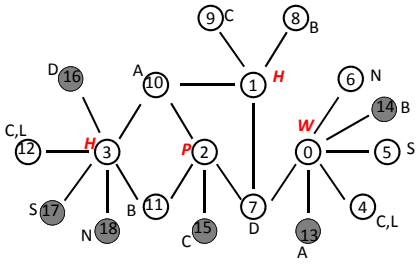


Fig. 2. Privacy-attaining network example

vertexID	Sensitive Label	Neighbor Labels
0	*W*	A, B, {C,L}, D, N, S
3	*H*	A, B, {C,L}, D, N, S
1	*P*	A, B, C, D
2	*H*	A, B, C, D

Table 2. neighbor labels

4 Algorithm

The main objective of the algorithms that we propose is to make suitable grouping of nodes, and appropriate modification of neighbors' labels of nodes of each group to satisfy the *l-sensitive-label-diversity* requirement.

We want to group nodes with as similar neighborhood information as possible so that we can change as few labels as possible and add as few noisy nodes as possible. The three algorithms that we propose modify labels, add edges and add noisy nodes.

We first devise two algorithms, Direct Noisy Node Algorithm (DNN) and Indirect Noisy Node Algorithm (INN) that sort nodes by degree and compare neighborhood information of nodes with similar degree. Considering that nodes with similar degree are more likely to have similar neighborhood information,

we expect to reduce computation by avoiding recomputing similarity for every pair of nodes. The two algorithms differ in the order in which they add nodes.

We also propose a third algorithm, Global-similarity-based Indirect Noise Node (GINN), that does not attempt to heuristically prune the similarity computation.

4.1 Algorithm DNN

Our first algorithm,(DNN), starts out by sorting V by degree. nodes with the same degree, which belong to the same *segment*, are grouped by the similarity of their neighborhood labels. For two nodes, v_1 with neighborhood label set (LS_{v_1}), and v_2 with neighborhood label set (LS_{v_2}), we calculate neighborhood label similarity (NLS) as follows:

$$NLS(v_1, v_2) = \frac{|LS_{v_1} \cap LS_{v_2}|}{|LS_{v_1} \cup LS_{v_2}|} \tag{1}$$

Larger value indicates larger similarity of the two neighborhoods.

Algorithm 1: Direct Noisy Node Algorithm (DNN)

Input: graph $G(V, E, L, L^s)$, parameter l;
Result: Modified Graph G'

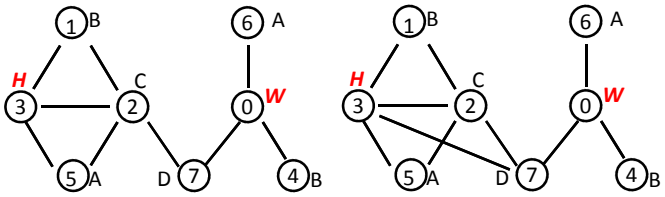
Sort degree sequence;
for *each* segment **do**
  if *the number of sensitive nodes* $\geq l$ **then**
    compute pairwise node similarities;
    group $\mathcal{G} \leftarrow v_1, v_2$ with $Max_{similarity}$;
    while $|\mathcal{G}| < l$ **do**
      group $\mathcal{G} \leftarrow$ next v;
    Modify neighbors of $\mathcal{G}$;
while $V_{left} > 0$ **do**
  if $|V_{left}| \geq l$ **then**
    compute pairwise node similarities;
    group $\mathcal{G} \leftarrow v_1, v_2$ with $Max_{similarity}$;
    while $|\mathcal{G}| < l$ **do**
      group $\mathcal{G} \leftarrow$ next v;
    Modify neighbors of $\mathcal{G}$;
  else if $|V_{left}| < l$ **then**
    for *each* $v \in V_{left}$ **do**
      $similarity(v, \mathcal{G}s)$;
      $\mathcal{G}_{Max\_similarity} \leftarrow v$;
    Modify neighbors of $\mathcal{G}_{Max\_similarity}$;
Return $G'(V', E', L')$;

In the *DNN* algorithm, after computing the similarities between pairs of nodes in the same segment, two nodes (x, y) with the maximal similarity value are chosen to be a group together. Other nodes are then merged to this group according to their similarities with either x or y, till this group obtains nodes with ℓ different sensitive labels. After one group is finalized, we calculate the number of nodes left in the segment to check whether there are enough nodes for forming another group. A grouping process may thus be carried out again. After all segments have been tackled in this way, we deal with nodes left over from different segments. Pairwise similarities are again computed among them, and a similar method is adopted to build groups attaining *ℓ-sensitive-label-diversity*. Nodes that are left after all the groupings, are then merged into existing groups according to the similarities between each such node and existing group members.

