Protecting Sensitive Labels in Social Network Data Anonymization

Mingxuan Yuan, Lei Chen, *Member*, *IEEE*, Philip S. Yu, *Fellow*, *IEEE*, and Ting Yu

Abstract—Privacy is one of the major concerns when publishing or sharing social network data for social science research and business analysis. Recently, researchers have developed privacy models similar to k-anonymity to prevent node reidentification through structure information. However, even when these privacy models are enforced, an attacker may still be able to infer one's private information if a group of nodes largely share the same sensitive labels (i.e., attributes). In other words, the label-node relationship is not well protected by pure structure anonymization methods. Furthermore, existing approaches, which rely on edge editing or node clustering, may significantly alter key graph properties. In this paper, we define a k-degree-l-diversity anonymity model that considers the protection of structural information as well as sensitive labels of individuals. We further propose a novel anonymization methodology based on adding noise nodes. We develop a new algorithm by adding noise nodes into the original graph with the consideration of introducing the least distortion to graph properties. Most importantly, we provide a rigorous analysis of the theoretical bounds on the number of noise nodes added and their impacts on an important graph property. We conduct extensive experiments to evaluate the effectiveness of the proposed technique.

Index Terms—Social networks, privacy, anonymous

1 Introduction

With the rapid growth of social networks, such as Facebook and Linkedin, more and more researchers found that it is a great opportunity to obtain useful information from these social network data, such as the user behavior, community growth, disease spreading, etc. However, it is paramount that published social network data should not reveal private information of individuals. Thus, how to protect individual's privacy and at the same time preserve the utility of social network data becomes a challenging topic. In this paper, we consider a graph model where each vertex in the graph is associated with a sensitive label. Fig. 1a shows an example of such a graph.

Recently, much work has been done on anonymizing tabular microdata. A variety of privacy models as well as anonymization algorithms have been developed (e.g., k-anonymity [26], l-diversity [20], t-closeness [17]). In tabular microdata, some of the nonsensitive attributes, called quasi identifiers, can be used to reidentify individuals and their sensitive attributes. When publishing social network data, graph structures are also published with corresponding social relationships. As a result, it may be exploited as a new means to compromise privacy.

A structure attack refers to an attack that uses the structure information, such as the degree and the subgraph of a node, to identify the node. To prevent structure attacks, a published graph should satisfy k-anonymity [18], [32], [15], [34]. The goal is to publish a social graph, which always has at least k candidates in different attack scenarios in order to protect privacy. Liu and Terzi [18] did pioneer work in this direction that defined a k-degree anonymity model to prevent degree attacks (Attacks use the degree of a node). A graph is k-degree anonymous if and only if for any node in this graph, there exist at least $k-1$ other nodes with the same degree.

Fig. 1a shows an example of a possible structure attack using degree information. If an adversary knows that one person has three friends in the graph, he can immediately know that node 2 is that person and the related attributes of node 2 are revealed. k-degree anonymity can be used to prevent such structure attacks. However, in many applications, a social network where each node has sensitive attributes should be published [5], [33]. For example, a graph may contain the user salaries which are sensitive [33]. In this case, k-degree alone is not sufficient to prevent the inference of sensitive attributes of individuals. Fig. 1b shows a graph that satisfies 2-degree anonymity but node labels are not considered. In it, nodes 2 and 3 have the same degree 3, but they both have the label "80K." If an attacker knows someone has three friends in the social network, he can conclude that this person's salary is 80K without exactly reidentifying the node. Therefore, when sensitive labels are considered, the l-diversity should be adopted for graphs. Again, the l-diversity concept here has the same meaning as that defined over tabular data [20]. For example, if we choose the distinct l-diversity, for the nodes with the same degree, their associated sensitive

- M. Yuan is with Huawei Noah Ark Lab, Hong Kong and the Department of Computer Science and Engineering, HKUST, Hong Kong. E- mail: csyuan@cse.ust.hk.
- L. Chen is with the Department of Computer Science and Engineering, HKUST, Hong Kong. E-mail: leichen@cse.ust.hk
- P.S. Yu is with the Department of Computer Science, University of Illinois at Chicago, IL 60607 and the Computer Science Department, King Abdulaziz University, Jeddah, Saudi Arabia. E-mail: psyu@cs.uic.edu.
- T. Yu is with the Department of Computer Science, North Carolina State University, NC 27695. E-mail: yu@csc.ncsu.edu.

Manuscript received 19 Dec. 2010; revised 13 Aug. 2011; accepted 31 Aug. 2011; published online 16 Dec. 2011.
Recommended for acceptance by R. Kumar.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2010-12-0681.
Digital Object Identifier no. 10.1109/TKDE.2011.259.

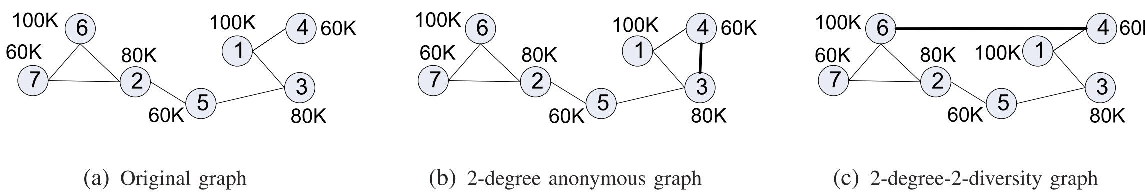

(a) Original graph (b) 2-degree anonymous graph (c) 2-degree-2-diversity graph

Fig. 1. Publish a graph with degree and label anonymity.

labels must have l distinct values. Fig. 1c shows a graph that satisfies 2-degree anonymity and 2-diversity. For each distinct degree appearing in this graph, there exist at least two nodes. Moreover, for those nodes with the same degree, they contain at least two distinct sensitive labels. Thus, the attacker cannot reidentify a node or find the node-label relation with degree knowledge. In this paper, we select the degree-attack, one of the popular attacks methods [18], [15], to show how we can design mechanisms to protect both identities and sensitive labels. With respect to other types of attacks, such as subgraph query attacks or hub node query attacks [15], we believe that the key ideas proposed in this work can be adopted to handle them as well, though more complicated extensions may be needed. We give an extensive discussion in Section 6.

Current approaches for protecting graph privacy can be classified into two categories: clustering [15], [7], [30], [4] and edge editing [10], [29], [32], [18], [30]. Clustering is to merge a subgraph to one super node, which is unsuitable for sensitive labeled graphs, since when a group of nodes are merged into one super node, the node-label relations have been lost. Edge-editing methods keep the nodes in the original graph unchanged and only add/delete/swap edges. For example, to protect privacy of Fig. 2a, we convert it to satisfy 3-degree anonymous and 3-diversity by adding edges as shown in Fig. 2b. However, edge editing may largely destroy the properties of a graph. The edge-editing method sometimes may change the distance properties substantially by connecting two faraway nodes together or deleting the bridge link between two communities. In Fig. 2b, the distance between nodes 6 and 12 is changed from 5 to 1 hop. This phenomenon is not preferred. Mining over these data might get the wrong conclusion about how the salaries are distributed in the society. Therefore, solely relying on edge editing may not be a good solution to preserve data utility.

To address this issue, we propose a novel idea to preserve important graph properties, such as distances between nodes by adding certain "noise" nodes into a graph. This idea is based on the following key observation. Most social networks satisfy the Power Law distribution [2], i.e., there exist a large number of low degree vertices in the graph which could be used to hide added noise nodes from being reidentified. By carefully inserting noise nodes, some graph properties could be better preserved than a pure edge-editing method. Fig. 2c is a graph converted from Fig. 2a by adding noise nodes to satisfy 3-degree-3-diversity. The distances between the original nodes are mostly preserved.

Our privacy preserving goal is to prevent an attacker from reidentifying a user and finding the fact that a certain user has a specific sensitive value. To achieve this goal, we define a k-degree-l-diversity (KDLD) model for safely publishing a labeled graph, and then develop corresponding graph anonymization algorithms with the least distortion to the properties of the original graph, such as degrees and distances between nodes.

To summarize, we made the following contributions:

- We combine k-degree anonymity with l-diversity to prevent not only the reidentification of individual nodes but also the revelation of a sensitive attribute associated with each node. We use distinct l-diversity to demonstrate our algorithm and give the detailed discussion about how more complex recursive (c, l)-diversity can be implemented.
- We propose a novel graph construction technique which makes use of noise nodes to preserve utilities of the original graph. Two key properties are considered: 1) Add as few noise edges as possible; 2) Change the distance between nodes as less as possible.
- We present analytical results to show the relationship between the number of noise nodes added and their impacts on an important graph property. We further conduct comprehensive experiments for both distinct l-diversity and recursive (c, l)-diversity to show our technique's effectiveness.

The rest of the paper is arranged as follows: Section 2 defines the problem and introduces the framework. Sections 3 and 4 describe the details of the algorithm implementation. Section 5 demonstrates how the more complex (c, l)-diversity

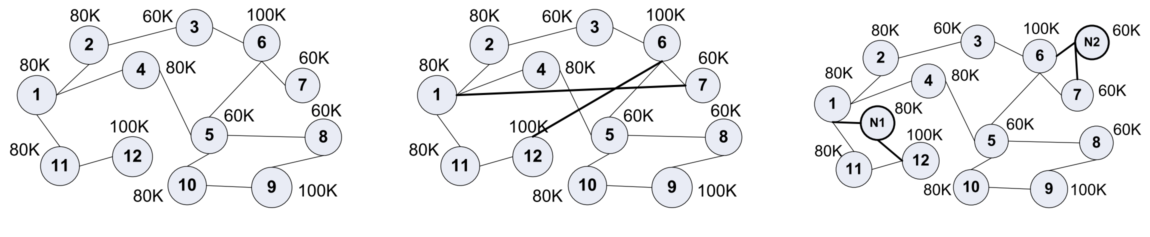

(a) Original graph (b) 3-degree-3-diversity by adding edges (c) 3-degree-3-diversity by adding noise

Fig. 2. Example for adding noise node.

model is adapted into our framework in detail. We discuss the possible extension of our work to handle more complex graph protection model in Section 6. We report the experiment results on real data sets in Section 7. Finally, we compare our work with related proposals in Section 8 and conclude in Section 9.

