Personalized Privacy Protection in Social Networks

Mingxuan Yuan Lei Chen
The Hong Kong University of Science & Technology
{csyuan, leichen}@ust.hk

Philip S. Yu
University of Illinois at Chicago
psyu@cs.uic.edu

ABSTRACT

Due to the popularity of social networks, many proposals have been proposed to protect the privacy of the networks. All these works assume that the attacks use the same background knowledge. However, in practice, different users have different privacy protect requirements. Thus, assuming the attacks with the same background knowledge does not meet the personalized privacy requirements, meanwhile, it looses the chance to achieve better utility by taking advantage of differences of users' privacy requirements. In this paper, we introduce a framework which provides privacy preserving services based on the user's personal privacy requests. Specifically, we define three levels of protection requirements based on the gradually increasing attacker's background knowledge and combine the label generalization protection and the structure protection techniques (i.e. adding noise edge or nodes) together to satisfy different users' protection requirements. We verify the effectiveness of the framework through extensive experiments.

1. INTRODUCTION

Nowadays, more and more people join multiple social networks on the Web, such as Facebook, Linkedin, and Livespace, to share their own information and at the same time to monitor or participate in different activities. Meanwhile, the information stored in the social networks are under high risk of attack by various malicious users, in other words, people's privacy could be easily breached via some domain knowledge. Thus, as a service provider, such as Facebook and Linkedin, it is essential to protect users' privacy and at the same time provide "useful" data. Many proposals have been proposed to protect the privacy of published social networks including some clustering-based approaches [6][16][3] and graph editing methods [8][15][17][19]. However, all the previous works have overlooked a very important fact, that is, different users may have different privacy preferences. Therefore, providing the same level privacy protection to all the users may not be fair and in addition may cause the published social network data useless.

To address the shortcomings of single level privacy protection, in this work, we define different levels of protection for users and incorporate them into the same published social network. In fact, current social network websites allow each user to set how much of his information can be observed by others. For example, in Facebook, a user can set what part of his profile or his connection information can be viewed by others. A user can have a clear estimation of the knowledge that an attacker can know about him. The knowledge an attacker uses to find the privacy information of a user is called the background knowledge. To provide different levels of privacy protection, we allow users to set personalized privacy requirements based on their own assumptions about the attacker's background knowledge. Specifically, for a node u in a published labeled graph[1], starting from the weakest background knowledge that an attacker only knows u's label information without any structure information, we define three levels of attacks to u by gradually increasing the strength of the attacker's background knowledge:

Level 1: An attacker only knows u's labels. For example, an attacker knows Bob is a 26-year old guy;

Level 2: An attacker knows u's labels and degree. For example, an attacker knows Bob is a 26-year old guy with degree 3;

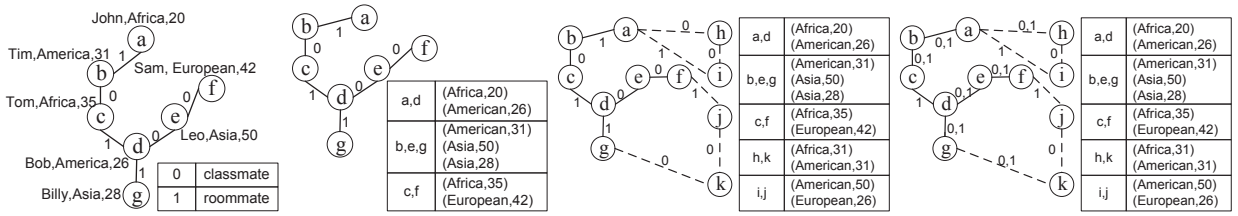
Level 3: An attacker knows u's labels, degree and the labels on the edges adjacent to u. For example, an attacker knows Bob is a 26-year old guy with degree 3 and Bob's three connections' types are classmate, roommate, roommate;

We extract these three levels of background knowledge due to the fact that the three kinds of settings are also supported by Facebook. We noticed that there exist much stronger attacks such as knowing the neighborhood of u [18] with label information. However, in this work, our focus is to demonstrate a framework which can publish a social network to satisfy different levels of privacy protection requirements, thus, we will not enumerate all the possible attacks. In the rest part of this paper, we use "Level x's background knowledge" to represent the corresponding background knowledge used in Level x's attack.

To achieve the goal of protecting each user to the level that is equal to or stronger than his own setting, we can simply protect all users in a graph to the strongest protection level requested by the users. This naive method assumes that the worst case happens to each node, which decreases the utility of the published graph. Here, the utility refers to the closeness between the published graph and the original graph. For example, Figure 1(a) is a graph which contains seven nodes. Each node has three labels: name, area, and age. Assume that we have two protection objectives for publishing this graph, which are: the probability that an attacker finds a person P is node u in the published graph (a.k.a. node re-identification) should be less than 50%, and the probability that an attacker finds person P_1 and P_2 have a connection (a.k.a. link re-identification) is less than 50%. For Figure 1(a), we could publish Figure 1(b) which provides Level 1 protection for all the nodes according to

[1] In this paper, we also call social network as graph for the sake of simplicity and each user is represented as a node in the graph.

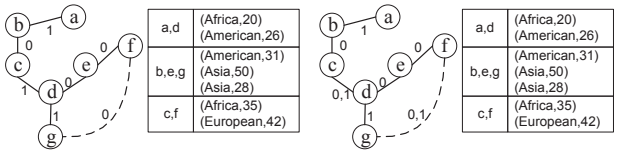
Proceedings of the VLDB Endowment, Vol. 4, No. 2



(a) Original Graph (b) Level 1 Protection (c) Level 2 Protection (d) Level 3 Protection

Figure 1: Naive Protection

the naive method discussed above. Nodes are divided into three groups ((a, d), (b, e, g) and (c, f)) as shown in the table of Figure 1(b). Since each *group's size* (the number of nodes in a group) is equal to or larger than 2, the success ratio of re-identifying a node using Level 1's knowledge is less than 50%. There's no edge between two nodes in the same group and the number of edges between the nodes in two groups is at most 3, which guarantees that the probability to find two nodes having a connection less than 50%. We give a comprehensive analysis of this probability in Section 3. The graphs that provide Level 2 and Level 3 protection are shown in Figure 1(c) and Figure 1(d) respectively. In Figure 1(c) no node/link could be re-identified with confidence larger than 50% even under an attack using nodes' label and degree information. We generalize the labels (on edges) in Figure 1(c) and guarantee that the attacker could not re-identify the nodes/links even with extra knowledge about edge labels as shown in Figure 1(d).



(a) Level 2 Protection (b) Level 3 Protection

Figure 2: Personalized Protection

Figure 1(c) and 1(d) assume all the users either need Level 2 or Level 3 protections, respectively. However, as we stated earlier, within the same social network, different users may have different requests on privacy protection levels. For example, if only Leo and Tom need Level 2 protection, Figure 2(a) can satisfy all users' privacy settings. In Figure 2(a), the nodes, c and f, have the same node labels and degree as Tom in Figure 1(a). This guarantees that the re-identification probability of Tom is 50%. Similarly, the number of candidate nodes for Leo is 3 which is also bigger than $2 = \frac{1}{50\%}$. We have to add 4 nodes and 6 edges to Figure 1(a) to derive Figure 1(c), but only need to add 1 edge to get Figure 2(a). Comparing the changes made to the original graph, Figure 2(a) is much more similar to the original graph than Figure 1(c). If Leo needs Level 2 protection and Tom needs Level 3 protection, Figure 2(b) can be published, which only generalizes 2 edges' labels. However, Figure 1(d) generalizes 7 edges' labels to make all nodes have Level 3 protection. Figure 2(b) is also more similar to Figure 1(a) than Figure 1(d). The two examples show that utility of the published graph increases when allowing personalized protections.