After we have formed these groups, we need to ensure that each group's members are indistinguishable in terms of the neighborhood information. Thus, neighborhood labels are modified right after every grouping operation, so that labels of nodes can be accordingly updated immediately for the next grouping operation. This modification process ensures that all nodes in a group have the same *neighborhood information*. The objective is achieved by a series of modification operations.

To modify graph with as low information loss as possible, we devise three modification operations: *label union*, *edge insertion* and *noise node addition*. Label union and edge insertion among nearby nodes are preferred to node addition, as they incur less alteration to the overall graph structure.

Edge insertion is to complement for both a missing label and insufficient degree value. A node is linked to an existing nearby (two-hop away) node with that label. Take the graph in Figure 3(a) as an example. Nodes 0 and 3 form one group. To make them have the same neighborhood information, node 0 needs a neighbor with label C, and since node 2 with label C is two hops away from node 0, an edge is inserted between node 0 and 2. Similarly, there's an edge inserted between node 3 and 7.

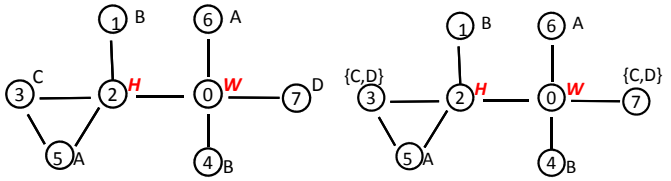


(a) Before edge insertion (b) Edge insertion

Fig. 3. Edge Insertion

Label union adds the missing label values by creating super-values shared among labels of nodes. The labels of two or more nodes coalesce their values to a single super-label value, being the union of their values. This approach maintains data integrity, in the sense that the true label of each node is included among

the values of its label super-value. Figure 4 shows an example of label union. node 0 and 2 form one group. In order for them to have the same neighborhood information, label union is happened to labels of node 3 and 7.



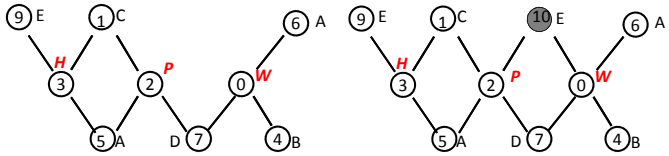
(a) Before label union (b) Label union

Fig. 4. Label Union

After such edge insertion and label union operations, if there are nodes in a group still having different neighborhood information, noise nodes with non-sensitive labels are added into the graph, so as to render the nodes in group indistinguishable in terms of their neighbors' labels. We consider the unification of two nodes' neighborhood labels as an example. One node may need a noisy node to be added as its immediate neighbor since it does not have a neighbor with that certain label that the other node has; such a label on the other node may not be modifiable, as its is already connected to another sensitive node, which prevents the re-modification on existing modified groups.

4.2 Algorithm INN

Our second Algorithm, Indirect Noisy Node Algorithm (*INN*), differs from the first algorithm in that it does *not* modify the neighbors of every group *right after* the grouping action. Only after it has detected enough nodes for one group, edge insertion and label union operations are performed; thereafter, noise node addition operation that is expected to make the nodes inside each group satisfy ℓ-sensitive-label-diversity are *recorded*, but *not* performed right away. Only after *all* the preliminary grouping operations are performed, the algorithm proceeds to process the *expected node addition* operation at the final step. Then, if two nodes are expected to have the same labels of neighbors and are within two hops (having common neighbors), only one node is added. In other words, we merge some noisy nodes with the same label, thus resulting in fewer noisy nodes. Take the graph in Figure 5(a) as an example. If nodes 0, 2, 3 form one group, then both nodes 0 and 2 need a neighbor with label E because node 3 has a neighbor with label E. Since nodes 0 and 2 are within two hops, having common neighbor node with label D, we can only add node 10 associated with label E. Similarly, a noisy node with label B is added and connected to both nodes 2 and 3. In addition, because we make restrict on the distance of the two nodes that need that noisy node, we limit the change of distance between nodes. Furthermore, a second difference between INN and DNN is that the INN Algorithm is trying to form groups out of nodes with *similar* degrees in the first step, rather than expecting



(a) Before addition (b) Addition of node 10

Fig. 5. Noisy Node Addition

them to have strictly the same degrees. In Line 5, pairwise node similarities are calculated among the nodes from *two neighbor segments*. Such segments have relatively similar degrees, since segments are sorted by degree.

