Preserving Privacy and Frequent Sharing Patterns for Social Network Data Publishing

Benjamin C. M. Fung
CIISE, Concordia University
Montreal, QC, Canada
fung@ciise.concordia.ca

Yan'an Jin
Huazhong University of Science and Technology
Hubei University of Economics, P. R. China
yan.an.jin@hbue.edu.cn

Jiaming Li
CIISE, Concordia University
Montreal, QC, Canada
l_jiamin@ciise.concordia.ca

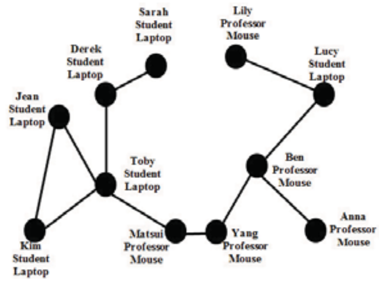
***Abstract*—Social network data provide valuable information for companies to better understand the characteristics of their potential customers with respect to their communities. Yet, sharing social network data in its raw form raises serious privacy concerns because a successful privacy attack not only compromises the sensitive information of the target victim but also the relationship with his/her friends or even their private information. In recent years, several anonymization techniques have been proposed to solve these issues. Most of them focus on how to achieve a given privacy model but fail to preserve the data mining knowledge required for data recipients. In this paper, we propose a method to k-anonymize a social network dataset with the goal of preserving frequent sharing patterns, one of the most important kinds of knowledge required for marketing and consumer behaviour analysis. Experimental results on real-life data illustrate the trade-off between privacy and utility loss with respect to the preservation of frequent sharing patterns.**

I. Introduction

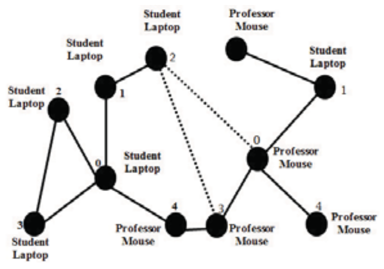
In recent years, the emergence of social network applications, such as Facebook, Twitter, and MySpace, has provided a new source of information for consumer behaviour analysis. By identifying the common preferences with respect to the customers' background information and their connections, a company can better customize their products and marketing strategy for different communities. Thus, there is an urge to share social network data together with the set-valued data of the participants. The set-valued data, for example, can be online purchase transactions or click history on advertisements on social network websites. However, releasing social network data in its raw form raises serious privacy concerns to the participants. In this paper, we present a method to anonymize the social network with the goals of hiding the identities of the participants and preserving the frequent sharing patterns within a community.

A. Motivating Scenario

Figure 1(a) depicts a typical social network of 11 participants together with their names, jobs, and purchased items via the advertisements in the social network. The social network service provider wants to share such useful data to its cooperative partners who placed advertisements for market analysis. Yet, sharing such information would compromise the privacy of participants, which in turn damages the image of the social network service provider. A naive method is to de-identify the social network data by simply removing the explicit identifiers, such as name and birthdate. However, many previous works



(a) Raw social network



(b) Anonymized social network

Fig. 1. Sample social network

in privacy-preserving data publishing [1] have already shown that simply removing explicit identifiers is insufficient because an adversary may utilize some external knowledge to identify an individual from the data. The following example illustrates a privacy attack on a de-identified social network.

Example I.1. Consider the social network in Figure 1(a). Even if the names of the participants have been removed before releasing the data, an adversary may still identify an individual using *neighborhood attack* [2]. Suppose the adversary knows the target victim $Toby$ has four friends and two of his friends know each other. Given such background knowledge, the adversary can easily identify $Toby$'s vertex from the social network. One effective way to thwart this kind of neighborhood attack is to ensure that the 1-neighborhood network structure of $Toby$ is isomorphically similar to the 1-neighborhood network structure of at least $k-1$ other vertices in the shared social network data. This privacy model is known as k-*anonymity* on social network data [2][3]. To make $Toby$ 2-anonymous, two edges, indicated by the dashed lines in Figure 1(b), are added between Ben and $Sarah$, and $Yang$ and $Sarah$. ∎

Achieving k-anonymity on social network is not a new problem. It has been previously studied in [2]. The challenge addressed in this paper is how to preserve the frequent patterns shared within a community [4], known as *frequent sharing patterns*, so that the data recipient can still retrieve them even from the k-anonymous social network data. Specifically, a frequent sharing pattern is a combination of vertex labels that is shared within a connected subgraph with a minimum number of vertices specified by the social network data holder. The following example illustrates the general idea of minimizing the impact on frequent sharing patterns. A formal definition is given in Section II.