2 Problem Description

In this paper, a social network graph is defined as:

Definition 1. *Social Network Graph: a social network graph is a four tuple $G(V, E, \sigma, \lambda)$, where V is a set of vertices, and each vertex represents a node in the social network. $E \subseteq V \times V$ is the set of edges between vertices, σ is a set of labels that vertices have. $\lambda : V \longrightarrow \sigma$ maps vertices to their labels.*

In this paper, we use the words "node" and "vertex" interchangeably. In a published (privacy preserving) social network graph, an attacker could reidentify a node by degree information [18] and further infer sensitive labels. To prevent this possible leakage, we define "k-degree-l-diversity" principle for published graphs, which have the same spirit of $k - l$ diversity in relational data [26], [20].

Definition 2(KDLD). *For each vertex in a graph, there exist at least $k - 1$ other vertices having the same degree in the graph. Moreover, the vertices with the same degree contain at least l distinct sensitive labels.*

Here, we use distinct l-diversity [20] to ensure that there exist at least l distinct labels in each equivalent class (group), i.e., a group of nodes having the same degree.[1] We use distinct l-diversity to demonstrate the basic working procedure of our method. We give the detailed discussion about how to handle more complex l-diversity such as recursive (c, l)-diversity in Section 5. We call a graph a *KDLD graph* if it satisfies the k-degree-l-diversity constraint. A *KDLD graph* protects two aspects of each user when an attacker uses degree information to attack: 1) The probability that an attacker can correctly reidentify this user is at most $\frac{1}{k}$; 2) The sensitive label of this user can at least be related with l different values. Since each equivalent class contains at least k nodes, when an attacker uses the degree to reidentify a node, the probability he correctly reidentifies this user is at most $\frac{1}{k}$. Furthermore, since there are at least l distinct labels in each equivalent class, a user's sensitive label is related to at least l values. Our goal is to protect each user's privacy by adding certain edges/nodes to transfer a social network graph to a KDLD graph. We refer the added edges/nodes as noise edges/nodes. To guarantee the correctness of information, for any user who appears in the published graph, we also preserve the sensitive label associated with this user's node. In order to keep the utility of the published graph, when generating the *KDLD* graph, two key properties should be preserved [24]:

- Low overhead: It is necessary to add as few noise edges as possible to reduce the additional overhead on the social infrastructure;
- Social distance: The noise edges/nodes added should connect nodes that are close with respect to the social distance.

Since each noise node connects with at least one noise edge, "Low Overhead" also limits the number of noise nodes that can be added. The social distance between two nodes u, v is the shortest path length between u and v in the original graph. The social distance between all node pairs of a graph is measured by **Average shortest path length** (APL). APL is a concept in network topology that is defined as the average of distances between all pairs of nodes. It is a measure of the efficiency of information or mass transport on a network. Some queries like "the nearest node for a group of nodes" are related to APL. The APL of a graph G is

$$APL_G = \frac{2}{N(N-1)} \sum_{\forall n_i, n_j \in G} d(n_i, n_j),$$

where, $d(n_i, n_j)$ is the length of the shortest path between nodes n_i and n_j, N is the number of nodes in the graph.

We design a two step algorithm to generate the *KDLD* graph which tries to preserve the above two key properties. In the first step, we compute a target degree for each node so that it makes the original graph satisfy *KDLD* constraint with the minimum sum of degree change. Clearly, smaller degree change needs fewer noise edges to implement the change. In the second step, we change each node's degree to its target degree by adding noise edges/nodes. We utilize the noise nodes to make the change of APL as small as possible. Our proposed two step algorithm considers both the "Low Overhead" and the "Preserve Social Distance" requirements.

Next, we first introduce two data structures we use in the rest of this paper, and then give the formal description the two steps in our algorithm. We borrow the concept of "degree sequence" used in [18] and define a *sensitive degree sequence P* as follows:

Definition 3. *Given a graph G, its sensitive degree sequence is a sequence of n triples: $[P[1], \dots, P[n]]$ where $P[1].d \geq P[2].d \geq \dots \geq P[n].d$, $P[i]$ is a triple (id, d, s) at position i in P, d is the degree, and s is the sensitive label associated with node id.*

We use lowercase p_u to represent node u's corresponding triple in P. For example, the sensitive degree sequence of the graph in Fig. 1b is [(2,3,80K), (3,3,80K), (1,2,100K), (4,2,60K), (5,2,60K), (6,2,100K), (7,2,60K)]. In it, $p_2 = (2, 3, 80K)$ represents node 2 in the graph. For a node u, we use $u.d$ to denote its degree and $u.s$ for its label. Now, we can define *KDLD* model of a graph based on its sensitive degree sequence:

Definition 4. *KDLD sequence: A sensitive degree sequence P is a KDLD sequence if P satisfies the following constraint: P can be divided into a group of subsequences $[[P[1], \dots, P[i_1]], [P[i_1 + 1], \dots, P[i_2]], [P[i_2 + 1], \dots, P[i_3]], \dots, [[P[i_m + 1], \dots, P[j]]$ such that for any subsequence $P_x = [P[i_x], \dots, P[i_{x+1}]]$, P_x satisfies three constraints: 1) All the elements in P_x share the same degree ($P[i_x].d = P[i_x + 1].d = \dots = P[i_{x+1}].d$); 2) P_x has size at least k ($i_{x+1} - i_x + 1 \geq k$); 3) P_x's label set $\{P[t].s | i_x \leq t \leq i_{x+1}\}$ have at least l distinct values.*

1. If nodes have other labels (quasi identifiers) than the sensitive label, the quasi identifiers of the nodes in each equivalent class will be adjusted to be the same.

TABLE 1
Meanings of Symbols Used

G	The original graph
G'	The published graph
N	The number of nodes in G
$KDLD$	k-degree-l diversity
P	The sensitive degree sequence of G
P'	The sensitive degree sequence of G'
P^{new}	The KDLD sequence generated from P
APL	Average Shortest Path Length
$ACSPL$	Average Change of Sensitive Label Path Length
$RRTI$	Remaining ratio of top influential users
CC	Clustering Coefficient

For example, the sensitive degree sequence of the Fig. 1c is [(2,3,80K), (6,3,100K), (1,2,100K), (3,2,80K), (4,2,60K), (5,2,60K), (7,2,60K)], it is a $2D2D$ sequence. For a *KDLD* sequence P, each its subsequence $[P[i_x], \ldots, P[i_{x+1}]]$ forms a same degree group c_x. Then, from P, we can get same degree groups: $C = \{c_1, c_2, \ldots, c_m\}$. For each $c_i \in C$, we use $c_i.d$ to denote the degree of the corresponding group. It is obvious that a graph $G(V, E)$ is a *KDLD* graph if and only if the sensitive degree sequence of G is a *KDLD* sequence. The benefit of using *KDLD* sequence is that it compactly represents the conditions to make a graph be a *KDLD* graph. The sensitive degree sequence in the above example is a $2D2D$ sequence, therefore Fig. 1c is a $2D2D$ graph. Based on the above two data structures, we use two steps to solve the following two subproblems, respectively:

- *KDLD* sequence generation: Given the sensitive degree sequence P of graph G and two integers k and l, compute a *KDLD* sequence P^{new} which contains the same set of nodes as P with minimum $L(P, P^{new}) = \sum_{\forall u} |p_u.d - p_u^{new}.d|$. This equation computes the degree change by comparing the same node u's degree in these two sequences. The purpose is to obtain a new *KDLD* sequence from G so that the degree change of all the nodes in G is as small as possible. Clearly, smaller degree change needs fewer noise edges to implement the change.
- Graph construction: Given a graph $G(V, E, \sigma, \lambda)$ and a sensitive degree sequence P^{new}, construct a new graph $G'(V', E', \sigma, \lambda')$ with $V \subseteq V'$. The sensitive degree sequence P' of G' is a *KDLD* sequence and P' has all the elements in P^{new} since G' is constructed from G by adding some noise nodes. Meanwhile, $|APL_G - APL_{G'}|$ is minimized.

Next, we introduce how to implement these two steps. To be convenient, we summarize the commonly used symbols of this paper in Table 1.

3 *KDLD* Sequence Generation

To generate a *KDLD* sequence, the triples in P should be divided into groups. All the corresponding nodes in the same group shall be adjusted to have the same degree. We employ two algorithms for this problem: Algorithm K-L-BASED and Algorithm L-K-BASED.[2] The algorithms tend to put the nodes with similar degrees into the same group to reduce the degree changes. Given the degree sequence P of the original graph, Algorithm K-L-BASED selects the first k elements in P as a group. We keep on merging the next element into the current group until the l-diversity constraint is satisfied. After a group satisfies k-degree and l-diversity constraints, we calculate two costs:

- C_{new}: the cost of creating a new group for the next k elements (the total degree changes that make all the nodes in this group have the same degree).
- C_{merge}: the cost of merging the next element into the current group and creating a new group for the next k elements by skipping the next element.

If C_{merge} is smaller, we merge the next element into the current group and continue this comparison process. If C_{new} is smaller, we create a new group with the next k elements and continue checking l-diversity constraints described above. When the remaining elements are less than k or contain less than l distinct sensitive labels, they are merged into the last group. Since nodes are sorted by their degrees in P, building groups using the above method helps to group the nodes with similar degrees together. For example, if using this algorithm to make $P[(1,5,s_1),(2,2,s_1),(3,2,s_1),(4,1,s_2),(5,1,s_2),(6,1,s_1),\ldots]$ satisfy *2D2D* constraint ($k = l = 2$), we get $P^{new} = [(1,5,s_1), (2,5,s_1),(4,5,s_2),(3,2,s_1),(5,2,s_2),(6,2,s_1),\ldots]$.

Different from Algorithm K-L-BASED which checks the k-degree first, Algorithm L-K-BASED tries to satisfy the l-diversity first. If we use t to represent the position number of each element in P. Each time Algorithm L-K-BASED picks l nongrouped $P[t]$s in P that do not share any common labels with the minimum $\sum t$. Minimum $\sum t$ guarantees the algorithm select l nodes with most similar degrees in the nongrouped nodes. The algorithm then continues merging *P[t]* with minimum t in the remaining elements until we get k items. Then similar to Algorithm K-L-BASED, we use the same cost function to determine if the next element should be merged or not. For example, if using Algorithm L-K-BASED to make $P[(1,5,s_1),(2,2,s_1),(3,2,s_1),(4,1,s_2),(5,1,s_2),(6,1,s_1),\ldots]$ satisfy *2D2D* constraint ($k = l = 2$), we can get $P^{new} = [(1,5,s_1),(4,5,s_2),(2,2,s_1), (5,2,s_2),(3,2,s_1),(6,2,s_1),\ldots]$. Ghinita et al. [12], [13] designed a heuristic algorithm for 1D l-diversity models, which do not have k-anonymous requirement. For distinct l-diversity with $k = l$, our LK-based algorithm follows the same logic as their heuristic algorithm.