To achieve personalized privacy protection in social networks, in this work, we design different methods for different privacy requirements. Specifically, For Level 1 protection, we use node label generalization. For Level 2 protection, we combine the noise node/edge adding methods based on the protection at Level 1. For Level 3 protection, we further use the edge label generalization to achieve the protection objectives.

The rest of the paper is organized as follows: Section 2 defines the problems. Sections 3 - 5 describe the details of the protection framework design. We report the experimental results on some data sets in section 6. Comparison of our work with previous proposals is given in Section 7, followed by the conclusion in Section 8.

2. PROBLEM DEFINITION

In this paper, we focus on the privacy preserving problem for an un-weighted graph with labels on both nodes and edges[2]. Each node in the graph has several labels, which represent the attributes of the node. Each edge in the graph has one label, which represents the type of the edge. Since one node may have several labels, we call the labels on node u as u's *label list*. We use $G(V, E)$ to simply represent the original graph where V stands for the node set and E stands for the edge set. We design a framework that allows users to set three different privacy protection requirements introduced in Section 1. The protection objectives guaranteed by this framework are: Given a constant k,

1. For each node u, the probability that an attacker re-identifies u is at most $\frac{1}{k}$. For the attacker, re-identifying u is to find which node is u in the published graph using certain background knowledge about u;
2. For any edge e in the published graph, the probability that an attacker identifies a node u_x involved in e is at most $\frac{1}{k}$;
3. For any two nodes u_x and u_y, the probability that an attacker identifies these two nodes having a connection is at most $\frac{1}{k}$. For example, the probability that the attacker concludes Tim and Tom have a connection from the published graph should be equal to or less than $\frac{1}{k}$;

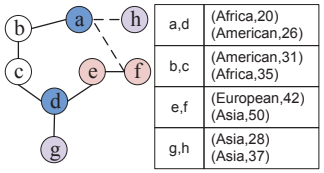


Figure 3: Generalizing after changing structure

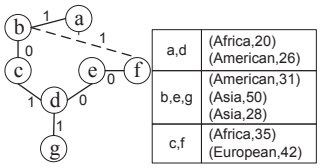


Figure 4: Degree anonymous problem

There are many proposals to avoid node re-identification from structural attacks[3] using graph editing techniques, such as *k*-degree anonymous[8], *k*-neighborhood anonymous[17], *k*-automorphism[19]. However, in this work, other than node re-identification, we would like to protect the connections (links) between nodes as stated in the above protection objectives. Thus, simply combining the graph editing techniques with micro data protection methods will not offer satisfactory results.

Take *k*-automorphism as an example, *k*-automorphism requires each node having at least *k-1* other nodes with the same structure in the published graph. Directly generalizing the labels on the nodes with same structure cannot protect the connection information well. For example, for Figure 1(a), if ignoring the edge labels,

[2]The formal definition of the graph model is shown in the Appendix A

[3]Structural attack stands for the attack based on connection information

a 2-automorphism graph can be constructed as shown in Figure 3 (nodes with the same color have the same structure). Figure 3 is first constructed by changing the graph to be a 2-automorphism graph based on structure information, then generalizing labels of nodes that have the identical structure to be the same. When an attacker uses the Level 2's background knowledge to re-identify a node, there are always two candidates. However, some connection information is still released. If the attacker knows *Tim* is 31 years old from American and *Tom* is 35 years old from Africa, he can find *Tim* and *Tom* has a connection with probability 100% from Figure 3, which is exactly the information that *Tim* and *Tom* do not want others to know. While in Figure 1(d), *Tim* is one of b, e and g. *Tom* is one of c and f. There are two edges between (b, e, g) and (c, f), thus the probability that an attacker concludes *Tim* and *Tom* has an edge is $\frac{2}{3\times 2} = 33\%$. The above example indicates that satisfying different levels of privacy protection requirements is not trivial. In fact, we have proven the problem to implement the personalized protections is NP-hard even with a simple optimal objective.

THEOREM 1. *Suppose cost = $\sum$node label generalization cost + $\sum$edge label generalization cost. For a graph G, the problem to create a graph G_A that satisfies each user's personal privacy requirement with minimum cost is a NP-hard problem*[4].

The proof can be found in Appendix B. The three levels of background knowledge are defined from the weakest to the strongest, a user who needs a higher level protection by default needs the lower level protections. So, in our framework, we implement the personalized protections from the lowest level to the highest level. In the rest part, we use set_{l_1} to represent the set of nodes that needs Level 1 or stronger protection, set_{l_2} to represent the set of nodes that needs Level 2 or stronger protection, and set_{l_3} to denote the set of nodes that needs Level 3 protection.

3. LEVEL 1 PROTECTION

The Level 1's background knowledge is about node label list, thus, we can use a generalization method to provide Level 1 protection. Simply speaking, generalization is to group nodes and make the nodes within each group have one generalized node label list. In order to keep the node labels distribution of each group in the published graph unchanged, we do not generalize all the label lists to be the same for a group of nodes. Instead, for each group, we publish all the label lists of the nodes in the group without the mapping[5]. Figure 1(b) gives an example of the published graph after generalization the graph in Figure 1(a). This method is the same as the Anatomy for tabular data proposed by Xiao et al. [14] and has the same effect as the full list pattern permutation proposed by Cormode et al.[2].

In order to achieve Level 1 protection, we need to divide all nodes into groups and these groups must guarantee the three privacy objectives are satisfied. As shown in [2], if each group's size is at least k, the first objective is achieved. To satisfy objectives (2) and (3), the following condition[2] must be satisfied:

THEOREM 2. *A division of nodes V into groups satisfies objectives (2) and (3) if for any node $u \in V$ and any group $g \subset V$:*

- $\forall (u, w), (u, z) \in E : w \in g \wedge z \in g \Rightarrow z = w$.
- $\forall (u, w) \in E : u \in g \wedge w \in g \Rightarrow u = w$.

The first condition constrains that any two nodes in the same group do not connect to a same node. The second condition constrains that no edges within a group. Theorem 2 ensures that the two objectives (2) and (3) are satisfied, but it is too restrictive. The following Theorem states that we can obtain the groups that satisfy objectives (2) and (3) easily by relaxing the constraints.

THEOREM 3. *Objectives (2) and (3) in Level 1 can be guaranteed, if*

- $\forall (v, w) \in E : v \in g \wedge w \in g \Rightarrow v = w$.
- $\forall$ *group* g_x, g_y, x *is the number of edges between* g_x *and* g_y, $x \leq \frac{|g_x||g_y|}{k}$.

The first condition constrains no edge within a group and the second condition constrains the number of edges between any two groups. The proof of this theorem is shown in Appendix C. The conditions in Theorem 3 is less restrictive than those in Theorem 2, which depend on the sparse property of social networks. As a consequence, groups are easier to be found based on the conditions in Theorem 3. For example, for Figure 1(a), if using the conditions in Theorem 2, no solution can be found. If using the conditions in Theorem 3, Figure 1(b) can be constructed. We call the two conditions in Theorem 3 as "***the Safety Grouping Condition (SGC)***".

Algorithm 1: Generate safe groups

```
1  while |V| > 0 do
2      u^s = the node with maximum degree in V;
3      V = V − u^s;
4      group g = new group {u^s};
5      C = C ∪ {g};
6      while |g| < k do
7          Set candidates = {};
8          for Each node u in V do
9              if Adding u into g does not violate SGC then
10                 candidates = candidates ∪ {u};
11         if |candidates| > 0 then
12             for Each node u in candidates do
13                 u.cost = the cost to add u into g;
14             u' = u in candidates with minimum u.cost;
15             g = g ∪ {u'}; V = V − {u'};
16         else
17             break;
18     for each g in C with |g| < k do
19         C = C − {g};
20         for each node u in g do
21             g' = the group in C with the min. cost to add u under SGC;
22             g' = g' ∪ {u};
```

Algorithm 1 computes the groups with size at least k. The algorithm targets on generating the groups under *SGC* with the minimum cost defined in Theorem 1. This algorithm has a similar structure as the algorithms in [3][2] which also group a graph's nodes with certain cost function. Each time, we select a node with the maximum degree that has not been grouped and create a new group for this node (line 2-5). Then in line 6-18, we repeatedly add nodes into the group with the minimum cost (estimated) under *SGC* until the group reaches size k. For any group which cannot reach size k, in line 19-23, we first delete this group. Then for each node in the group, we add it into a suitable group which has the minimum cost for the insertion operation.