Example I.2. Consider Figure 1(a) again. The pattern $\{Laptop\}$ has support 5 because a maximum of five connected vertices contain $Laptop$. Similarly, the pattern $\{Mouse\}$ has support 4. Let the data-holder-specified minimum support be 5. Then $\{Student\}$ and $\{Laptop\}$ are frequent but $\{Professor\}$ and $\{Mouse\}$ are not. To make $Toby$'s neighborhood structure 2-anonymous, we have the option to add two edges as described in Example I.1 or add an edge between $Lily$ and Ben. The latter option is less desirable because adding an edge between $Lily$ and Ben would increase the support of $\{Mouse\}$ and $\{Professor, Mouse\}$ from 4 to 5, resulting in some false frequent sharing patterns. ■

B. Challenges and Contributions

The challenges of anonymizing social network data for frequent sharing patterns mining are summarized as follows. First, social network data are a composition of graph data and set-valued data, representing the relationships among the participants and the (sensitive) personal information of the participants, respectively. Thus, existing anonymization methods for k-anonymity [5], ℓ-diversity [6], and confidence bounding [7] that are designed for tabular data are not applicable to social network data. Second, in order to preserve the frequent sharing patterns, a straight-forward approach is to first extract all frequent sharing patterns and then minimize the impact on the extracted patterns in the anonymization process. However, the preprocessing step of extracting the frequent sharing patterns from social network is expensive. Third, real-life social network data are usually very large; therefore, it is essential to develop a scalable anonymization algorithm.

The contributions of this paper are summarized as follows. First, to the best of our knowledge, this is the first anonymization algorithm to achieve k-anonymity on social network data while minimizing the impact on frequent sharing patterns in the set-valued data. Second, our proposed method is not only effective but also scalable to anonymize large volume of social network data. Third, we verify the effectiveness of our proposed method by extensive experiments on real-life data. The results suggest that our algorithm can effectively preserve the privacy with reasonable trade-off between privacy and information utility measured in terms of preserving frequent sharing patterns.

The rest of the paper is organized as follows. In Section II, we formally define the problem. Our proposed anonymization method for preserving frequent sharing patterns is presented in Section III. Our experimental results are shown in Section IV. Related works are discussed in Section V. Section **??** concludes the paper.

II. The Problem

In this paper, we consider a social network as an undirected, unweighted graph $G = (V, E, L)$, where V represents a set of vertices, $E \subseteq V \times V$ is a set of edges without labels, L denotes a set of *categorical labels* or simply *labels* on V. $L(v) \subseteq L$ denotes a set of labels of a vertex $v \in V$. For example in Figure 1(a), $L(v_{Toby}) = \{Student, Laptop\}$ and $L(v_{Lily}) = \{Professor, Mouse\}$. The *1-neighborhood* of a vertex v, denoted by $N^1(v)$, is the induced subgraph of the neighbors of v. For example, Figure 2 depicts the 1-neighborhood of $Toby$, i.e., $N^1(v_{Toby})$.

The research problem studied in this paper is to transform a given social network G with labeled vertices into a k-anonymous version while preserving as many frequent spatterns as possible. The notions of k-anonymity and frequent spatterns are formally defined as follows.

A. Privacy Model

Suppose an adversary knows the 1-neighborhood network structure of a target victim as background knowledge, and wants to identify the vertex of the target victim in G. To thwart this identity attack, we employ the privacy model of k-anonymity on social network [2]. The general idea is to ensure that the 1-neighborhood network structure of any vertex in a social network G is isomorphically similar to the 1-neighborhood network structure of at least $k-1$ other vertices in G.

Definition II.1 (*k-anonymous social network*)**.** Let G be a social network. Let k be a privacy threshold specified by social network data holder. A vertex v in G is *k-anonymous* if there exists at least $k-1$ other vertices $u_1, \ldots, u_{k-1} \in V$ such that $N^1(v)$ and $N^1(u_1), \ldots, N^1(u_{k-1})$ are isomorphic. A social network G is *k-anonymous* if every vertex $v \in V$ in G is k-anonymous [2]. ■

For example, the social network in Figure 1(b) satisfies 2-anonymity.

B. Frequent Spatterns

Consider a social network $G = (V, E, L)$ as defined above. Below, we formally define the notions of *sharing pattern* (*spattern*), *maximal subgraph*, and *frequent spattern* [4].

Definition II.2 (*Spattern*)**.** A *sharing pattern*, or simply *spattern*, p is a non-empty set of labels, $p \subseteq L$ and $p \neq \emptyset$. A vertex $v \in V$ *contains* a pattern p if $p \subseteq L(v)$. ■

To determine the popularity of a pattern within a community, we define the notion of maximal subgraph of a spattern.