Algorithm 2: Indirect Noisy Node Algorithm (INN)

Input: graph $G(V, E, L, L^s)$, parameter l;
Result: Modified Graph G'

1 *Sort* degree sequence;
2 **for** *each segment* **do**
3 $\quad \mathcal{S}et \leftarrow segment[i-1] \bigcup segment[i]$;
4 $\quad$ **if** *the number of nodes with sensitive label in* $\mathcal{S}et \geq l$ **then**
5 $\quad\quad$ compute pairwise node similarities in $\mathcal{S}et$;
6 $\quad\quad$ group $\mathcal{G} \leftarrow v_1, v_2$ with $Max_{similarity}$;
7 $\quad\quad$ **while** $|\mathcal{G}| < l$ **do**
8 $\quad\quad\quad$ group $\mathcal{G} \leftarrow$ next v;
9 $\quad\quad$ Modify neighbors without actually adding noisy nodes;
10 **while** $V_{left} > 0$ **do**
11 $\quad$ **if** $|V_{left}| \geq l$ **then**
12 $\quad\quad$ compute pairwise node similarities;
13 $\quad\quad$ group $\mathcal{G} \leftarrow v_1, v_2$ with $Max_{similarity}$;
14 $\quad\quad$ **while** $|\mathcal{G}| < l$ **do**
15 $\quad\quad\quad$ group $\mathcal{G} \leftarrow$ next v;
16 $\quad\quad$ Combine labels;
17 $\quad$ **else if** $|V_{left}| < l$ **then**
18 $\quad\quad$ **for** *each* $v \in V_{left}$ **do**
19 $\quad\quad\quad$ $\mathcal{G}_{last\_grouped} \leftarrow v$;
20 $\quad$ Modify neighbors without actually adding noisy nodes;
21 Add expected noisy nodes;
22 **Return** $G'(V', E', L')$;

Last, a third difference is that, in the INN Algorithm, nodes left after cross-segments grouping are all merged to the final group, instead of merged to existing groups according to the similarities between each node and existing group members. The main techniques adopted for label modification are the same as algorithm *DNN*. Again, inserting edges to nearby nodes and label union are preferred to node addition; the latter is done only after all the nodes have been grouped, so as to reduce the number of noisy nodes inserted.

4.3 Algorithm GINN

Algorithms *DNN* and *INN* both start out with degree sorting, and then compute label similarities among nodes with same or similar degrees. By contrast, our third algorithm, Global-Similarity-based Indirect Noisy Node Algorithm (*GINN*), computes dissimilarity based on *both degree and labels*. In effect, no priority is given to degree and there is no sorting step. During group formation, all nodes that have not yet been grouped are taken into consideration, in clustering-like fashion. In the first run, two nodes with the minimum dissimilarity are grouped together. Their neighbor labels are modified to be the same immediately so that nodes in one group always have the same neighbor labels. The desirable noisy nodes are again recorded, as in INN, while they are not added at this stage. Then, nodes having the minimum dissimilarity with any node in the group are clustered into the group till the group has ℓ nodes with different sensitive labels. Thereafter, the algorithm proceeds to creating the next group. If there are fewer than ℓ nodes left after the last group's formation, these remainder nodes are clustered into existing groups according to the similarities between nodes and groups. As in Algorithm *INN*, after all the group formation operations, the algorithm proceeds to process the expected node addition in the final step. For nodes expected to have the same labels which are within two hops (having common neighbors), from each, only one node will be added, same as the expected node addition process in Algorithm *INN*.