Finally, Algorithm 1 gets the node group set C. Then, based on C, a graph like Figure 1(b) can be generated, which offers Level 1 protection for all the nodes. We use G_{L_1} to represent the generated graph at Level 1 protection.

4. LEVEL 2 PROTECTION

At Level 1, the *label-node* mapping is hidden by generalization. However, if an attacker knows both the label and the degree information of some nodes, he can still successfully re-identify some

[4]Here the cost stands for the sum of label generalization levels. For example, in Figure 1(d), if let the labels in the group {a,d} to be the same, they become [(Africa, American), 2*], thus the node label generalization cost of this group is 4.

[5]The generalization cost can still be computed as the sum of label generalization levels when making all the labels of the nodes in each group to be the same.

nodes in G_A. For example, if an attacker knows Bob's degree is 3, he can immediately find that node d is Bob in Figure 1(b). Therefore, in order to achieve the three protection objectives, the node re-identification using node label and degree information should be avoided for the nodes in set_{l_2}.

We generate a new published graph based on G_{L_1}. The target is to make sure that for each node in set_{l_2}, there are at least another *k-1* nodes in G_A identical to this node with respect to the attacker's background knowledge. At Level 2, we assume that an attacker also uses degree information, thus, the *k*-degree anonymous model [8] is needed to prevent the attack. *k*-degree anonymous model guarantees that the degree of each node in set_{l_2} appears at least k times in the published graph. However, when node label information is also obtained by the attacker, the nodes having the same degree cannot be arbitrarily selected. For example, the graph G' in Figure 4 is a graph generated from Figure 1(b) which satisfies 2-degree anonymous model. In G', each degree appears at least twice. Assume that an attacker knows Tim is from Africa with degree 1, since only nodes a and g have degree 1 in G', g cannot have the label "Africa", the attacker immediately knows a is Tim. This situation happens since the node label lists still provide some information, although they have already been generalized at Level 1. To implement *k*-degree anonymous model based on G_{L_1}, for any node u in set_{l_2}, there should exist k nodes with the same degree as u and having u's label list. In this paper, we call a graph that satisfies the above property ***k-degree anonymous graph***. We can construct a *k*-degree anonymous graph by making the nodes in the same group have the same degree if that group contains nodes in set_{l_2}.

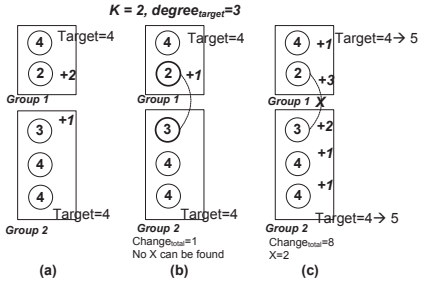


Figure 5: The example of degree anonymous process

In order to guarantee the privacy objectives (2) and (3), during the process of modifying G_{L_1} to a *k*-degree anonymous graph, *SGC* should be maintained. The k-degree anonymous graph generation method introduced in [8] does not satisfy the safe grouping constraint, thus we design a noise node/edge adding algorithm to construct the *k*-degree anonymous graph under *SGC*. The algorithm contains two steps:

- Generate a *k*-degree anonymous graph by adding nodes/edges under *SGC*;
- Since the new created nodes/edges do not have labels, assign labels to them.

We call the newly created nodes/edges "noise nodes/edges", next we discuss the detailed steps of adding noise nodes/edges.

4.1 Adding nodes/edges into G_{L_1}

To construct a k-degree anonymous graph, we first make the nodes in the same group have the same degree if that group contains nodes in set_{l_2} by adding nodes/edges. Specifically, we use the following three steps to perform the node/edge adding.

The first step is to set a target degree for each node. For a node u, if the group g that u belongs to contains at least one node in set_{l_2}, we set the target degree of u to be the largest degree in g. Otherwise, we set u's target degree as its original degree. For a group g that contains at least one node in set_{l_2}, since all the nodes in g need to have the same target degree, we also call this target degree as g's target degree ($g.target$). For example, in Figure 5(a), the number in each node stands for the degree of it. Then the target degrees of the two groups are set as 4. The number beside each node is the number of degree this node needs to be increased.

After each node has a target degree, the second step is to randomly create edges between the nodes which need to increase their degrees under *SGC*. For example, an edge may be created between two nodes as shown in Figure 5(b). If all the nodes reach their target degrees, we get a *k*-degree anonymous graph and finish the construction. Otherwise, we continue to construct the graph by adding some noise nodes, which is the next step.

In the third step, we add noise nodes to enable all the nodes in G_{L_1} have their target degrees by connecting them with noise nodes under *SGC*. When we add noise nodes, these noise nodes should be involved in certain groups to make them indistinguishable from the original nodes in G_{L_1}. In order to achieve this, we can let the noise nodes form groups with size k themselves, meanwhile, we need to hide the noise nodes by making their degree to a pre-selected value $degree_{target}$. $degree_{target}$ is the degree that the maximum number of groups of size k in G_{L_1} have[6].

Next, we first introduce Algorithm 2 which can make all the nodes in G_{L_1} reach their target degrees using several groups of same degree noise nodes. Then we prove an important property of this algorithm about the number of added noise nodes and the target degrees. Based on this property, we finally show how to set input parameters of Algorithm 2 to make all noise nodes exactly have degree $degree_{target}$ without violating *SGC*. In this part, we'll use two constants: $Change_{total}$ and $Change_{max}$. $Change_{total}$ represents the sum of degree needs to be increased for all the nodes in G_{L_1}. $Change_{max}$ stands for the maximum sum of degree needs to be increased in each group of G_{L_1}.

Algorithm 2 has four inputs: graph G_{L_1}, groups C, privacy parameter k and X, where X is the number of noise node groups that will be used to construct the new graph. In Algorithm 2, we create X groups of noise nodes and arrange them as an $X \times k$ array A. Each row of this array is a group of noise nodes. We use $A[i][j]$ to represent the noise node at position $[i, j]$ in A. New edges are created between the noise nodes in A and the nodes whose degrees need to be increased in G_{L_1}. Each time, we let all the nodes in one group reach their target degrees by creating edges from the nodes in this group to the noise nodes in A in a column by column style (line 4-12). Column by column means if an edge is connected with the noise node $A[i][j]$, then the next edge created will connect to $A[i+1][j]$. If $A[i+1][j]$ does not exist ($i+1 == X$), the edge is connected to the first noise node in the next column ($A[0][(j+1)\%k]$). The first created edge is linked to $A[0][0]$. After we complete these steps, all the nodes in G_{L_1} reach their target degrees. It is easy to see all the noise nodes either have the degree $d = \left\lfloor \frac{Change_{total}}{X \times k} \right\rfloor + 1$ or $(d-1)$ now. To let all the noise nodes have the same degree d, next we use the following method to build connections between noise nodes: 1) Select two rows with the maximum number of $(d-1)$ degree nodes in A; 2) Randomly select two nodes with the degree $(d-1)$ in each row and create an edge between them; 3) Repeat the above two steps until all the rows only contain nodes with degree d.