All the nodes in the same group will be adjusted to have the same degree in the next graph construction step. We use the mean degree of a group as its target degree. This target degree is also used to estimate the cost in Algorithm K-L-BASED and Algorithm L-K-BASED. The Algorithm K-L-BASED runs in $O(n)$ time since it only scans the degree sequence P

2. For the case that each node has some quasi-identifier labels, the generalization cost on quasi-identifier labels in each group can be combined into the cost function in the *KDLD* sequence generation algorithms.

TABLE 2
Algorithm 1: Neighborhood_Edge_Editing()

```
for each node u need to increase degree do                                  /* Case 1 */
    $d$ = $u$'s degree;
    $d'$ = $u$'s target degree;
    for $i = 0$; $i < d' - d$; $i$++ do
        Find $v$,$w$ has link $(u, v)$, $(v, w)$ and $v$ need to decrease degree;
        if Such $v$,$w$ exist then
            Remove link $(v, w)$;
            Add link $(u, w)$;
        else
            break;
for each node u need to increase degree do                                  /* Case 2 */
    for each node v need to increase degree do
        if $u$, $v$ are 2 hop neighbor then
            if $u$, $v$ do not have link then
                Add link $(u, v)$;
for each node u need to decrease degree do                                  /* Case 3 */
    for each node v need to decrease degree do
        if $u$, $v$ have link then
            Remove link $(u, v)$;
            if $\neg$($u$, $v$ are 2 hop neighbor) then
                Add link $(u, v)$;
```

once. The Algorithm L-K-BASED can also run in $O(n)$ time if we store a candidate list for each node.

4 Graph Construction

4.1 Algorithm Skeleton

After getting the new sensitive degree sequence P^{new}, a new graph G' should be constructed. Suppose P' is the sensitive degree sequence of G', P'_o is the sensitive degree sequence of G' that only contains the nodes in G. Our graph construction algorithm makes $P'_o == P^{new}$ and P' as a *KDLD* sequence. For example, if graph G's degree sequence $P = [(1,3,s_1), (2,2,s_2),(3,1,s_1),(4,1,s_1),(5,1,s_2)]$ and $P^{new} = [(1,3,s_1), (2,3,s_2),(3,1,s_1),(4,1,s_1),(5,1,s_2)]$, then a new graph G' can be constructed with degree sequence $P' = [(1,3,s_1,(2,3, s_2),(3,1,s_1),(4,1,s_1),(5,1,s_2),(6,1,s_1)]$.

Motivated by the observation example (Fig. 5), we propose graph construction algorithms by making use of noise nodes to preserve the APL better. We use the following algorithm to construct the published graph which preserves the APL. The algorithm contains five steps:

- Step 1: Neighborhood_Edge_Editing()
  We add or delete some edges if the corresponding edge-editing operation follows the ***neighborhood rule*** (The details can be found in Section 4.2). By doing this, the sensitive degree sequence P of original graph G is closer to P^{new} in case APL is preserved;
- Step 2: Adding_Node_Decrease_Degree()
  For any node whose degree is larger than its target degree in P^{new}, we decrease its degree to the target degree by making using of noise nodes;
- Step 3: Adding_Node_Increase_Degree()
  For any node whose degree is smaller than its target degree in P^{new}, we increase its degree to the target degree by making using of noise nodes;

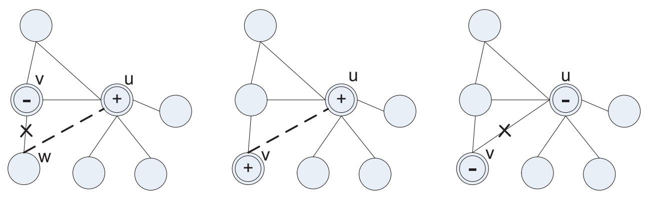


Fig. 3. Edge editing with neighborhood rule.

- Step 4: New_Node_Degree_Setting()
  For any noise node, if its degree does not appear in P^{new}, we do some adjustment to make it has a degree in P^{new}. Then, the noise nodes are added into the same degree groups in P^{new};
- Step 5: New_Node_Label_Setting()
  We assign sensitive labels to noise nodes to make sure all the same degree groups still satisfy the requirement of the distinct l-diversity. It is obvious that after Step 4 and Step 5, the sensitive degree sequence P' of the published graph G' is a KDLD sequence.

In Steps 2, 3, and 4, we carefully connect the noise nodes into the graph to make the change of APL as less as possible. Next, we introduced these steps in detail.

4.2 Step 1

We use Algorithm 1 (Table 2) to change some nodes' degrees by adding/deleting edges first. In order to preserve APL, we force each change to follow the ***neighborhood rule*** described as follows:

- Case 1: Node u needs to increase its degree, v needs to decrease its degree and u,v are direct neighbors. In this case, we randomly select one direct neighbor w of v which does not connect to u. Remove the edge (v,w) and add a new edge (u,w) (See Fig. 3a). By doing this, u's degree is increased by 1 and v's degree is decreased by 1 while all the other nodes' degrees remain unchanged. This operation changes the distance between u,w from 2 to 1 and the distance between v,w from 1 to 2. Therefore, only the paths passing through u,w or v,w change their lengths at most by 1;
- Case 2: Two nodes u, v are two hop neighbors and they both need to increase their degree. If there does not exist link (u,v), we add link (u,v) (See Fig. 3b). The distance between u,v is changed from 2 to 1. Only the lengths of shortest paths passing though u, and v change by 1;
- Case 3: Two nodes u and v that both need to decrease their degrees are direct neighbors. If after removing link (u,v), u and v are still two hop neighbors, remove the link (u,v) (See Fig. 3c). The distance between u,v is changed from 1 to 2. So it only changes the lengths of shortest paths passing through u,v at most by 1.

A node's degree could be either increased or decreased. So the above policies consider all the cases for two nodes u and v whose degrees need to be changed. It is easy to see that any edge-editing operation between these two nodes may

TABLE 3
Algorithm 2: Adding_Node_Decrease_Degree()

```
1  for every node $u$ that need to decrease the degree do
2      $d = u$'s degree;
3      $target =$ the target degree of $u$;
4      while true do
5          Select a sensitive value $s$ from $u$'s original one hop neighbor;
6          Create a new node $n$ with sensitive value $s$;
7          $d' = 1$;
8          $target_{new} = Select\_Closest\_Degree\_In\_Group(d + 2 -$ (the target degree));
9          Connect node $u$ with $n$;
10         $d = d + 1$;
11         while true do
12             Random select a link $(u, v)$ which is in $G$;
13             Delete link $(u, v)$, Create link$(n, v)$;
14             $d' = d' + 1$;
15             $d = d - 1$;
16             if $d' == target_{new} \vee d == target$ then
17                 break;
18         if $d == target$ then
19             break;
```

TABLE 4
Algorithm 3: Adding_Node_Increase_Degree()

```
1  for every node $u$ in $G$ which needs to increase its degree do
2      for $i$=0; $i < increase\_num$; $i$++ do
3          Create a new node $n$;
4          Connect node $u$ with $n$;
5          for every node $v$ that is one or two hop neighbor of node $u$ do
6              if $v$ needs to increase its degree then
7                  Connect node $v$ with $n$;
8          while $n$'s degree is not in $P'$ $\wedge n$'s degree $>$ min group degree do
9              Remove the last connection created to n;
10             i=i-1;
```

cause the lengths of shortest paths passing through u and v change at least by 2 except the above three situations.

4.3 Step 2

For any node u whose degree is still larger than its target degree after Step 1, we decrease its degree following the steps below (Algorithm 2, as shown in Table 3):

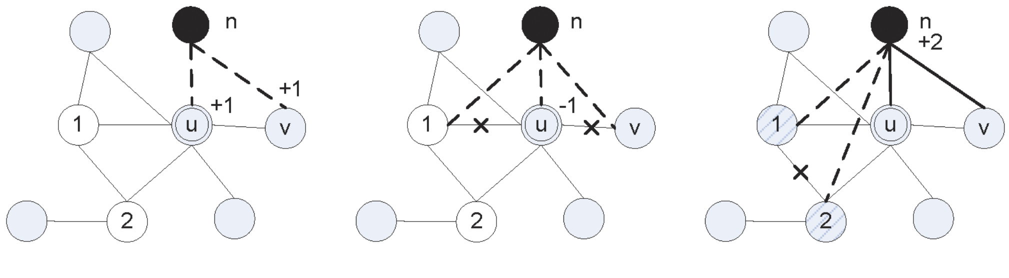
- Create a new node n and connect u with n, now $u.d = p_u.d + 1$ and $n.d = 1$.
- Set n's target degree $target_{new}$ to a degree in P'. If $u.d + 2 - p'_u.d$ is less than the minimum degree in P', set $target_{new}$ as this minimum degree. Otherwise, set $target_{new}$ as a degree in P' which is less than and closest to $u.d + 2 - p'_u.d$. We select $u.d + 2 - p'_u.d$ because we change u's degree through n. As a consequence, n's degree is changed to $u.d + 2 - p'_u.d$.
- Randomly select an edge (u, v) from the original graph, delete this edge, then add a new edge (n, v). By doing this, v only changes from u's one hop neighbor to its two hop neighbor. We repeat this random edge modification process until $n.d = target_{new}$ or $u.d = p'_u.d$. An example of decreasing node degree by adding the noise node can be found in Fig. 4b. If $n.d = target_{new}$ and $u.d > p'_u.d$, goto the first step to create a new noise node. If $u.d = p'_u.d$ but $n.d < target_{new}$, node u's degree has been successfully adjusted to $p'_u.d$. We finish the adjustment of node u, and leave node n to noise hiding step. Since each time, we reduce u's degree by 1 to increase n's degree by 1, and the creation of n (described in previous step) adds degree 1 to both u and n, $u.d + 2 - p'_u.d$ is n's degree when u reaches its target degree.