Algorithm 3: Global-Similarity-based Indirect Noisy Node Algorithm (*GINN*)

Input: graph $G(V, E, L, L^s)$, parameter l, ω;
Result: Modified Graph G'

while $V_{left} > 0$ **do**
  if $|V_{left}| \geq l$ **then**
    compute pairwise node similarities;
    group $\mathcal{G} \leftarrow v_1, v_2$ with $Max_{similarity}$;
    Modify neighbors of $\mathcal{G}$;
    while $|\mathcal{G}| < l$ **do**
      $dissimilarity(V_{left}, \mathcal{G})$;
      group $\mathcal{G} \leftarrow v$ with $Max_{similarity}$;
      Modify neighbors of $\mathcal{G}$ without actually adding noisy nodes ;
  else if $|V_{left}| < l$ **then**
    for *each* $v \in V_{left}$ **do**
      $similarity(v, \mathcal{G}s)$;
      $\mathcal{G}_{Max\_similarity} \leftarrow v$;
    Modify neighbors of $\mathcal{G}_{Max\_similarity}$ without actually adding noisy nodes;
Add expected noisy nodes;
Return $G'(V', E', L')$;

Algorithm 1 shows pseudocode for Algorithm *DNN*. Algorithm 2 shows pseudocode for Algorithm *INN*. Algorithm 3 shows pseudocode for Algorithm *GINN*.

5 Experimental Evaluation

In this section we evaluate our approaches using both synthetic and real data sets. All of the approaches have been implemented in Python. The experiments are conducted on an Intel core, 2Quad CPU, 2.83GHz machine with 4GB of main memory running Windows 7 Operating System.

5.1 Data Sets

The first dataset that we use was proposed in [1]. This data set is a network of hyperlinks between weblogs on US politics. It was recorded in 2005 by Adamic and Glance. It consists of a total of 1490 webblogs (as nodes) and 19090 hyperlinks between the weblogs (as edges) posted by 47 websites. These political blogs can be found in online weblog directories such as "Blogaram", "BlogCatalog", "LeftyDirectory","CampaignLine". We view these directories as the labels of nodes. A blog listed in more than one directory has a coumpound label, for instance "Blogarama, BlogCatalog". We assume that 2 of the 184 labels are sensitive, namely, "BlogCatalog" and "CampaignLine".

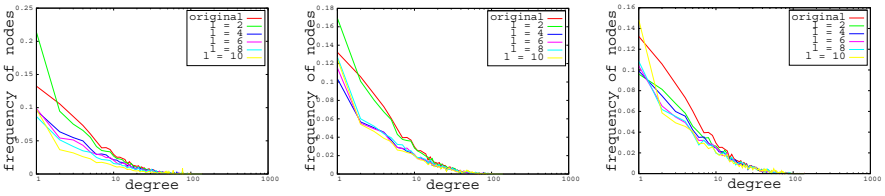
The second data set that we use is generated from the Facebook dataset proposed in [17]. Nodes represent users. Edges represent friendship relationships. We sample a subgraph of the dataset with 6339 nodes and 35452 edges. As the dataset does not contain labels, we randomly assign labels "A", "B", "C", ..., "J", "a", "b", "c", ..., "j". We assume that labels "a", "b", "c", ..., "j" are sensitive. Ten percent of the nodes are assigned sensitive labels.

The third data set that we use is a family of synthetic graphs with varying number of nodes. We generate graphs with power law property (using *Networkx* [18]). We vary the number of nodes from 100 to 10000. We randomly assign labels "A", "B", "C", ..., "J", "a", "b", "c", ..., "j". We assume that labels "a", "b", "c", ..., "j" are sensitive. Ten percent of the nodes are assigned sensitive labels.

The first and second datasets are used for the evaluation of effectiveness (utility and information loss). The third data set is used to measure runtime and scalability (running time).

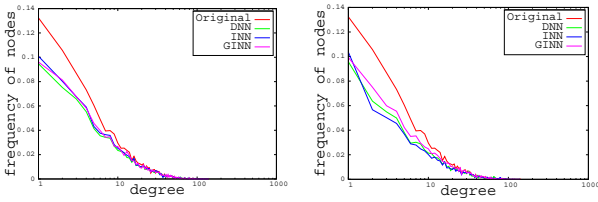
5.2 Data Utility

In this subsection, we compare the data utilities we preserve from the original graphs, in view of measurements on degree distribution, label distribution, degree centrality [22], clustering coefficient, average path length, graph density, and radius. We show the number of the noisy nodes and edges needed for each approach. Degree centrality of node v is defined as the number of nodes adjacent to it in a graph. Clustering coefficient includes global clustering coefficient and local clustering coefficient. Global clustering coefficient is defined as: $\frac{3*\triangle}{\Lambda}$, where $\triangle$ is the number of triangles in a graph, and Λ is the number of connected triples. Local clustering coefficient is defined as $\mathcal{C}_i = \frac{|e_{jk}|}{k_i*(k_i-1)}$, where $|e_{jk}|$ is the number