Let's use an example to show how Algorithm 2 works. In Figure 6, $Change_{total} = 8$, $k = 2$ and $X = 2$. Algorithm 2 first creates two noise node groups and arranges them as a 2×2 ($X \times k$) array.

[6]If all the noise nodes have degree $degree_{target}$, they are hidden into those groups who have the maximum number of original groups with the same degree. This makes the noise nodes mixed with as many original nodes as possible to avoid the filtering.

The number in each noise node represents this noise node's id in the 2×2 array. Then Algorithm 2 begins to create edges between the nodes in G_{L_1} and the noise nodes. The number on each edge stands for the adding order of this edge. The edges 1-8 are created sequentially. After doing this, all the nodes in G_{L_1} reach their target degrees. Here $Change_{total} = 8$, $d = \left\lfloor \frac{Change_{total}}{X \times k} \right\rfloor + 1 = 3$. All the noise nodes have degree 2 now (In this example, all noise nodes have degree $d-1$. In normal case, some noise nodes may have degree d). Algorithm 2 finally adds edges 9 and 10 to make all the noise nodes have the same degree 3.

Algorithm 2: Degree Anonymizing Algorithm with X

```
Create X × k new nodes and store them in a X × k array A;
Set each line of nodes in A as a new group;
int i = 0, j = 0;
while ¬(each node in G_{L_1} reaches its target degree) do
    Random select a g ∈ C that ∃ u ∈ g, u.degree < g.target;
    while ¬(each node in g reaches its target degree) do
        Random select a u ∈ g with u.degree < g.target;
        Create an edge between u and A[i][j];
        i = i + 1;
        if i == X then
            i = 0;
            j = (j + 1)%k;
d = ⌊Change_total / (X × k)⌋ + 1;
while A contains node with degree d − 1 do
    Random select two rows l_1 and l_2 in A with maximum number of d − 1
    degree nodes;
    Random select u_1 ∈ l_1, u_2 ∈ l_2 that
    u_1.degree = u_2.degree = d − 1;
    Create an edge between u_1 and u_2;
```

By analyzing Algorithm 2, we find it has the following property:

THEOREM 4. *Algorithm 2 constructs a k-degree anonymous graph without violating* SGC *and all the noise nodes have the degree* $degree_{target}$*, if*

$$degree_{target} = \left\lfloor \frac{Change_{total}}{X \times k} \right\rfloor + 1 \tag{1}$$

$$X \geq max(\left\lceil \frac{Change_{max}}{k} \right\rceil, 2) \tag{2}$$

$$(degree_{target} \times k \times X + Change_{total})\%2 \equiv 0 \tag{3}$$

The proof of Theorem 4 can be found in the Appendix D.

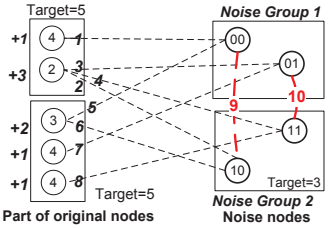


Figure 6: A running example of Algorithm 2

After the second step of the node/edge adding, $Change_{total}$ and $Change_{max}$ are both constants. As we introduced at the beginning of the third step, $degree_{target}$ is selected as the degree that the maximum number of groups of size k in G_{L_1} have, thus, $degree_{target}$ is also a constant. By solving the formulas in Theorem 4, we can derive X for Algorithm 2. If an integer X does not exist for Equation 1 or the Equation 2/3 cannot be satisfied, we can get a proper X by increasing the value of $Change_{total}$. There are several methods to increase the value of $Change_{total}$. Increasing the target degree of a group that contains the odd number of nodes by 1 can increase $Change_{total}$ by an odd number[7]. Increasing the target degree of a group that contains the even number of nodes by 1 or a group that contains the odd number of nodes by 2 can increase $Change_{total}$ by an even number. Delete one noise edge added in the previous step can also increase $Change_{total}$ by 2. Figure 5 (c) shows an example to get a suitable X by increasing $Change_{total}$. Suppose $degree_{target} = 3$, at the beginning, $Change_{total}$ is 1 which is an odd number. So we let group 2's target degree increase by 1 since group 2 contains odd number of nodes. By doing this, $Change_{total}$ is increased to 4. However, we still cannot get an integer X. Then we can increase the $Change_{total}$ to 8 by deleting the added edge in the second step and increasing group 1's target degree by 1. Finally, $X = 2$ is found and Algorithm 2 can be invoked. A degree anonymous graph is constructed under *SGC* where all the noise nodes have degree 3.

After completing the three steps, for each node u in G_{L_1} that needs to increase its degree by x, we connect u with x nodes which belong to either G_{L_1} or newly added noise nodes. All the noise nodes connected with u are in different groups. Since the degree of the noise node groups is the same as the degree of some existing groups, the noise nodes are hidden into those original nodes. In order to show the hidden effects, we did three tests in the Appendix G. The results showed that the number of the noise node groups is much fewer than the number of existing groups with the same degree and the neighborhoods of the noise nodes are not special comparing with those of the original nodes in G_{L_1}.

4.2 Label Assignment

After adding the noise nodes/edges, we should assign labels to them. Suppose $No_{con}(A_1, A_2, l))$ is the number of edges in the original graph that connect label A_1 with A_2 through an edge with label l_x, for all A_1, A_2 and l, a node/edge label assignment with less $No_{con}(A_1, A_2, l)$ changes is preferred. So we use the following heuristic method to do the assignment.

We first assign labels to the noise edges. For the edges that connect with at least one noise node, we randomly assign labels to them following the edge label distribution in the original graph. For an edge between two nodes u_1 and u_2 in G_{L_1}, denote u_1's label list as $(A_{u_1,1}, A_{u_1,2}, ..., A_{u_1,t})$ and u_2's label list as $(A_{u_2,1}, A_{u_2,2}, ..., A_{u_2,t})$, we select the edge label l_x that satisfies:

$$max(\sum_{j=1}^{t} No_{con}(A_{u_1,j}, A_{u_2,j}, l_x)), \tag{4}$$

where $No_{con}(A_{u_1,j}, A_{u_2,j}, l_x)$ is the number of edges in the original graph that connect label $A_{u_1,j}$ with $A_{u_2,j}$ through an edge with label l_x. Since the new created connections between labels appear the maximum number of times in the original graph, selecting l_x tends to change the connection between labels as less as possible.

After each noise edge obtains a label, we decide the labels of noise nodes according to the assigned edge labels and the original graph's connectivities. For a noise node u, suppose the labels on the edges adjacent to u are: $\{l_1, l_2, ..., l_d\}$, the selection rule of u's label list $(A_{u,1}, A_{u,2}, ..., A_{u,t})$ is:

$$max(\sum_{i=1}^{d}\sum_{j=1}^{t} No_{con}(A_{u,j}, A_{u_{e_i},j}, l_i)), \tag{5}$$

where u_{e_i} is the node connecting with u through edge i, $A_{u_{e_i},j}$ is u_{e_i}'s label on dimension j, and $No_{con}(A_{u,j}, A_{u_{e_i},j}, l_i)$ is the number of edges in the original graph that connect label $A_{u,j}$ with $A_{u_{e_i},j}$ through an edge with label l_i. We use formula 5 to select the node labels which make the new created connections between labels appear the maximum number of times in the original graph. When all the noise nodes have labels, the graph can be published by generalizing the node labels of each noise node group, which is denoted as G_{L_2}.

[7] We can force the grouping algorithm to generate at least one group with the odd number of nodes that contains nodes in set_{l_2}.