4.4 Step 3

For each vertex u in G which needs to increase its degree, we use Algorithm 3 (Table 4) to make its degree reach the target degree. We first check whether there exists a node v within two hops of u, and v also needs to increase its degree (In Step 2, some noise nodes are added, such v may exist.), we connect n with v. Since v is within two hops of u, connecting v with n will not change the distance between u and v. Fig. 4a shows an example of this operation. After this step, if n's degree is bigger than the minimum degree in P^{new} but does not appear in P^{new}, we recursively delete the last created link until the degree of n equals to a degree in P^{new}. Otherwise, we leave n to Step 4 for processing and continue adding noise to u if $u.d < p'_u.d$. By doing this, u's degree is increased to its target degree.

4.5 Step 4

In this step, we make sure each noise node has a degree in P^{new}. By doing this, all the noise nodes are added into existing same degree groups in P^{new}. Furthermore, since we



(a) Increase Degree (b) Decrease Degree (c) Hide Noise

Fig. 4. Strategies to adding noise nodes.

TABLE 5
Algorithm 4: New_Node_Degree_Setting()

```
1  Select pairs of noise nodes that each pair of nodes are within 3 hops to each other;
2  Build a link for each pair;
3  for every node $n$ has even degree do
4      Select an even degree $target_{new}$ ($n.d < target_{new}$) in $P^{new}$;
5  for every node $n$ has odd degree do
6      Select an odd degree $target_{new}$ ($n.d < target_{new}$) in $P^{new}$;
7  for each noise node $n$ do
8      while $n.d \neq target_{new}$ do
9          Find a link $(u, v)$ with the minimum $\frac{dis(u,n)+dis(v,n)}{2}$ in current graph where $u$ and $v$ are not
           connected to $n$;
10         Remove link $(u, v)$;
11         Add link $(n, u)$;
12         Add link $(n, v)$;
```

assume an attacker use the degree information of some nodes to do the attack, if the noise nodes have degrees as the same as some original nodes, an attacker cannot distinguish them from the original nodes with the degree information. So, we need to make all the noise nodes' degrees to be some degrees that already exist in P^{new}. To keep the APL, during the process, we only change connections in the noise nodes' neighborhoods to minimize the distance changes.

The details of Step 4 is shown in Algorithm 4 (Table 5). We first select pairs of noise nodes (which need to increase their degrees) that each pair of nodes are within three hops to each other and build a link for each pair. If two noise nodes are within three hops, they are either connected to the same node in the original graph or they are, respectively, connected to two directly connected nodes in the original graph. If the two noise nodes are connected to the same original node, connecting them does not change any shortest path between original nodes. If the two noise nodes are, respectively, connected to two original nodes which are direct neighbors, since connecting these two noise nodes does not change the distance between the two original nodes, the shortest path lengths between the nodes in the original graph are not changed, too.

Then, for each noise node whose degree is an even number, we select an even degree in P^{new} (a degree in P^{new} which is an even number) that is close to and bigger than this noise node's current degree as its target degree. For example, if a noise node's degree is 8 and the degrees appear in P^{new} are $\{3, 4, 5, 6, 7, 9, 10, 12, 15, 16\}$, we set its target degree as 10. We find all the nearest edges which do not directly connect with current processing noise node. The "nearest" edge to a noise node means the average distance of the two points (u, v) of an edge to this noise node $\frac{dis(u,noise)+dis(v,noise)}{2}$ is minimum among all edges. We select one edge within the nearest edge set randomly, remove the edge from the graph and connect the endpoints of this edge to the current processing noise node as shown in Fig. 4c. After that, the degree of the noise node increases by 2, and meanwhile, all the other vertices' degrees remain unchanged. We repeat this procedure until the noise node's degree reaches the target degree. For each noise node with an odd degree, we select an odd target degree and conduct the same adjustment.

Since the two endpoints of the removed edge are both near to the noise node, directly connecting them to the noise node makes them stay in the neighborhood of the noise node.

TABLE 6
Algorithm 5: New_Node_Label_Setting()

```
1  for every noise node $n$ do
2      $u$ is the node in $G$ which $n$ is created for;
3      Randomly select a node $v$ where there exists link $(u, v)$ in $G$;
4      set $n$'s sensitive label as $v$'s sensitive label;
```

Moreover, since the distance between the two endpoints increases only by 1, the lengths of paths through these two endpoints are increased only by 1 as well. When only even or odd degree groups exist, it's easy to change the KDLD sequence generation algorithm to get a new sensitive degree sequence that contains both even and odd degrees by increasing some groups' degrees by 1.

4.6 Step 5

The last step is to assign sensitive labels to noise nodes to make all the same degree group still satisfy the requirement of distinct l-diversity. Since in each same degree group, there are already l distinct sensitive labels in it, it is obviously the new added noise nodes can have any sensitive label. We use the following way to sensitive label to a noise node n: suppose u is the original node in G which n is created for. We randomly find a label from the direct neighbors of u in the original graph G. Since there is a node with this label directly connected with u in the original graph, the noise node n with the same label connected with u can help preserve the distances between this label and other labels in the graph. The label distribution's change after this setting is very small. We show this in Section 7.4.3 of the experiment part. The details of this step is shown in Algorithm 5 (Table 6).

4.7 Anonymization Utility Analysis

In Step 1, we add or delete some edges following the "neighborhood rule." To show the effectiveness of the noise nodes adding strategies in Steps 2, 3, and 4, we analyze the bound of the APL change when given the number of added noise nodes in the average case.

Theorem 1. *Suppose G_0 is the graph generated after Step 1 with N nodes, whose APL is APL_{G_0}. If G_M is the KDLD graph generated by adding M noise nodes. Our algorithm guarantees that the APL of G_M, APL_{G_M} is bounded in average case:*

$$\left(1-\frac{4}{N+1}\right)^M APL_{G_0} \leq APL_{G_M} \leq \left(1+\frac{4}{N}\right)^M APL_{G_0} + \frac{2}{N-4}.$$

Here, "in average case" means the expected change value. We assume the adding/deleting of each edge changes the same number of shortest paths. When adding 1,000 noise nodes into a graph with 10,000 nodes, the upper bound of APL_{G_M} is around $1.49APL_{G_0}$ and the lower bound of APL_{G_M} is around $0.67APL_{G_0}$. The detailed proof can be found in Appendix I, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2011.259.

TABLE 7
Algorithm 6: New_Node_Label_Setting()

```
1  for each same degree group $C$ do
2      Group $C_o$ = group of original nodes in $C$;
3      Group $C_n$ = group of noise nodes in $C$;
4      Array $L_o$ = sensitive labels on the nodes in $C_o$;
5      Array $N_o$ = $L_o$'s occurrence numbers in $C_o$;
6      Array $N_n$ = an array with size $|L_o|$;
7      for $i = 0; i < |L_o|; i$++ do
8          $N_n[i] = \lfloor |C_n| \times \frac{N_o[i]}{|C_o|} \rfloor$;
9      if $\sum_{i=0}^{|N_n|-1} N_n[i] < |C_n|$ then
10         Compute $max$ where $N_o[max]$ is the maximum value in $N_o$;
11         $N_n[max] = N_n[max] + |C_n| - \sum_{i=0}^{|N_n|-1} N_n[i]$;
12     while $\neg$ ($N_o + N_n$ satisfies recursive $(c,l)$-diversity) do
13         Compute $min$ where $N_n[min] + N_o[min]$ is the minimum value in $N_n + N_o$;
14         Compute $max$ where $N_n[max] + N_o[max]$ is the maximum value in $N_n + N_o$;
15         $N_n[max] = N_n[max] - 1$;
16         $N_n[min] = N_n[min] + 1$;
17     Assign labels to noise nodes in $C_n$ where $\forall i$, $N_n[i]$ noise nodes have label $L_o[i]$;
```

5 More Complex L-Diversity Model

Besides distinct l-diversity, Machanavajjhala et al. [20] also proposed two other l-diversity models: entropy l-diversity and recursive (c, l)-diversity. Entropy l-diversity requests tighter privacy constraints. However, it is too restrictive for the practical purpose [20], [12], [13]. Recursive l-diversity has a more relaxed condition. For any equivalent group C (in our case, an equivalent group is a same degree group), if there are m sensitive labels appearing in C, C satisfies recursive (c, l)-diversity if $f_1 < c(f_l + f_{l+1} + ... + f_m)$. Where c and l are two pregiven constants and f_i is the number of occurrences of the ith most frequent sensitive label in C. For example, in Fig. 2b, there are eight nodes with degree 2 and the sensitive labels associated with these nodes are 60K, 80K, and 100K, respectively. 80K is the most frequent sensitive label which appears four times. 60K is the second one which appears three times and 100K is the third one which appears two times. Thus, this same degree group satisfies 2-diversity with $c = 1$ $(4 < 1(2 + 3))$ and 3-diversity with $c = 3$ $(4 < 3(2))$. In this section, we show how to adapt our algorithm to make it provide recursive (c, l)-diversity instead of distinct l-diversity protection for sensitive labels.

If the generated KDLD sequence P^{new} satisfies recursive (c, l)-diversity and all the same degree groups also satisfy (c, l)-diversity after adding noise nodes, our algorithm works for recursive (c, l)-diversity. So, the key points are to modify the KDLD sequence generation algorithm and graph construction' Step 5 (New_Node_Label_Setting()).

5.1 Step 5 for Recursive (c, l)-Diversity

We use Algorithm 6 (Table 7) to assign sensitive labels to noise nodes. For a same degree group, the basic procedure to assign labels to new nodes added into this group is: 1) Assign labels to new nodes follows the label distribution of the original nodes in this group; 2) If the (c, l)-diversity is not satisfied, assign the least frequent label in current group to a noise node who is assigned with the most frequent label (By doing this, the most frequent label's occurrence number is decreased by 1 and the least frequent label's occurrence number is increased by 1). We repeatly do this until the new group satisfies the recursive (c, l)-diversity.