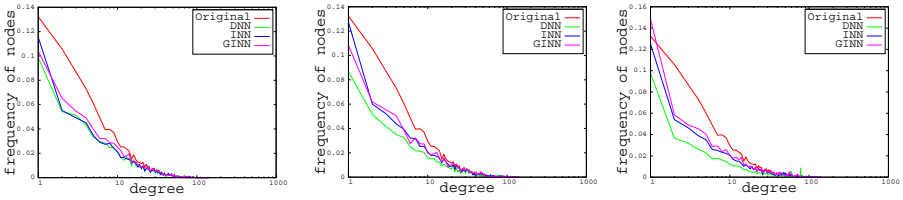


(a) DNN (b) INN (c) GINN

Fig. 6. Facebook Graph Degree Distribution



(a) $l = 2$ (b) $l = 4$

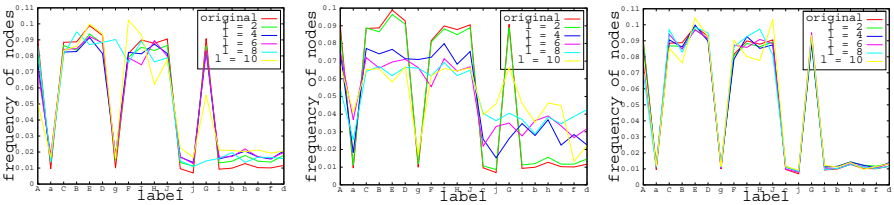


(c) $l = 6$ (d) $l = 8$ (e) $l = 10$

Fig. 7. Facebook Graph Degree Distribution with Different Value of l

of edges between the neighbors of v_i while k_i is the number of v_i's neighbors. We measure the average of local clustering coefficient of a graph.

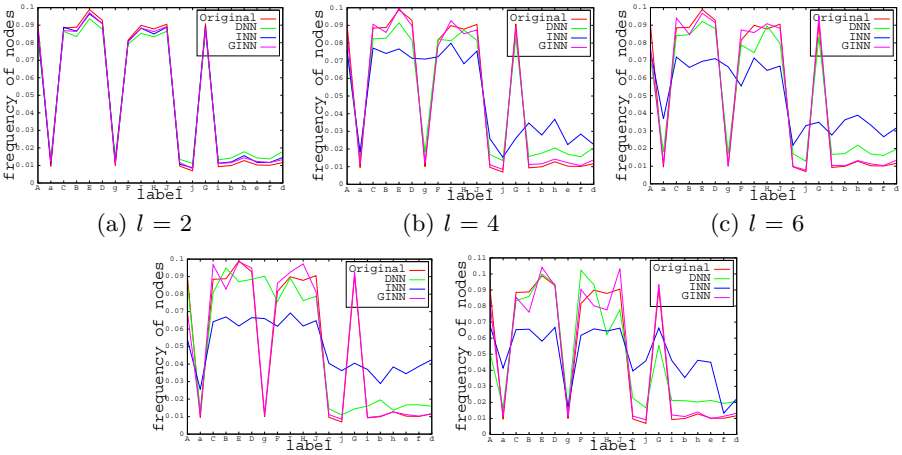
Figure 6 and 7 show the degree distribution of the Facebook graph both before and after modification. X-axis shows the value of degree, and y-axis shows the number of nodes with a certain degree. Each subfigure in Figure 6 shows degree distributions of graphs modified by one algorithm. Each subfigure in Figure 7 shows the comparisons among three algorithms under different values of parameter l. We can see that the degree distributions of the modified graphs resemble the original ones well, especially when l is small.