5. LEVEL 3 PROTECTION

At Level 3, an attacker also knows the labels on the edges adjacent to set_{l_3}. Then, in this case, the anonymous graph generated at Level 2 cannot prevent the node from re-identification if the attacker also uses the edge label information. For example, in Figure 1(c), c and f are assigned to the same group and have the same degree. If an attacker knows that Tom is a 35 year-old man from Africa who has two neighbors both with type 1 (roommate), he can use this information to re-identify that c is Tom. In order to prevent such an attack, at level 3, when an attacker uses node label list, degree and edge label information of any node in set_{l_3} to search the published graph, the real matching node should be mixed with at least $k-1$ other nodes. We call a graph that satisfies the above property ***k-degree-label anonymous graph***. We call the vector of labels on the edges adjacent to a node u as u's ***degree label sequence***. For example, Figure 1(d) is a 2-degree-label anonymous graph. The degree label sequence of node d is $[1,(0,1),(0,1)]$.

We can change a k-degree anonymous graph to be a k-degree-label anonymous graph by generalizing the labels on edges. That is, if a group contains at least one node that needs Level 3 protection, for any two nodes in it, change their degree label sequence to be the same by generalizing edge labels. Figure 2(b) shows the results of applying edge label generalization when Tom requires Level 3 and Leo requires Level 2 protection. In Figure 2(a), c's degree label sequence is $[0,1]$ and f's degree label sequence is $[0,0]$. By generalizing edge $e(c,d)$ and $e(f,g)$'s labels to be $(0,1)$, both c and f's degree label sequences become $[0,(0,1)]$.

Algorithm 3 shows how to generate a k-degree-label anonymous graph from G_{L_2}. In order to generalize the edge labels with the minimum level of edge label generalization, in line 6-8, we sort the labels in the degree label sequence using the same sorting rule[8]. In line 9-14, we generalize the labels appearing at the same position of their corresponding degree label sequences in a group to be the same. We repeatedly adjust each group that needs Level 3 protection until G_{L_2} becomes a k-degree-label anonymous graph.

6. EXPERIMENTS

6.1 Utilities

The analysis in our algorithm description part proves the privacy preserving effect. Another important question is: how to measure the quality of the published graph. For the labeled graphs, people use the change of certain queries' results to measure the quality[17][2]. In this paper, we test two kinds of queries:

- One hop query (1 hop)
  This query is: "The number of node pairs (u_1,u_2), where u_1 has label l_1, u_2 has label l_2, and u_1/u_2 are directly connected with edge label l_e". A one hop query can be represented by (l_1,l_e,l_2). If each node has more than one label, the query is represented by (d,l_1,l_e,l_2), where l_1 and l_2 are the labels on dimension d. For example, in Figure 1(a), each node has two kinds of labels (area, age), query *(1, Asia, 1, Africa)* means the number of directly connected pairs with edge type "1" and the two endpoints of this pair have location (dimension 1) labels "Asia" and "Africa" respectively.
- Two hop query (2 hop)
  This query is "The number of node tuple (u_1,u_2,u_3), where u_1 has label l_1, u_2 has label l_2, u_3 has label l_3. u_1 and u_2 are directly connected with edge label l_{e_1}, and u_2 and u_3 are directly connected with edge label l_{e_2}". A two hop query can also be represented as $(l_1,l_{e_1},l_2,l_{e_2},l_3)$, and extended to form $(d,l_1,l_{e_1},l_2,l_{e_2},l_3)$ with a dimension parameter if each node has more than one label.

[2] used the above two types of queries to check information change of direct and indirect connections, respectively. We measure the utility using the average relative query error of all possible basic form queries. For a query q, its relative error is $\frac{|n-n'|}{n}$. Where n and n' are the results of executing query q on the original and the privacy-preserved published graph, respectively.

Algorithm 3: Generate degree label anonymous graph

```
1  while true do
2      if G_{L_2} is a degree label anonymous graph then
3          break;
4      for each group g in C do
5          if g contains a node in set_{l_3} then
6              for each node u in g do
7                  Sort the degree label sequence of u;
8              Vector general = the degree label sequence of u_0 (u_0 ∈ g);
9              for each node u in g, u ≠ u_0 do
10                 for each edge label L in u's degree label sequence do
11                     Find the mapping label L' of L in general;
12                     L' = L ∪ L';
13             for each node u in g do
14                 Set u's degree label sequence = general;
```

Degree Distribution (*DD*)[8, 19], as one of the basic graph characteristics, is considered by nearly all social network analysis works. We also tested the *DD*'s change between the original graph and the published graph to show the structure difference. Li [7] showed the *EMD* (Earth Mover Distance) is a good measure of the difference between two distributions and used *EMD* to define the *t-closeness* model. So in this paper, we use the *EMD* between the original graph's *DD* and published graph's *DD* to represent the *DD*'s change. The larger the *EMD* is, the larger *DD* changes. We give the detailed *EMD* between two distributions in Appendix E.

6.2 Querying Anonymized Data

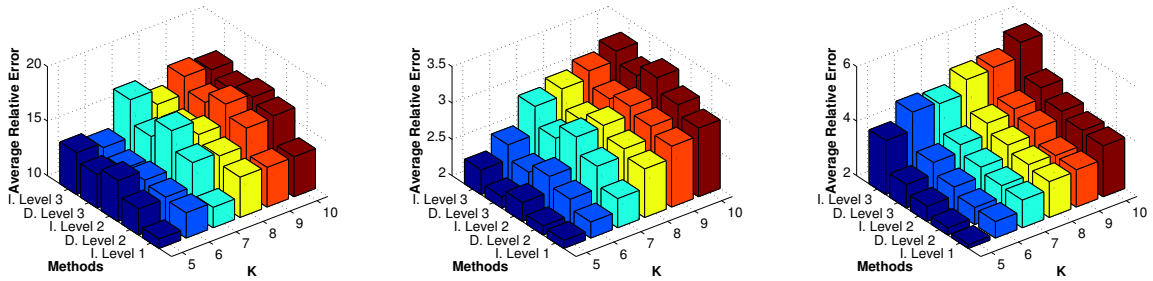
The published graph's degree distribution can be directly computed. In order to perform queries on the published graph with generalized labels, we can use Sampling Consistent Graphs method [2] to randomly sample a graph that is consistent with the published data, and analyze the sampled graph. When conducting sampling, we choose an assignment of labels to nodes for each group, which is consistent with the published graph. For each edge, we assign a label that is consistent with the generalized one. The query can be operated on the sampled graph. Normally, an "expected" answer can be evaluated by computing the mean of query results on several consistent sampling graphs.

6.3 Results

We test our algorithms on two real data sets and one synthetic data set: Speed Dating data (*SD*, 552 nodes), ArXiv data (*AX*, 19835 nodes) and ArXiv Data with uniform labels (*AXU*). Appendix F presents the details about these three data sets. In this part, "I. Level x" is used to show the utility of the published graph in which all the nodes have Level x protection (naive protection). "D. Level x" stands for the result of the published graph in which different nodes have different protection requirements.

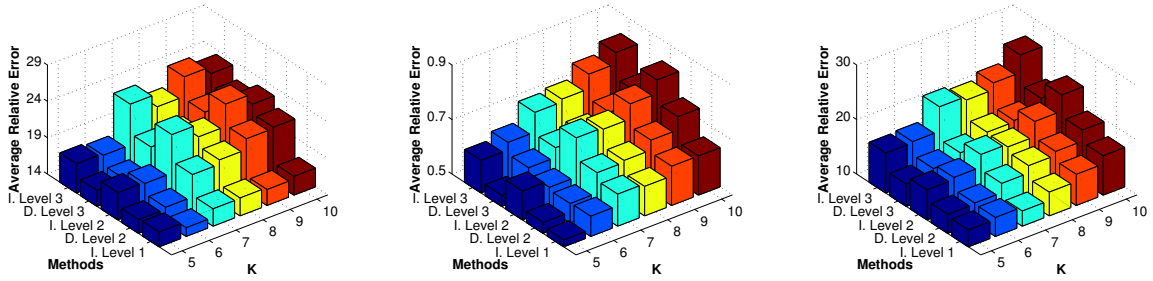
We set all the nodes need Level 1 protection, 5% nodes need Level 2 protection and 5% nodes need Level 3 protection. The nodes that need Level 2 and Level 3 protections are randomly selected. We call the random setting of Level 2 and Level 3 protection nodes as random privacy selection. Since all the nodes need Level 1 protection, "I. Level 1" and "D. Level 1" have the same results, we use "I. Level 1" to represent both of them.