TABLE 8
Algorithm 7: KDLD Sequence Generation Algorithm for Recursive (c, l)-Diversity

```
1  $P$ = the sensitive degree sequence of the original graph $G$;
2  Set $R$ = {};
3  while $|P| > 0$ do
4      Group $C = \{P[0]\}$;
5      int $d = P[0].d$;
6      Remove $P[0]$ out of $P$;
7      while $\neg$($C$ satisfies the Safety Grouping Condition) do
8          for $i = 0; i < |P|; i$++ do
9              if $P[i].d \equiv d$ then
10                 $C = C \cup \{P[i]\}$;
11                 Remove $P[i]$ out of $P$;
12                 break;
13             else
14                 if $P[i].s$ is not in the top $l-1$ appearance label set of $C$ then
15                     $C = C \cup \{P[i]\}$;
16                     Remove $P[i]$ out of $P$;
17                     break;
18         if $i == |P|$ then
19             $R = R \cup C$;
20             break;
21     Set the target degrees of elements in $C$ as corresponding nodes' mean degree;
22     Copy $C$ into $P^{new}$ if $C$ satisfies SG condition;
23 Assign the elements in $R$ to existing groups;
```

Theorem 2. *For any equivalent group C which satisfies recursive (c, l)-diversity (i.e., $f_1 < c(f_l + f_{l+1} + \cdots + f_m)$) and $\frac{f_1+1}{f_1(m-l+1)} < c$, after adding noise nodes into C with their sensitive labels setted by Algorithm 6, the new group still satisfies (c, l)-diversity.*

The detailed proof of this theorem can be found in Appendix II, available in the online supplemental material. We can generate the KDLD sequence where each same degree group in it satisfies $f_1 < c(f_l + f_{l+1} + \cdots + f_m)$ and $\frac{f_1+1}{f_1(m-l+1)} < c$ to achieve the recursive (c, l)-diversity requirement.

5.2 KDLD Sequence Generation for Recursive (c, l)-Diversity

We design Algorithm 7 (Table 8) to generate the $KDLD$ sequence for recursive (c, l)-diversity. The basic idea is to put the nodes with similar degrees together to minimize the cost $L(P, P^{new}) = \sum_{\forall u} |p_u.d - p_u^{new}.d|$. We call the condition that a same degree group C must satisfies the ***Safety Grouping Condition***, which requests: 1) $C \geq k$; 2) $f_1 < c(f_l + f_{l+1} + \cdots + f_m)$; 3) $\frac{f_1+1}{f_1(m-l+1)} < c$. Note when we use the pure edge-editing method to construct the published graph, the Safety Grouping Condition only contains the first two constraints. Given the degree sequence P, Each time, we create a new group C for $P[0]$ and remove $P[0]$ out of P. We use d to record the degree of $P[0]$. Then, we recursively

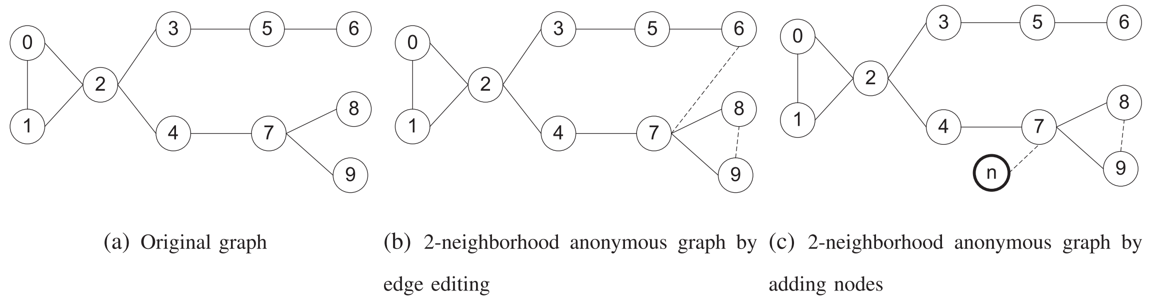

(a) Original graph
(b) 2-neighborhood anonymous graph by edge editing
(c) 2-neighborhood anonymous graph by adding nodes

Fig. 5. Publish a graph with neighborhood graph anonymity.

check the next element. If this element has degree d, we directly add it into the current group C. Otherwise, we check whether its label is a label that appears top $L-1$ times in C. If not, we add this element into C. After adding one element, we remove this element out of P and start check from the head of P since the label appearance numbers may be changed. We use this method to increase the size of C until C satisfies the Safety Grouping Condition. After getting a group, we copy it into P^{new} and start to construct the next group. If a group cannot reach the Safety Grouping Condition after checking all left nodes in P, we record the elements in this group into a set R. Finally, for each element in R, we assign it to an existing group if this group satisfies the Safety Grouping Condition after adding this node.

6 Discussion

For stronger graph protection models such as k-neighborhood anonymity [32], it is also helpful to preserve APL by carefully adding some nodes. A graph is k-neighborhood anonymous if: for every node there exist at least other $k-1$ nodes sharing isomorphic neighborhood graph. For example, a 2-neighborhood anonymity graph of Fig. 5a (shown in Fig. 5b) generated by edge editing can be published. If an attacker uses a node's neighborhood graph to reidentify it, he always gets at least two candidates. However, the distance between nodes 6 and 7 are changed from 5 to 1 in Fig. 5b. If we simply add one node, we can also generate a 2-neighborhood anonymous graph as shown in Fig. 5c. In Fig. 5c, the shortest distances between nodes are changed much smaller than Fig. 5b.

The basic procedure to generate a k-neighborhood anonymous [32] graph is: 1) Sort all nodes by their neighborhood graph size in descending order; 2) Recursively adjust two nodes' neighborhood graphs to be the same until a k-neighborhood anonymity graph is generated. When adjusting neighborhood graphs G_u and G_v with $|G_u| > |G_v|$ to be the same, new nodes should be introduced into G_v. An unanonymized node with the smallest degree has the highest priority to be added. The noise node adding strategy should be considered in this step to improve the utility of the published graph.

7 Experiments

In this section, we test our algorithm on the distinct l-diversity. We also conduct corresponding experiments for recursive (c,l)-diversity model. Both testing results show the effectiveness of our algorithm. Interested readers can find the testing results for recursive (c,l)-diversity model in the Appendix IV, available in the online supplemental material.

7.1 More Utilities

We first test how well the published graph represents the original graph. In order to measure the changes on the original graph, besides APL, we examine another two utilities: ***Average Change of Sensitive Label Path Length (ACSPL)*** and ***Remaining ratio of top influential users (RRTI)***.

1. $ACSPL$: In order to measure the connections between any two sensitive labels (including the same label), we define average path length between any two labels l_1 and l_2 as:

$$APL_{G,(l_1,l_2)} = \frac{\sum_{\forall n_i.s=l_1,n_j.s=l_2} d(n_i,n_j)}{\sum_{\forall n_i.s=l_1,n_j.s=l_2} 1}.$$

The average change of average sensitive value path length is defined as:

$$ACSPL_{G,G'} = \frac{\sum_{\forall l_1,l_2} Abs(APL_{G,(l_1,l_2)} - APL_{G',(l_1,l_2)})}{\binom{M}{2} + M},$$

where M is the number of unique sensitive attribute labels, $\binom{M}{2}$ is the number of two combinations of these M values. $\binom{M}{2} + M$ computes the number of all possible two combinations of the sensitive label set.

2. $RRTI$: One important data mining task on a graph are to find the top influential users (experts) in it. We test the remain ratio of top influential users to show how the published graph preserves this utility. Suppose the set of top 20 percent influential users in G is INF_G and the corresponding one in G' is $INF_{G'}$, the remaining ratio of the top $|INF_G|$ users is computed as:

$$RRTI = \frac{|INF_G \cap INF_{G'}|}{|INF_G|}.$$

The larger $RRTI$ is, the better the published graph preserves the information in the original graph. We use the PageRank algorithm [23] to compute the users' influential values.

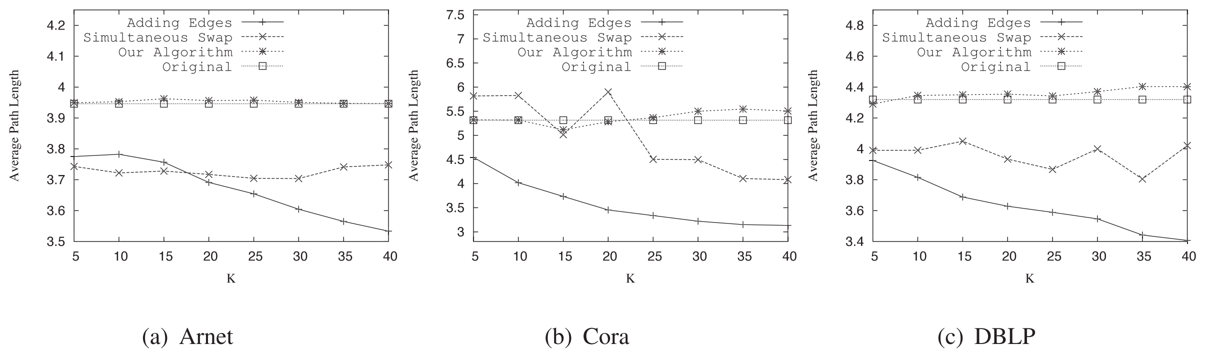


Fig. 6. Average path length for different k.

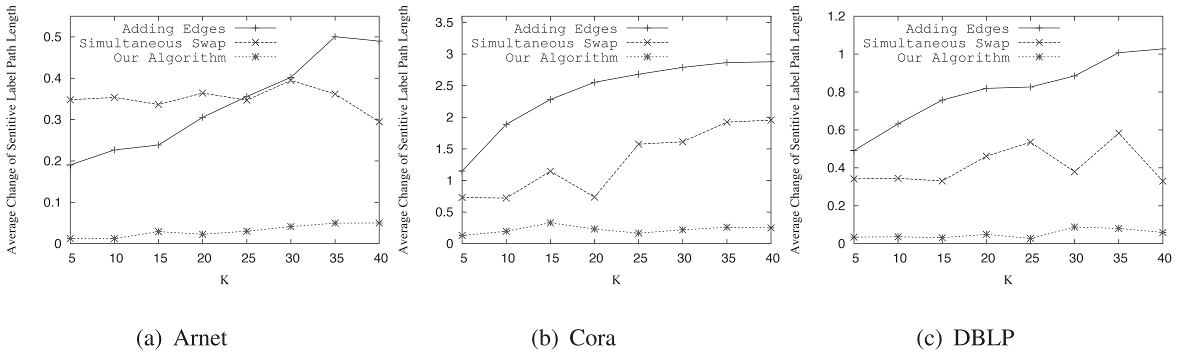


Fig. 7. Average change of sensitive label path length for different k.