(a) *DNN* (b) *INN* (c) *GINN*

Fig. 8. Facebook Graph Label Distribution

Figure 8 and 9 show the label distribution of the Facebook graph both before and after modification. X-axis shows the labels, and y-axis shows the number of nodes with a certain label. Each subfigure in Figure 8 shows label distributions of graphs modified by one algorithm. Each subfigure in Figure 9 shows the comparisons among three algorithms under different values of parameter l. We can see that the label distributions of the modified graphs by Algorithm *GINN* resemble the original ones quite well. Figure 10 shows the degree and

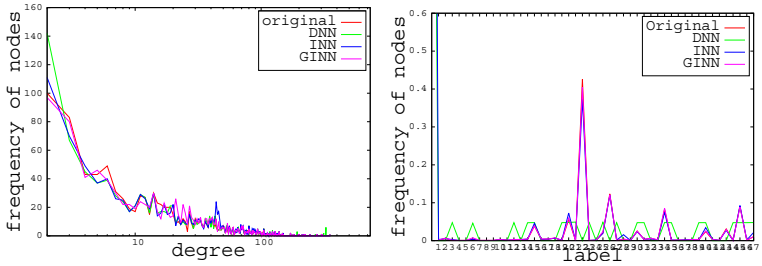


(a) $l = 2$ (b) $l = 4$ (c) $l = 6$

(d) $l = 8$ (e) $l = 10$

Fig. 9. Facebook Graph Label Distribution with Different Value of l

label distribution of the Polblog graph before and after modification. The results similarly reflect that degree distribution and label distribution of graph modified by Algorithm *GINN* best resemble the original ones. Figure 11 shows

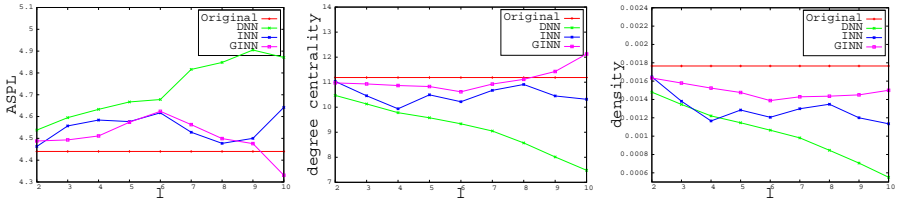


(a) Degree distribution (b) Label distribution

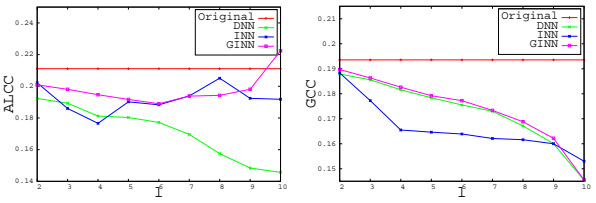
Fig. 10. Polblog Data Degree and Label Distribution

the properties of the the Facebook graph before and after modification. Figure 11(d)(e) shows the global and local clustering coefficient of the Facebook graph before and after modification. Measurements on both metrics show that Algorithm *GINN* remains the clustering coefficients of graphs better than the other

two in most cases. The differences caused by *GINN* are at most ten percentage of the original value. Figure 11(a) shows the average path length of the original graph, and the modified graphs. X-axis represents the value parameter l. Y-axis represents the average shortest path length. We can see that *GINN* preserves average shortest path length quite well. As l increases, the average shortest path length increases in general. Figure 11(b) shows degree centrality of the original graph and the modified graphs. Figure 11(c) shows graph density of the original graphs, and the modified graphs. The measurements on these two metrics show that Algorithm *GINN* produces much better results than *DNN* and *INN* do. Algorithm *GINN* can preserve these graph properties to a higher degree.