[8] The label in a lower level is less than the label in a higher level ($0 < (0,1)$). For the labels in the same level, define an order between them.



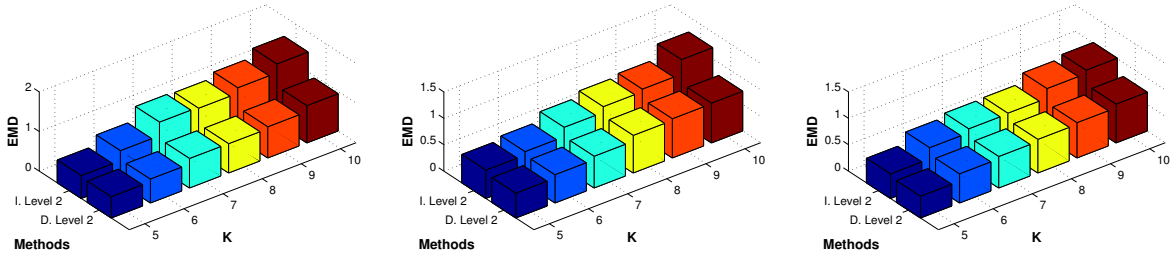
(a) Speed Dating (b) ArXiv (c) ArXiv with uniform labels

Figure 7: Average relative error on all possible one hop queries



(a) Speed Dating (b) ArXiv (c) ArXiv with uniform labels

Figure 8: Average relative error on all possible two hop queries



(a) Speed Dating (b) ArXiv (c) ArXiv with uniform labels

Figure 9: Average change of the degree distribution

In order to avoid the influence of the random privacy requirement selection of nodes, we randomly divide the nodes in each graph to 10 non-overlapping folds. Each time, the nodes that need Level 2 and Level 3 protection are randomly selected in one fold. Thus totally 10 random selections that do not have any overlapping are tested for each graph. This kind of selection has the same spirit as the cross-violation sampling selection in the AI field [11].

Figures 7 and 8 shows the average relative errors of one hop queries and two hop queries respectively. Each point in the figures is the mean result on the corresponding published graphs of the 10 different random privacy requirement selections. For each published graph of one random privacy selection, we generate 20 consistent sampling graphs to get the "expected" result. From the results, we can observe that the graphs allowing personalized protections have higher utility than the ones that make all nodes achieve the highest protection level. We also find that for each protection situation, in most cases the mean of the average relative error increases with the increasing of k. This satisfies the intuition, since the published graph with large k has more difference to the original graph than the one with small k. However, at several points, such as the point $k = 7$ of in Figure 8(a), the utility of some methods decreases a little suddenly. One reason is the random characteristics of the algorithms, which might make the consistent graph sampling sometimes generate a better result for a large k. Another reason is the effect of the noise nodes/edges. If the generalization at Level 1 is not good enough, some connections disappear during the modification. The noise nodes/edges create some new connections in the graph, which could possibly add some disappeared connections back. A larger k adds more noise nodes/edges, thus more missing connections are added back.

Since only Level 2 protection changes the degree, we show degree distribution's change of "I. Level 2" and "D. Level 2". Figure 9 shows the *EMD* between the original graph's *DD* and the published graph's *DD*. In Figure 9, each point is the mean result on the corresponding published graphs of 10 different random privacy requirement selections. From the result we can see, "D. Level 2" performs much better than "I. Level 2". This is because allowing personalized protection changes the original graph's structure much less than the method that protects all nodes with the highest level.

Next we do the significant testing for random privacy selections to show the benefit is not because of the bias of the privacy requirement selections. We define the research hypothesis as "The personalized protection always gets better utility than naive protection". The corresponding null hypothesis is "The personalized protection

does not always get better utility than naive protection". We do paired t-test (p=0.05) for each k based on the 10 random privacy selections. The testing is conducted for both Level 2 and Level 3 respectively. For Level 2, we compare "I. Level 2" and "D. Level 2". For Level 3, we compare "I. Level 3" and "D. Level 3". The results are shown in Table 1. In the table, 5/6 means the null hypothesis has been successfully disproved for 5 times in the 6 testing for k from 5 to 10. In Table 1, totally 84 testing are operated and only 8 of them failed. From the results we can see, in most cases, the research hypothesis can be proven to be true, especially for two hop queries and degree distribution.

Table 1: The result of significant testing

	SD		AX		AXU	
	Level 2	Level 3	Level 2	Level 2	Level 2	Level 3
1 Hop	4/6	4/6	5/6	5/6	5/6	6/6
2 Hop	6/6	6/6	5/6	6/6	6/6	6/6
DD	6/6	/	6/6	/	6/6	/

We also test the time efficiency of our algorithms, the details can be found in the Appendix G. The results show our algorithms have good time performance. For a graph with around 20000 nodes, the algorithms can finish computation within 10 minutes.

7. RELATED WORK

When publishing an anonymized social network, the unique patterns such as node degree or subgraph can be used to re-identify the nodes/links [6]. The attack that only uses certain background knowledge and doesn't "actively" change the graph is called passive attack, and the one "actively" changes the graph when social networks are collecting data is called active attack. Most current works can be categorized into two classes: to prevent passive attack [8, 6, 4, 16, 3, 15, 17] and to prevent active attack [13].

Most works focused on preventing passive attack. Hay[6] proposed a clustering algorithm to prevent privacy leakage using vertex refinement, subgraph, and hub-print queries. Zheleva[16] developed algorithms to prevent the sensitive link leakage by mining observed links in published network. Campan[3] discussed how to implement k-anonymous when consider both node labels and structure information. Cormode[4][2] introduced (k, l)-groupings for bipartite graph and social networks to do the protection respectively. Liu[8] defined and implemented k-degree anonymity model on network structure, that is for a published network, for any node, there exists at least *k-1* other nodes have the same degree as this node. Zhou[17] considered stricter model: for every node there exist at least *k-1* other nodes share isomorphic neighborhoods when taking node labels into account. Zou[19] considered the strongest structure 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 this node. Ying[15] studied how random deleting and swapping edges changes graph properties and proposed an eigenvalues oriented random graph change algorithm.

One method to prevent an active attack is to recognize the fake nodes added by attackers and remove them before publishing the data. Shrivastava[13] proposed an algorithm that can identify fake nodes based on the neighborhood graph difference between normal nodes and fake nodes.

There are also some other related works. Backstrom[1] described active attacks based on randomness analysis and demonstrated that an attacker may plant some constructed substructures associated with the target entities. Liu[9] treated weights on the edges as sensitive values and proposed a method to preserve shortest paths between most pairs of nodes in the graph.

The works to prevent the passive attack are most relevant to our work. Compared with those works, ours is the first that tries to allow users to provide the personal privacy setting in social networks. This provides a new opportunity to achieve higher utilities when publishing a graph. We also considered the link protection besides preventing the node re-identification.

8. CONCLUSION

In this paper, we design a comprehensive privacy protection framework for the labeled social networks. This framework allows different users to set different privacy protection requirements. With the framework, we increase the utilities of the published graph and at the same time satisfy all user's privacy requests. The framework combines the structure protection techniques (such as graph editing) and micro data protection techniques (such as generalization). Our method provides the possibility of offering different services to different users. We prove the effectiveness of the proposed framework through extensive experiments.