7.2 Data Sets

In this experiment, we exam our algorithm on three real data sets: Arnet Data Set (6,000 nodes and 37,848 edges), Cora data set (2,708 nodes and 5,429 edges), and DBLP data set (6,000 nodes and 29,843 edges). Details of these data sets can be found in Appendix III, available in the online supplemental material.

7.3 Results

To demonstrate the effectiveness of our graph construction algorithm, we compare our work with two pure edge-editing graph construction algorithms: adding edges [18], simultaneous swap [18]. We test the $KD3D$ model for Cora and DBLP data sets, and $KD5D$ model for Arnet data set, we choose a larger l for Arnet data set since it contains more distinct sensitive labels. We generate the *KDLD* graph for each data set using the K-L-BASED sensitive degree sequence generation algorithm. Note here we use different graph construction algorithms to generate anonymized graphs for the same *KDLD* sequence.

Figs. 6a, 6b, and 6c are APL results of the three data sets, respectively, in terms of changing k values using different graph construction algorithms. From the results we can see that for all the three data sets, our noise node adding algorithm performs much better than two edge-editing algorithms. This is because our methods do not connect nodes far away. As a result, our method keeps the distance property well.

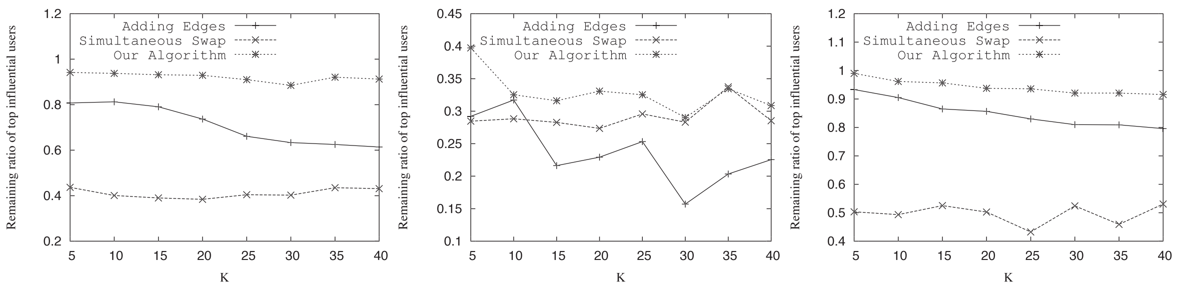
We show the $ACSPL$ results in Figs. 7a, 7b, and 7c. The less the ACSPL value is, the better a graph construction algorithm works. From the figures, we can observe that the curve of our noise node adding algorithm is quite low compared with adding edge method and simultaneous swap method. Our algorithm helps to preserve the distances between labels.

We show the $RRTI$ results in Figs. 8a, 8b, and 8c. The larger the RRTI value is, the better a graph construction algorithm works. Except simultaneous swap has competitive performance as our algorithm at two points ($k = 30$ and $k = 35$) of Cora, for all the other cases, our algorithm also works the best.

To summarize, after comparing APL, $ACSPL$, and $RRTI$ values, our adding noise node method performs much better than the two edge-editing algorithms. Making use of noise nodes to construct a graph helps to keep the utilities of the published graph.

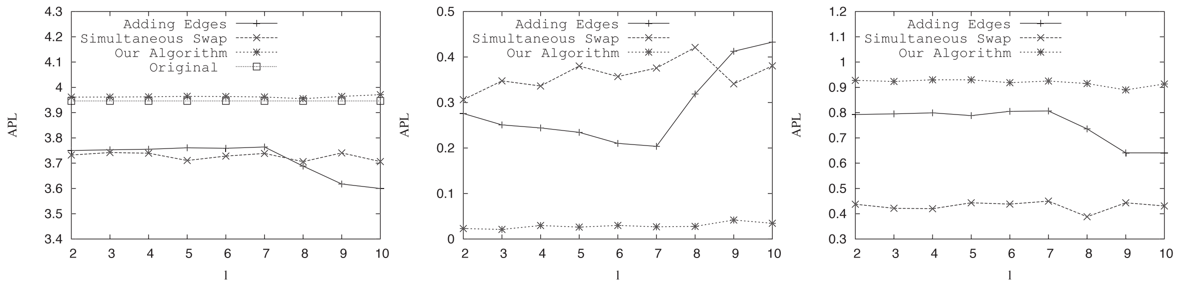
We also test the APL, $ACSPL$, and $RRTI$ results of the three data sets with different k based on L-K-BASED sensitive degree sequence generation algorithm. Very similar results as the Algorithm K-L-BASED are observed. Due to the page limitation, we do not show the test results here. When the k value becomes large compared with l value, the results of K-L-BASED algorithm and L-K-BASED algorithm become nearly the same. In our testing, the sum of degree change's difference between L-K-BASED and K-L-BASED sensitive degree sequence generation algorithm is within 5 percent. In our following experiments, we only show the results of our algorithm with the K-L-BASED sensitive degree sequence generation algorithm.

Since Cora and DBLP data sets do not contain the enough number of sensitive labels, in order to see the effect of l, we test $15DlD$ models of Arnet Data Set. The results are shown in Fig. 9. Similar results can be observed as the changing k case, our algorithm performs much better than the two edge-editing algorithms.



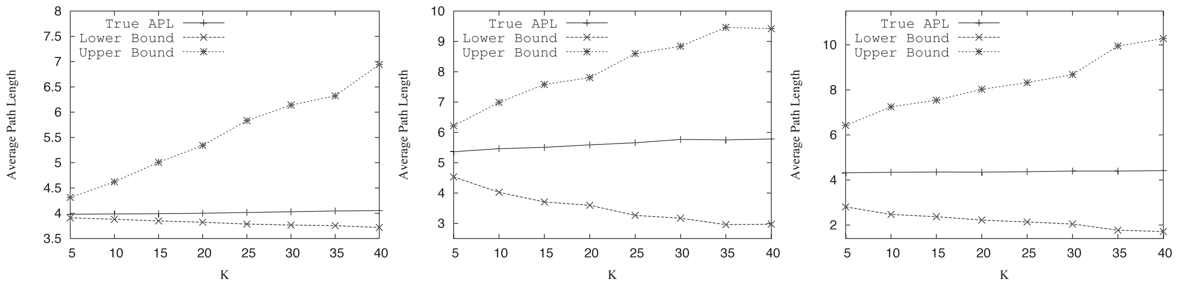
(a) Arnet (b) Cora (c) DBLP

Fig. 8. Remaining ratio of top influential users for different k.



(a) APL (b) ACSPL (c) RRTI

Fig. 9. Utilities of Arnet Data Set for different l.



(a) Arnet (b) Cora (c) DBLP

Fig. 10. APL and the estimated bounds.

7.4 More Experiments

Besides the quality of the published graph, we also test several other aspects of our algorithm.

7.4.1 Bounds of APL Change

In order to test the bounds derived in Section 4.7, we compute the bounds for these three data sets and compare the true APLs with them. The results are shown in Fig. 10.

7.4.2 Percentage of Noise Nodes

Fig. 11a shows the percentage of noise nodes added by our algorithm with different ks. In all cases, our algorithm add less than 7 percent noise nodes. Our method only put a small "burden" to achieve a much better effect comparing with pure edge-editing algorithms.

7.4.3 Change of Label Distribution

Fig. 11b shows the percentage of average label distribution change. The percentage of average label distribution change is calculated as:

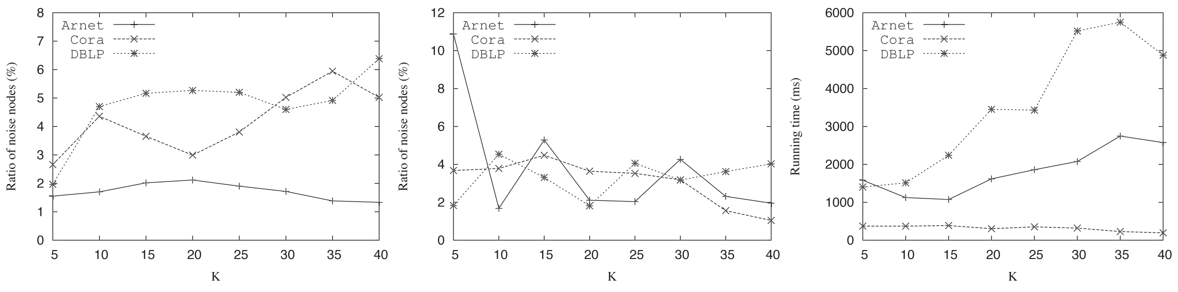
$$\frac{\sum_{i=1}^{M} \frac{|ratio_{l_i,G} - ratio_{l_i,G'}|}{ratio_{l_i,G}}}{M} \times 100\%,$$

where M stands for the number of unique node labels, $ratio_{l_i,G}$ stands for label l_i's percentage in G, and $ratio_{l_i,G'}$ stands for its percentage in G'. From the result, we can see although some noise nodes are added, the change of label distribution is very small. The maximum change ratio is 11 percent and in most cases, the change ratio is less than 6 percent. This change includes the labels that only appear a small number of times in the original graph. A small change to this kind of labels will cause a large distribution change.

7.4.4 Algorithm Efficiency

We record the running time of our algorithm for different k in Fig. 11c.[3] From the result we can see our algorithm is very efficient, the largest running time is less than 6,000 ms.

3. We test the running time of noise node adding algorithms on a PC computer running with the Microsoft Windows Vista operating system, which has Intel Core 2 Duo CPU with 2.53 GHz, 4.0 GB main memory.



(a) Percentage of noise nodes (b) Change of label distribution (c) Algorithm efficiency

Fig. 11. More experiments.