(a) Average shortest path length (b) Degree centrality (c) Density

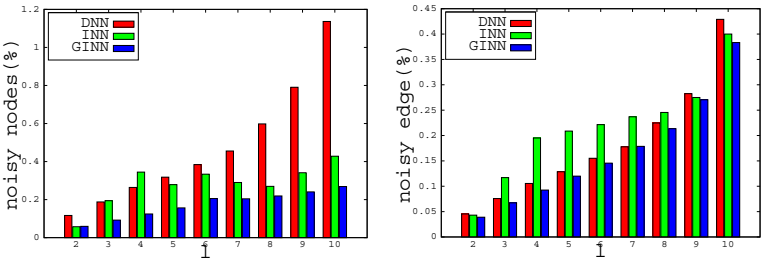


(d) Average local clustering coefficient (e) Global clustering coefficient

Fig. 11. Facebook Graph Structural Properties

Figure 12 shows the number of the noisy nodes and edges added. The amount of noisy nodes and edges suggests the inevitable impairment of data utility after modification. As expected, the number of noisy nodes and noisy edges increases as the parameter l increases in general. Still, comparing the three algorithm, we can see that Algorithm *GINN* always brings the fewest noisy nodes and edges, and thus the least impairment to the graphs structure. We also measure the radius and diameter of the graphs before and after anonymization. We find that in some cases the radius remains the same, while in other cases they increase by one hop. Similarly, in some cases, the diameter remains the same, and in more cases, they increase by one or two hops.

To sum up, these measurements on degree, average shortest path length, density, radius and clustering coefficient show that the graph structure properties are preserved to a large extent. The strong resemblance of the label distributions in most cases indicates that the label information, another aspect of graph information, is well maintained. They suggest as well that Algorithm *GINN* does



(a) Number of noisy node (b) Number of noisy edge

Fig. 12. Noisy Nodes Noisy Edges

preserve graph properties better than the other two while these three algorithms achieve the same privacy constraint.

5.3 Information Loss

In view of utility of released data, we aim to keep information loss low. Information loss in this case contains both structure information loss and label information loss. We measure the loss in the following way: for any node $v \in V$, label dissimilarity is defined as:

$$\mathcal{D}(l_v, l'_v) = 1 - \frac{|l_v \cap l'_v|}{|l_v \cup l'_v|} \tag{2}$$

where l_v is the set of v's original labels and l'_v the set of labels in the modified graph.

Thus, for the modified graph including n noisy nodes, and m noisy edges, information loss is defined as

$$IL = \omega_1 n + \omega_2 m + (1 - \omega_1 - \omega_2) \sum \mathcal{D}(l_v, l'_v) \tag{3}$$

where ω_1 ,ω_2 and $1 - \omega_1 - \omega_2$ are weights for each part of the information loss.

Figure 13 shows the measurements of information loss on the synthetic data set using each algorithm. Algorithm *GINN* introduces the least information loss, compared to the other two algorithms.

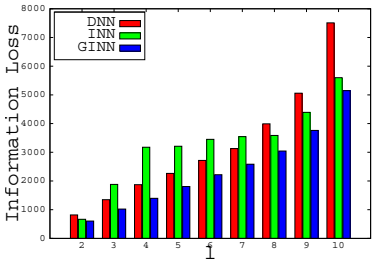


Fig. 13. Information Loss

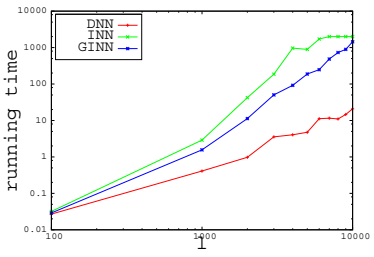


Fig. 14. Running Time

5.4 Algorithm Scalability

We measure the running time of the methods for a series of synthetic graphs with varying number of nodes in our third dataset described above. Figure 14 presents the running time of each algorithm as the number of nodes increases. Algorithm *DNN* is faster than the other two algorithms, showing good scalability. What contributes to such efficiency is that Algorithm DNN adds noisy nodes directly, and no additional process is implemented, as in Algorithm INN and *GINN*. Algorithm *GINN* can also be adopted for quite large graphs as follows: We separate the nodes to two different categories, with or without sensitive labels. Such smaller granularity reduces the number of nodes the anonymization method needs to process, and thus improves the overall efficiency. Algorithm *INN* exhibits the worst scalability. This is because of the large quantity of noisy nodes added, as shown in Figure 12.

6 Conclusions

In this paper we have investigated the protection of private label information in social network data publication. We consider graphs with rich label information, which are categorized to be either sensitive or non-sensitive. We assume that adversaries possess prior knowledge about a node's degree and the labels of its neighbors, and can use that to infer the sensitive labels of targets. We suggested a model for attaining privacy while publishing the data, in which node labels are *both* part of an adversary's background knowledge *and* sensitive information that has to be protected. We accompany our model with an algorithm that transforms a network graph before publication, so as to limit an adversaries confidence about sensitive label data by making any node with a sensitive label indistinguishable from at least $\ell - 1$ other nodes with different sensitive labels, in terms of their neighborhood information. Our experiments on both real and synthetic data sets confirm the effectiveness, efficiency and scalability of our approach in maintaining critical graph properties while providing a comprehensible privacy guarantee.