Acknowledgement

This work is supported in part by Hong Kong RGC grants N_HKUST612/09, 619308 and US NSF through grants IIS-0914934, DBI-0960443, OISE-0968341 and OIA-0963278.

9. REFERENCES

[1] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW'07*, pages 181–190, 2007.

[2] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. Class-based graph anonymization for social network data. *Proc. VLDB Endow.*, 2(1): 766–777, 2009.

[3] A. Campan and T. M. Truta. A clustering approach for data and structural anonymity in social networks. In *PinKDD'08*, 2008.

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

[5] W. Eberle and L. Holder. Discovering structural anomalies in graph-based data. In *ICDMW'07*, pages 393–398, 2007.

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

[7] N. Li and T. Li. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE'07*, pages 106–115, 2007.

[8] K. Liu and E. Terzi. Towards identity anonymization on graphs. In *SIGMOD'08*, pages 93–106, 2008.

[9] 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.

[10] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *PODS'04*, pages 223–228, 2004.

[11] J. Neville, B. Gallagher, and T. Eliassi-Rad. Evaluating statistical tests for within-network classifiers of relational data. In *ICDM'09*, pages 397–406, 2009.

[12] C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *KDD'03*, pages 631–636, 2003.

[13] N. Shrivastava, A. Majumder, and R. Rastogi. Mining (social) network graphs to detect random link attacks. In *ICDE'08*, pages 486–495, 2008.

[14] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *VLDB'06*, pages 139–150, 2006.

[15] X. Ying and X. Wu. Randomizing social networks: a spectrum preserving approach. In *SDM'08*, pages 739–750, 2008.

[16] E. Zheleva and L. Getoor. Preserving the privacy of sensitive relationships in graph data. In *PinKDD'07*, pages 153–171, 2007.

[17] B. Zhou and J. Pei. Preserving privacy in social networks against neighborhood attacks. In *ICDE'08*, pages 506–515, 2008.

[18] B. Zhou, J. Pei, and W. Luk. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *SIGKDD Explor. Newsl.*, 2(10): 12–22, 2008.

[19] L. Zou, L. Chen, and M. T. Özsu. k-automorphism: a general framework for privacy preserving network publication. *Proc. VLDB Endow.*, 2(1): 946–957, 2009.

APPENDIX

A. GRAPH MODEL

The graph model used in this paper is defined as follows:

DEFINITION 1. *Published Graph: a published graph for a social network is defined as a six-tuple $G(V, E, \sigma, \lambda, \beta, \lambda')$, where V is a set of nodes and each node represents a node in a social network. $E \subseteq V \times V$ is a set of edges between nodes. σ is an array, each of its dimensions represents a set of labels that nodes have. λ is an array that $\forall i$, $\lambda[i] : V \longrightarrow \sigma[i]$ maps nodes to their labels. β is a set of labels that edges have. $\lambda' : E \longrightarrow \beta$ maps edges to their labels.*

In this graph model, both nodes and edges have labels.

B. NP HARD PROOF

PROOF. The k-anonymous problem of the micro data [10] is NP-hard. We build an empty graph G and let each item in the micro data as a node. The quasi-identifiers of each item are set as the labels of the item's corresponding node. We do not build any connections between nodes. Thus all the nodes in G have the same degree 0. Then the second part of the cost function is 0. The personalized protection problem is exactly the k-anonymous problem of the micro data. So this problem is a NP-Hard problem. □

C. SAFE GROUPING CONDITION PROOF

PROOF. We first prove under the condition in Theorem 3, for any edge e in the published graph, the probability that the attacker knows a node u_x is involved in this edge is less than $\frac{1}{k}$. If the condition in Theorem 3 is satisfied, for any edge $e(u_x, u_y)$, there are more than k candidates for u_x and also more than k candidates for u_y. Since there is no overlapping between the candidate sets of u_x and u_y (the first condition), the probability that an attacker can successfully find whether a node is an endpoint of an edge is less than or equal to $\frac{1}{k}$.

Next we prove that the probability that the attacker knows a node u_x and a node u_y having a connection is less than or equal to $\frac{1}{k}$. Suppose node u_x is in group g_x and node u_y is in group g_y. Let a_x and a_y denote the number of all possible label list assignments for g_x and g_y respectively. a'_x and a'_y are the possible label list assignment numbers when one node has been assigned with a certain label list. Since g_x and g_y are two non-overlapping groups, the probability that the attacker finds u_x and u_y having connection $P(e(u_x, u_y))$ is:

$$P(e(n_x, n_y)) = \frac{n_e \times a'_x \times a'_y}{a_x \times a_y},$$

where n_e is the number of edges whose two endpoints might be u_x and u_y (u_x's label list appears in the label list set of one endpoint and u_y's appears in that of the other one).

With generalization, a node may have the label list of any node in the same group. So $n_e = x$, $a_x = |g_x|!$, $a_y = |g_y|!$, $a'_x = (|g_x| - 1)!$ and $a'_y = (|g_y| - 1)!$,

$$P(e(n_x, n_y)) = \frac{n_e \times (|g_x| - 1)! \times (|g_y| - 1)!}{|g_x|! \times |g_y|!} = \frac{n_e}{|g_x||g_y|}$$

According to Theorem 3, $n_e = x \leq \frac{|g_x||g_y|}{k}$, then

$$P(e(n_x, n_y)) \leq \frac{|g_x||g_y|}{k \times |g_x||g_y|} = \frac{1}{k}$$

So, conditions in Theorem 3 can guarantee the privacy objectives (2) and (3). □

D. ALGORITHM PROPERTY PROOF

PROOF. It is obvious all the nodes in G_{L_1} can reach their target degrees using Algorithm 2. Next, we show *SGC* is guaranteed when building connections between the original nodes and the noise nodes. When adding edges between nodes in G_{L_1} and A, the algorithm selects a group g in G_{L_1}. Then the algorithm repeatedly adds edges between the nodes in g and A until g becomes a group with all the nodes in it having the same degree. The nodes in A are selected column by column to add edges. Each row of nodes in A is a group that only contains noise nodes. Since $X \geq \frac{Change_{max}}{k}$, we get $\frac{Change_{max}}{X} \leq k$. The number of edges x between any group g_x in G_{L_1} and any group g_y in A is at most k after this step. Since $|g_x| \geq k$ and $|g_y| = k$, we have $\frac{|g_x||g_y|}{k} \geq k \geq x$. This operation also does not create any edge within a group, the conditions in Theorem3 are satisfied.

We then show the algorithm let all the noise nodes have the degree $degree_{target}$ without violating the safe group condition. Since Algorithm 2 adds edges to noise nodes' array A column by column. $degree_{target} = \left\lfloor \frac{Change_{total}}{X \times k} \right\rfloor + 1$, all the noise nodes can only have the degree $degree_{target}$ or $(degree_{target} - 1)$ after the first step. And all the nodes with the degree $(degree_{target} - 1)$ appear at the bottom part of the array. That is if $A[i][j]$ has degree $(degree_{target} - 1)$, $A[i + 1][j]$, $A[i][j + 1]$ and $A[i + 1][j + 1]$ all have the degree $(degree_{target} - 1)$.

- At the beginning, the difference between the numbers of $(degree_{target} - 1)$ degree nodes in any two rows is at most 1;
- Each time, either the selected two rows have the same number of $(degree_{target} - 1)$ degree nodes or one row has one more $(degree_{target} - 1)$ degree node than the other.
- The numbers of $(degree_{target} - 1)$ degree nodes in the two rows of the last step can only be (1, 1) or (2, 1);
- Since $degree_{target} \times X \times k$ and $Change_{total}$ are both even numbers, the number of $(degree_{target} - 1)$ degree nodes is $(degree_{target} \times X \times k$ - $Change_{total})$ at the beginning. Each step we remove 2 nodes with the degree $(degree_{target} - 1)$. Since both $(degree_{target} \times X \times k - Change_{total})$[9] and 2 are even numbers, only (1, 1) case can appear in the last step.