TABLE 9
The Filtering Results (Percent)

(1) Filtering by degrees

Dataset	K=5	10	15	20	25	30	35	40
Arnet	3.81	5.87	5.39	5.48	4.48	4.22	3.76	4.02
Cora	3.48	5.97	4.25	3.55	4.91	5.13	8.03	5.42
DBLP	4.35	9.16	8.96	8.43	8.24	7.64	8.7	9.43

(2) Filtering by CCs

Dataset	K=5	10	15	20	25	30	35	40
Arnet	7.55	7.58	8.14	8.04	8.11	6.84	5.65	5.71
Cora	3.99	8.23	4.77	4.14	4.93	6.44	7.77	6.09
DBLP	3.61	10.61	11.3	8.93	8.79	8.13	8.37	10.56

(3) Outliers by nodes' CCs

Dataset	K=5	10	15	20	25	30	35	40
Arnet	20.3	10.8	10.6	10.5	8.1	6.1	3.6	2
Cora	0	22.2	5	11.1	3.7	18.2	0	16.7
DBLP	16.67	25	0	10.71	3.57	8.93	3.85	9.38

(4) Outliers by nodes' influential values

Dataset	K=5	10	15	20	25	30	35	40
Arnet	6	13.4	8	15.9	13.9	16.5	1.9	7
Cora	17.6	24.5	19.2	20.8	11.5	21.7	4.6	21.3
DBLP	27.53	25.48	15.06	24.73	10.36	24.61	23.34	23.69

(5) Outliers by node neighbors' degrees

Dataset	K=5	10	15	20	25	30	35	40
Arnet	0	0	0.2	10.1	0.1	0.1	0	0
Cora	0	7.8	5	11.1	11.1	18.2	2	16.7
DBLP	1.39	0	3.85	9.55	10.71	7.14	7.69	0

(6) Outliers by node neighbors' CCs

Dataset	K=5	10	15	20	25	30	35	40
Arnet	6.5	6.6	7.4	16.5	6	4.3	5.9	4.1
Cora	13.1	17.1	7.5	18	10.4	18	7.1	18.8
DBLP	16.67	22	15.38	25	11.11	17.86	20.53	23.42

(7) Outliers by node neighbors' influential values

Dataset	K=5	10	15	20	25	30	35	40
Arnet	0	0	0.2	10	0.1	1	1.1	0.6
Cora	0	22.2	5	14.8	0	18.2	0.1	16.8
DBLP	23.83	5.12	24.92	16.69	21.71	21.52	10.26	12.5

7.4.5 New Added Nodes Hiding

Since each noise node n has a degree included in P^{new}, there at least exists k original node with the same degree as n. An attacker cannot determine whether a node is a noise node based on the degree information. For any noise node n, our algorithm always connects n with some nodes that are near to the node when n is created. As a result, all n's neighbors are from a small region of the graph. This is the best way to guarantee that any noise node's connection satisfies the basic characteristic [25], [28] of social network. That is, for any node, most neighbors of it should be close to each other.

In this section, we show the noise nodes are well mixed with original nodes by testing the ability that an attacker can filter out the noise nodes.

Test 1. In this testing, we make a strong assumption that the attacker knows the accurate characteristics of noise nodes and uses these characteristics to filter noise nodes out. We want to test whether the characteristics of noise nodes are special comparing to original nodes.

We first suppose the attacker already knows the degrees of all the noise nodes and uses this information to filter all the nodes with these degrees out. Table 9(1) shows the average percentage of real noise nodes in the filtered out nodes. From the result, we can see that our algorithm hides the noise nodes well, there exists a large portion of original nodes have the same degrees as noise nodes.

Clustering coefficient (CC) of a vertex is an important value [25], [28] to represent the characteristic of its neighborhood graph. It is defined as the actual number of edges between the vertex's directed neighbors divides the max possible number of edges between these directed neighbors. We suppose an attacker knows the CC values of all the noise nodes and do the filtering. The result is shown in Table 9(2), similar results as the degree filtering method can be observed. A large portion of original nodes have the same CCs as noise nodes. The result shows the connection of noise nodes are the same as most original nodes.

Test 2. There are some works [22], [9] to detect abnormal structures in graph databases. For each degree that contains noise nodes, we extract all the labeled neighborhood graphs of the nodes with this degree in the published graph. We use the tool in [9] to detect abnormal structures in the graph database formed by these neighborhood graphs. Eberle and Holder [9] designed graph-based approaches to uncovering anomalies in labeled graphs where the anomalies consist of unexpected entity/relationship deviations that resemble the nonanomalous behavior. They make use of the minimum description length principle and probabilistic approaches in their tool. In our experiment, the interesting result is, for all the cases, either no abnormal structure is found or the abnormal structures found out are all in the neighborhood graphs of normal nodes. The result shows the neighborhood graph of the noise nodes do not have any special characteristics for noise filtering purpose.

Test 3. In this testing, we enhance an attacker's background knowledge by the number of noise nodes in each same degree group. An attacker originally has the knowledge of some node's degrees. From our algorithm, he also knows we change the degrees of original nodes as less as possible. So, in the very worst case, for a same degree group C with degree d, the attacker may know the number of noise nodes in it. For a same degree group C, if there are m noise nodes in C, we follow the distance-based outliers detection [14], [16] method to extract m nodes out based on certain characteristics of each node. Distance-based outliers detection is a general method to detect outliers in a set. We divide C into two sets C_1 and C_2 with $|C_2| = m$ where the nodes in C_2 are detected as outliers. Suppose in the extracted nodes, there are r nodes which are real noise nodes, we compute the correct filtering ratio as $ratio = \frac{r}{m} \times 100\%$. We compute the average correct filtering ratio of all same degree groups which contain at least one noise node. We conduct two experiments:

- Filtering 1: For a node n in C, n is represented by its certain connection characteristic. We use two connection characteristics: CC and the influential value to do the filtering.
- Filtering 2: For a node n in C with degree d, n is represented by a sorted vector with size d which contains certain connection characteristic of n's neighbors. We use three connection characteristics: degree, CC, and the influential value to do the filtering.

Tables 9(3) and 9(4) show the average correct filtering ratio of Filtering 1. From the result, we can see the correct filtering ratios are very low. Even an attacker knows the number of noise nodes we added for each degree, it is still impossible for them to find these noise nodes. Tables 9(5), 9(6), and 9(7) show the average correct filtering ratio of Filtering 2. Similar results can also be observed. Most of the detected outliers are original nodes. The neighbors of noise nodes do not exhibit special connection characteristics either. The results show the noise nodes added by our algorithm have no special graph characteristics. They are well mixed with the original nodes in the graph.

From the experiments, we can see comparing with the current edge-editing-based algorithms, our noise node adding algorithm can generate a privacy preserving graph efficiently, which provides better utilities.

8 Related Work

Simply removing the identifiers in social networks does not guarantee privacy. The unique patterns, such as node degree or subgraph to special nodes, can be used to reidentify the nodes [15]. The attack that uses certain background knowledge to reidentify the nodes/links in the published graph is called "passive attack." There are two models proposed to publish a privacy preserved graph: edge-editing-based model [18], [32], [34], [6], [29] and clustering-based model [15], [30], [4], [7], [3]. The edge-editing-based model is to add or delete edges to make the graph satisfy certain properties according to the privacy requirements. Clustering-based model is to cluster "similar" nodes together to form super nodes. Each super node represents several nodes which are also called a "cluster." Then, the links between nodes are represented as the edges between super nodes which is called "super edges." Each super edge may represent more than one edge in the original graph. We call the graph that only contains super nodes and super edges as a clustered graph.

Most edge-editing-based graph protection models implement k-anonymity [26] of nodes on different background knowledge of the attacker. Liu and Terzi [18] defined and implemented k-degree-anonymous model on network structure, that is for published network, for any node, there exists at least other $k-1$ nodes have the same degree as this node. Zhou and Pei [32] considered k-neighborhood anonymous model: for every node, there exist at least other $k-1$ nodes sharing isomorphic neighborhoods. In paper [33], the k-neighborhood anonymity model is extended to k-neighborhood-l-diversity model to protect the sensitive node label. Zou et al. [34] proposed a k-Automorphism protection model: A graph is k-Automorphism if and only if for every node there exist at least $k-1$ other nodes do not have any structure difference with it. Cheng et al. [6] designed a k-isomorphism model to protect both nodes and links: a graph is k-isomorphism if this graph consists k disjoint isomorphic subgraphs. The sensitive attributes of nodes are protected by anatomy model [27] in a k-isomorphism graph. Ying and Wu [29] proposed a protection model which randomly changes the edges in the graph. They studied how random deleting and swapping edges change graph properties and proposed an eigenvalues oriented random graph change algorithm. All the edge-editing-based models prefers to generate a published graph with as less edge change as possible. Our model is also based on edge-editing method. The main difference of our work with other previous works is that besides as less change as possible, we guarantee the published graph preserves another important utility, the average path length

APL, which reflects the distortion of node relationships on the original graph.

For clustering-based models, since a clustered graph only contains super nodes and super edges, by making each cluster's size at least k, the probability to reidentify a user can be bounded to be at most $\frac{1}{k}$. Hay et al. [15] proposed a heuristic clustering algorithm to prevent privacy leakage using vertex refinement, subgraph, and hub-print attacks. Campan and Truta [4] discussed how to implement clustering when considering the lost of both node labels and structure information. Zheleva and Getoor [30] developed a clustering method to prevent the sensitive link leakage. Cormode et al. [7], [3] introduced (k,l)-clusterings for bipartite graphs and interaction graphs, respectively. Campan et al., [5] implemented a p-sensitive-k-anonymity clustering model which requests each cluster satisfy distinct l-diversity. Since a clustered graph only contains super nodes and super edges, to mine a clustered graph, people samples a group of graphs which are consisted with this clustered graph. Then, the mining task can be finished by mining each sampled graph and computing the average results. Since a user should do the sampling, the utility of the published graph does not have any guarantee and this user could never know how many samplings can guarantee to get a "well enough" result. Comparing with a clustering-based model, the benefit of our model is that we can guarantee to preserve some utilities of the published graph.

Besides the "passive attack," there's another type of attack on social networks, which is called "active attack." "Active attack" is to actively embed special subgraphs into a social network when this social network is collecting data. An attacker can attack the users who are connected with the embedded subgraphs by reidentifying these special subgraphs in the published graph. Backstrom et al. [1] described active attacks based on randomness analysis and demonstrated that an attacker may plant some constructed substructures associated with the target entities. One method to prevent the active attack is to recognize the fake nodes added by attackers and remove them before publishing the data. Shrivastava et al. [25] and Ying et al. [28] focused on a special active attack named Random Link Attack. Shrivastava et al. [25] proposed an algorithm that can identify fake nodes based on the triangle probability difference between normal nodes and fake nodes. Ying et al. [28] proposed another method, which uses spectrum analysis to find the fake nodes. To publish a graph that is potentially changed by Random Link Attack, the publisher can use a two step mechanism. First, the graph is filtered by the methods introduced by Backstrom et al. [1] or Shrivastava et al. [25]. Then, he can generate the published graph using our model from the filtered graph.