References

1. L. A. Adamic and N. Glance. The political blogosphere and the 2004 U.S. election: divided they blog. In *LinkKDD*, 2005.
2. L. Backstrom, C. Dwork, and J. M. Kleinberg. Wherefore art thou R3579X?: anonymized social networks, hidden patterns, and structural steganography. *Commun. ACM*, 54(12), 2011.
3. S. Bhagat, G. Cormode, B. Krishnamurthy, and D. S. and. Class-based graph anonymization for social network data. *PVLDB*, 2(1), 2009.
4. A. Campan and T. M. Truta. A clustering approach for data and structural anonymity in social networks. In *PinKDD*, 2008.
5. A. Campan, T. M. Truta, and N. Cooper. *P*-sensitive *k*-anonymity with generalization constraints. *Transactions on Data Privacy*, 3(2), 2010.

6. J. Cheng, A. W.-C. Fu, and J. Liu. K-isomorphism: privacy-preserving network publication against structural attacks. In *SIGMOD*, 2010.
7. G. Cormode. Personal privacy vs population privacy: learning to attack anonymization. In *SIGKDD*, 2011.
8. G. Cormode, D. Srivastava, T. Yu, and Q. Zhang. Anonymizing bipartite graph data using safe groupings. *The VLDB Journal*, 19(1), 2010.
9. S. Das, Ö. Egecioglu, and A. E. Abbadi. Anonymizing weighted social network graphs. In *ICDE*, 2010.
10. C. Dwork. Differential privacy: a survey of results. In *TAMC*, 2008.
11. A. G. Francesco Bonchi and T. Tassa. Identity obfuscation in graphs through the information theoretic lens. In *ICDE*, 2011.
12. M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis. Resisting structural re-identification in anonymized social networks. *PVLDB*, 1(1), 2008.
13. Y. Li and H. Shen. Anonymizing graphs against weight-based attacks. In *ICDM Workshops*, 2010.
14. K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD*, 2008.
15. L. Liu, J. Wang, J. Liu, and J. Zhang. Privacy preserving in social networks against sensitive edge disclosure. In *SIAM International Conference on Data Mining*, 2009.
16. A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. ℓ-diversity: privacy beyond k-anonymity. In *ICDE*, 2006.
17. MPI. http://socialnetworks.mpi-sws.org/.
18. Networkx. http://networkx.lanl.gov/.
19. Y. Song, S. Nobari, X. Lu, P. Karras, and S. Bressan. On the privacy and utility of anonymized social networks. In *iiWAS*, 2011.
20. L. Sweeney. K-anonymity: a model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5), 2002.
21. C.-H. Tai, P. S. Yu, D.-N. Yang, and M.-S. Chen. Privacy-preserving social network publication against friendship attacks. In *SIGKDD*, 2011.
22. O. Tore, A. Filip, and S. John. Node centrality in weighted networks: generalizing degree and shortest paths. *Social Networks*, 32(3), 2010.
23. W. Wu, Y. Xiao, W. Wang, Z. He, and Z. Wang. K-symmetry model for identity anonymization in social networks. In *EDBT*, 2010.
24. X. Ying and X. Wu. Randomizing social networks: a spectrum perserving approach. In *SDM*, 2008.
25. X. Ying and X. Wu. On link privacy in randomizing social networks. In *PAKDD*, 2009.
26. M. Yuan, L. Chen, and P. S. Yu. Personalized privacy protection in social networks. *PVLDB*, 4(2), 2010.
27. L. Zhang and W. Zhang. Edge anonymity in social network graphs. In *CSE*, 2009.
28. E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *PinKDD*, 2007.
29. B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *ICDE*, 2008.
30. B. Zhou and J. Pei. The k-anonymity and ℓ-diversity approaches for privacy preservation in social networks against neighborhood attacks. *Knowledge and Information Systems*, 28(1), 2010.
31. L. Zou, L. Chen, , and M. T. Özsu. K-automorphism: a general framework for privacy-preserving network publication. *PVLDB*, 2(1), 2009.