From the above analysis, we can see Algorithm 2 could finally adjust all the nodes' degrees in A to be $degree_{target}$. This operation would not add any edge within a group in A. The number of edges created between any two groups in A is at most k, where $k \leq \frac{k^2}{k}$. So the algorithm satisfies *SGC*. □

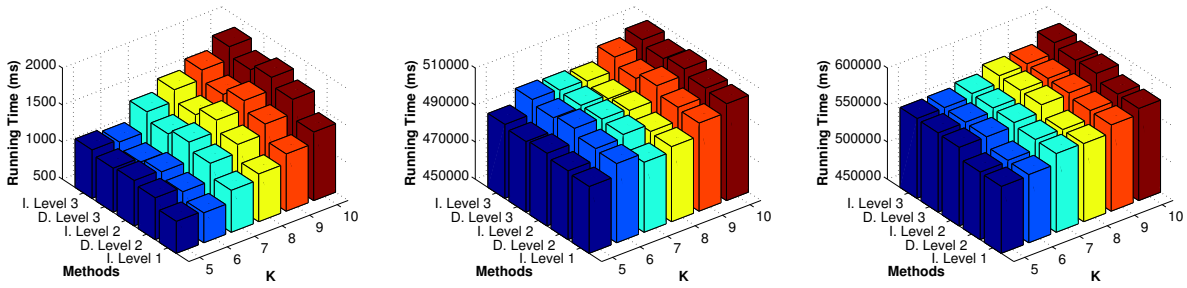
E. DISTANCE BETWEEN DEGREE DISTRIBUTIONS

As shown in paper [7], for two distributions on continuous numerical values $P[(v_1, p_1), (v_2, p_2), ...(v_m, p_m)]$ and $Q[(v_1, q_1), (v_2, q_2), ...(v_m, q_m)]$ (v_i is the numerical value, p_i and q_i are distribution values), let $r_i = p_i - q_i$, $(i = 1, 2, ..., m)$, the earth mover distance between P and Q can be calculated as:

$$\begin{aligned} EMD[P, Q] &= \frac{1}{m-1}(|r_1| + |r_1 + r_2| + ... + |r_1 + r_2 + ... + r_m|) \\ &= \frac{1}{m-1} \sum_{i=1}^{m} |\sum_{j=1}^{i} r_j| \end{aligned}$$

In our case, suppose min_d and max_d are the minimum and maximum degree of the original graph. According to our algorithms, all node degrees in the published graph also falls into range

[9] Since $(degree_{target} \times X \times k + Change_{total})$ is even, $(degree_{target} \times X \times k - Change_{total})$ is also even.



(a) Speed Dating (b) ArXiv (c) ArXiv with uniform labels

Figure 10: Running Time

$[min_d, max_d]$. So we can generate the degree distributions of the original graph and the published graph on $[min_d, max_d]$ and compute their distance directly. For example, the degree distributions of Figure 1(a) and Figure 1(d) are $[(3, \frac{1}{7}), (2, \frac{3}{7}), (1, \frac{3}{7})]$ and $[(3, \frac{2}{11}), (2, \frac{9}{11}), (1, \frac{0}{11})]$ respectively.

F. DATASETS

- ***Speed Dating Data***: We extract a graph from Speed Dating (flowingdata.com) with 552 nodes and 4194 edges. The average degree of this graph is 15.2. Each node has 19 labels, and these labels represent 4 types of information: age, race, field and hobbies, where hobbies contain 16 items. Each edge denotes a "date" between two people. We use the same method as [2] to set the edge labels. If both people were positive about their counterpart, the edge label is set to "match"; otherwise it is set to "unmatch". The numbers of edges with type "match"/"unmatch" are 3504 and 690 respectively.
- ***ArXiv Data***: ArXiv(arXiv.org) is an e-print service system in Physics, Mathematics, Computer Science, Quantitative Biology, Quantitative Finance and Statistics. We extract a subset of co-author graph in Computer Science, which contains 19835 nodes and 40221 edges. Each node denotes an author, and each edge means two authors have at least 1 co-author paper. The average degree of this graph is 4.06. Each edge has an integer weight that represents the number of papers co-authored by the two endpoints. We set label "seldom" to edges with weight 1, "normal" to edges with weight 2 - 5 and "frequent" to edges with weight larger than 5. The numbers of edges with these three labels are 33856, 6058 and 312 respectively. We set each author's research field as his/her node label. The research field is set as the category of papers he/she published most. There are totally 37 different values of node labels.
- ***ArXiv Data with uniform labels***: We also generate a synthetic graph based on the ArXiv graph by uniformly set node labels and edge labels. Each node has one label selected from 20 different values. Each edge has one label selected from 3 different values. All the values are uniformly distributed.

G. MORE EXPERIMENTS

We test the running time of our algorithms. The results are shown in Figure 10. From the results, we can see the personalized privacy protection do not introduce extra computation time. The algorithms introduced in this paper have good time efficiency. For a graph with around 20000 nodes, the algorithms can finish computation within 10 minutes.

In order to show the noise nodes are well mixed with the normal nodes in the published graph, we conduct three tests. For the first one, we suppose an attacker already knows the degree and the group size of the noise nodes' groups. We test the ratio of real noise nodes in the selected nodes if the attacker uses this information to filter the published graph. Table 2 shows the results. In the table, "I." stands for the protection using the naive method (protecting all nodes to the highest level). "D." stands for the personalized protection. From the table, we can see the real noise node is the minority in the selected nodes. Even an attacker already knows the degree of the noise nodes, he still cannot filter them out. The results of AX and AXU are very small since the graphs are sparse (average degree = 4) which makes most of the degree change can be handled by connecting the original nodes. In the second test, we use

Table 2: Correct filtering ratio using the degree information

	SD (%)		AX (%)		AXU (%)	
K	I.	D.	I.	D.	I.	D.
5	8.70	8.54	0.00	0.16	0.00	0.15
6	9.52	8.45	0.24	0.22	0.24	0.23
7	36.25	29.93	0.00	0.35	0.00	0.49
8	0.00	25.56	0.32	0.27	0.32	0.29
9	18.75	19.82	0.36	0.31	0.00	0.56
10	0.00	19.98	0.40	0.44	0.40	0.41

Clustering Coefficient (*CC*) of a vertex, which is commonly used [8][19] to represent the vertex's neighborhood graph. *CC* 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 assume that an attacker knows the *CC* value's range of each noise node. Suppose the *CC* value of a noise node u is c, an attacker uses the range $[c(1-10\%), c(1+10\%)]$ to filter u. We show the ratios of noise nodes in the filtered results in Table 3. From the results, we can see the *CC* values of the noise nodes are not special comparing with the normal nodes.

Table 3: Correct filtering ratio using the CC information

	SD (%)		AX (%)		AXU (%)	
K	I.	D.	I.	D.	I.	D.
5	5.05	4.68	0.00	0.00	0.00	0.11
6	4.78	4.55	0.85	0.85	0.84	0.38
7	11.07	9.09	0.00	0.00	0.00	0.32
8	0.00	8.49	1.12	1.12	1.13	0.24
9	9.82	8.76	1.27	1.27	0.00	0.51
10	0.00	12.06	1.40	1.40	1.40	0.38

There are also some works[12][5] to detect abnormal structures in graph databases. Thus, in the third test, for each degree that contains noise nodes, we extract all the neighborhood graphs of the nodes with this degree in the published graph. We use the tool in [5] to detect abnormal structures in the graph database formed by these neighborhood graphs. 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 results show that the neighborhood graphs of the noise nodes do not have any special characters to be filtered out.