There are also two other works which focus on the edge weight protection in weighted graphs. Liu et al. [19] treated weights on the edges as sensitive labels and proposed a method to preserve shortest paths between most pairs of nodes in the graph. Das et al. [8] proposed a Linear Programming-based method to protect the edge weights while preserving the path of shortest paths. These two works focused on the protection of edge weights instead of nodes, which are different with our work.

Some works studied other attacks besides the "passive attack" and "active attack." Zheleva [31] analyzed the ability of an attacker to learn the unpublished attributes of users in an online social network when he uses the published attributes and relationships between users to do the data mining. Arvind [21] studied the ability of an attacker to reidentify nodes in an anonymized graph G_a when he knows a different graph G_{aux} whose membership partially overlaps with G_a. The attacker knows the mapping of some "seed nodes" between G_a and G_{aux}. Narayanan and Shmatikov [21] showed from these "seed nodes," a large portion of other nodes can be reidentified. These two attacks are different with the "passive attack" which we target on in this paper.

9 Conclusion

In this paper, we propose a k-degree-l-diversity model for privacy preserving social network data publishing. We implement both distinct l-diversity and recursive (c, l)-diversity. In order to achieve the requirement of k-degree-l-diversity, we design a noise node adding algorithm to construct a new graph from the original graph with the constraint of introducing fewer distortions to the original graph. We give a rigorous analysis of the theoretical bounds on the number of noise nodes added and their impacts on an important graph property. Our extensive experimental results demonstrate that the noise node adding algorithms can achieve a better result than the previous work using edge editing only. It is an interesting direction to study clever algorithms which can reduce the number of noise nodes if the noise nodes contribute to both anonymization and diversity. Another interesting direction is to consider how to implement this protection model in a distributed environment, where different publishers publish their data independently and their data are overlapping. In a distributed environment, although the data published by each publisher satisfy certain privacy requirements, an attacker can still break user's privacy by combining the data published by different publishers together [11]. Protocols should be designed to help these publishers publish a unified data together to guarantee the privacy.

Acknowledgments

This work is supported in part by Hong Kong RGC grants N_HKUST612/09, National Grand Fundamental Research 973 Program of China under Grant 2012CB316200, and NSFC 619308, US National Science Foundation (NSF) through grants CNS-1115234, IIS-0914934, and CNS-0747247. The DBLP data set is based on the Proximity DBLP database prepared by the Knowledge Discovery Laboratory, University of Massachusetts Amherst.

References

[1] L. Backstrom, C. Dwork, and J.M. Kleinberg, "Wherefore Art Thou r3579x?: Anonymized Social Networks, Hidden Patterns, and Structural Steganography," *Proc. Int'l Conf. World Wide Web (WWW),* pp. 181-190, 2007.

[2] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science,* vol. 286, pp. 509-512, 1999.

[3] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava, "Class-Based Graph Anonymization for Social Network Data," *Proc. VLDB Endowment*, vol. 2, pp. 766-777, 2009.

[4] A. Campan and T.M. Truta, "A Clustering Approach for Data and Structural Anonymity in Social Networks," *Proc. Second ACM SIGKDD Int'l Workshop Privacy, Security, and Trust in KDD (PinKDD '08)*, 2008.

[5] A. Campan, T.M. Truta, and N. Cooper, "P-Sensitive K-Anonymity with Generalization Constraints," *Trans. Data Privacy*, vol. 2, pp. 65-89, 2010.

[6] J. Cheng, A.W.-c. Fu, and J. Liu, "K-Isomorphism: Privacy Preserving Network Publication against Structural Attacks," *Proc. Int'l Conf. Management of Data*, pp. 459-470, 2010.

[7] G. Cormode, D. Srivastava, T. Yu, and Q. Zhang, "Anonymizing Bipartite Graph Data Using Safe Groupings," *Proc. VLDB Endowment*, vol. 1, pp. 833-844, 2008.

[8] S. Das, O. Egecioglu, and A.E. Abbadi, "Privacy Preserving in Weighted Social Network," *Proc. Int'l Conf. Data Eng. (ICDE '10)*, pp. 904-907, 2010.

[9] W. Eberle and L. Holder, "Discovering Structural Anomalies in Graph-Based Data," *Proc. IEEE Seventh Int'l Conf. Data Mining Workshops (ICDM '07)*, pp. 393-398, 2007.

[10] K.B. Frikken and P. Golle, "Private Social Network Analysis: How to Assemble Pieces of a Graph Privately," *Proc. Fifth ACM Workshop Privacy in Electronic Soc. (WPES '06)*, pp. 89-98, 2006.

[11] S.R. Ganta, S. Kasiviswanathan, and A. Smith, "Composition Attacks and Auxiliary Information in Data Privacy," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 265-273, 2008.

[12] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis, "Fast Data Anonymization with Low Information Loss," *Proc. 33rd Int'l Conf. Very Large Data Bases (VLDB '07)*, pp. 758-769, 2007.

[13] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis, "A Framework for Efficient Data Anonymization Under Privacy and Accuracy Constraints," *ACM Trans. Database Systems*, vol. 34, pp. 9:1-9:47, July 2009.

[14] J. Han, *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, Inc., 2005.

[15] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis, "Resisting Structural Re-Identification in Anonymized Social Networks," *Proc. VLDB Endowment*, vol. 1, pp. 102-114, 2008.

[16] E.M. Knorr, R.T. Ng, and V. Tucakov, "Distance-Based Outliers: Algorithms and Applications," *The VLDB J.*, vol. 8, pp. 237-253, Feb. 2000.

[17] N. Li and T. Li, "T-Closeness: Privacy Beyond K-Anonymity and L-Diversity," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE '07)*, pp. 106-115, 2007.

[18] K. Liu and E. Terzi, "Towards Identity Anonymization on Graphs," *SIGMOD '08: Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 93-106, 2008.

[19] L. Liu, J. Wang, J. Liu, and J. Zhang, "Privacy Preserving in Social Networks against Sensitive Edge Disclosure," Technical Report CMIDA-HiPSCCS 006-08, 2008.

[20] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "L-Diversity: Privacy Beyond K-Anonymity," *ACM Trans. Knowledge Discovery Data*, vol. 1, article 3, Mar. 2007.

[21] A. Narayanan and V. Shmatikov, "De-Anonymizing Social Networks," *Proc. IEEE 30th Symp. Security and Privacy*, pp. 173-187, 2009.

[22] C.C. Noble and D.J. Cook, "Graph-Based Anomaly Detection," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '03)*, pp. 631-636, 2003.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd, "The Pagerank Citation Ranking: Bringing Order to the Web," *Proc. World Wide Web Conf. Series*, 1998.

[24] K.P. Puttaswamy, A. Sala, and B.Y. Zhao, "Starclique: Guaranteeing User Privacy in Social Networks Against Intersection Attacks," *Proc. Fifth Int'l Conf. Emerging Networking Experiments and Technologies (CoNEXT '09)*, pp. 157-168, 2009.

[25] N. Shrivastava, A. Majumder, and R. Rastogi, "Mining (Social) Network Graphs to Detect Random Link Attacks," *Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE '08)*, pp. 486-495, 2008.

[26] L. Sweeney, "K-Anonymity: A Model for Protecting Privacy," *Int'l J. Uncertain. Fuzziness Knowledge-Based Systems*, vol. 10, pp. 557-570, 2002.

[27] X. Xiao and Y. Tao, "Anatomy: Simple and Effective Privacy Preservation," *Proc. 32nd Int'l Conf. Very Large Databases (VLDB '06)*, pp. 139-150, 2006.

[28] X. Ying, X. Wu, and D. Barbara, "Spectrum Based Fraud Detection in Social Networks," *Proc. IEEE 27th Int'l Conf. Very Large Databases (VLDB '11)*, 2011.

[29] X. Ying and X. Wu, "Randomizing Social Networks: A Spectrum Preserving Approach," *Proc. Eighth SIAM Conf. Data Mining (SDM '08)*, 2008.

[30] E. Zheleva and L. Getoor, "Preserving the Privacy of Sensitive Relationships in Graph Data," *Proc. First SIGKDD Int'l Workshop Privacy, Security, and Trust in KDD (PinKDD '07)*, pp. 153-171, 2007.

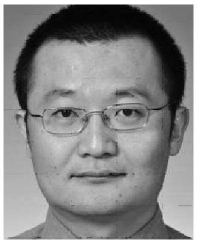
[31] E. Zheleva and L. Getoor, "To Join or Not to Join: The Illusion of Privacy in Social Networks with Mixed Public and Private User Profiles," *Proc. 18th Int'l Conf. World Wide Web (WWW '09)*, pp. 531-540, 2009.

[32] B. Zhou and J. Pei, "Preserving Privacy in Social Networks Against Neighborhood Attacks," *Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE '08)*, pp. 506-515, 2008.

[33] 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*, vol. 28, pp. 47-77, 2011.

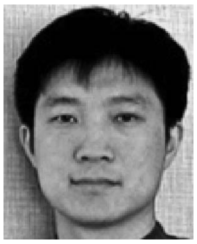
[34] L. Zou, L. Chen, and M.T. Özsu, "K-Automorphism: A General Framework for Privacy Preserving Network Publication," *Proc. VLDB Endowment*, vol. 2, pp. 946-957, 2009.

Mingxuan Yuan received the PhD degree from the Department of Computer Science and Engineering at Hong Kong University of Science and Technology, China, in 2011. He is a researcher in the Huawei Noah Ark Lab, Hong Kong. His research interests include privacy problems of social networks.



Lei Chen is currently an associate professor in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include social networks, probabilistic and uncertain data, cloud data processing, and graph data. He is a member of the IEEE and the IEEE Computer Society.

Philip S. Yu is a professor in the Department of Computer Science at the University of Illinois at Chicago and also holds the Wexler Chair in Information Technology. He spent most of his career at the IBM Thomas Watson Research Center and was manager of the Software Tools and Techniques Group. His research interests include data mining, Internet applications and technologies, database systems, multimedia systems, parallel and distributed processing, and performance modeling. He is an associate editor of the *ACM Transactions on the Internet Technology* and the *ACM Transactions on Knowledge Discovery from Data*. He is a fellow of the ACM and the IEEE.



Ting Yu is currently an associate professor in the Department of Computer Science at North Carolina State University. His research interests include trust management and privacy preservation in open systems, and database management systems. He is a member of the ACM.