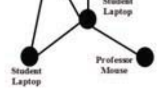


Fig. 2. 1-neighborhood of $Toby$

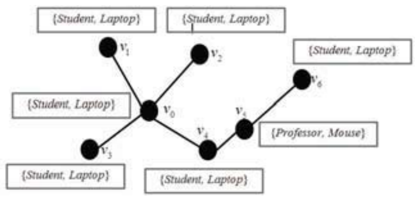


Fig. 3. Example of maximal subgraph of a spattern

Definition II.3 (*Maximal subgraph of spattern*)**.** A connected subgraph G_s of G is a *maximal subgraph of a spattern* p, denoted by $G_s(p)$, if $\forall v \forall u((v \in G_s \rightarrow p \subseteq L(v)) \wedge (u \in N^1(v) \wedge u \notin G_s \rightarrow p \nsubseteq L(u)))$. The *support* of spattern p in G_s, denoted by $Sup(p|G_s)$, is the number of vertices in G_s containing p. ■

The first condition $v \in G_s \rightarrow p \subseteq L(v)$ states that all vertices in G_s contain the pattern p. The second condition $u \in N^1(v) \wedge u \notin G_s \rightarrow p \nsubseteq L(u)$ states that the subgraph containing the pattern p is maximal.

Example II.1. Consider Figure 1(a). Vertex v_{Toby} contains spatterns $\{Student\}$, $\{Laptop\}$, and $\{Student, Laptop\}$. Figure 3 depicts the two maximal subgraphs G_1 (composed of v_0, v_1, v_2, v_3, and v_4) and G_2 (composed of v_6) of spattern $\{Student, Laptop\}$. $Sup(\{Student, Laptop\}|G_1) = 5$ and $Sup(\{Student, Laptop\}|G_2) = 1$. G_3 (composed of v_0, v_1, v_2, and v_3) is not a maximal subgraph of a spattern since v_4 is connected to G_3, meanwhile, v_4 and G_3 have the same pattern $\{Student, Laptop\}$. ■

Definition II.4 (*Frequent spattern*)**.** Let $G_1(p), \ldots, G_m(p)$ be all the maximal subgraphs of a spattern p in G. The *support* of a spattern p in G, denoted by $Sup(p)$, is the $max(Sup(p|G_1(p)), \ldots, Sup(p|G_m(p)))$. Let $MinSup$ be the minimum support threshold specified by the social network data holder. A spattern p is a frequent spattern in G if $Sup(p) \geq MinSup$. ■

Definition II.5 (*Maximal frequent spattern*)**.** A frequent spattern is a *maximal frequent spattern* in G if any of its proper superset is not frequent in G. ■

Example II.2. Consider Figure 1(b) with the additional edges. Suppose $MinSup = 5$. $\{Laptop\}$, $\{Student\}$, and $\{Student, Laptop\}$ are frequent spatterns. $\{Professor\}$, $\{Mouse\}$, and $\{Professor, Mouse\}$ have supports 4, so they are not frequent spatterns. ■

C. Problem Statement

Definition II.6 (*Social Network Anonymization for Frequent Spatterns*)**.** Given a social network G with labeled vertices, a k-anonymity requirement, and a minimum support threshold $MinSup$, the *problem of anonymization of social network for frequent spatterns* is to transform G to satisfy the given k-anonymity requirement while preserving as many frequent spatterns as possible. ■

The problem of achieving k-anonymity in a social network has been proven to be NP-hard [2]. Thus, we propose a heuristic approach to tackle the problem.

Algorithm 1 Overview of the Anonymization Algorithm

```
Input: Social network $G = (V, E, L)$ and anonymization threshold $k$;
Output: $k$-anonymous social network;
1: $VList \leftarrow V$;
2: Sort $VList$ by degrees in descending order;
3: while $VList \neq \emptyset$ do
4:     $TopK \leftarrow$ first $k$ disjointed vertices in $VList$;
5:     Call $SmoothingDegree(TopK)$;
6:     Call $MakeIsomorphic(TopK, AffectedV)$;
7:     $VList.Remove(TopK)$;
8:     $VList.InsertAndSort(AffectedV)$;
9: end while
```

III. The Anonymization Method

In this section, we present a method to anonymize the social network $G = (V, E, L)$ to achieve k-anonymity. Algorithm 1 provides an overview of the algorithm. According to the power law degree distribution [8], most of the vertices in a social network have low degrees, and only few vertices have large degrees. Therefore, our proposed method starts anonymization from the vertices with the largest degrees. The vertices with lower degrees are much easier to anonymize. The algorithm first sorts the vertices V by their degrees in descending order, stores the sorted vertices in $VList$, iteratively processes the first k disjointed vertices in $VList$, denoted by $TopK$, and then removes $TopK$ from $VList$. Each iteration of processing the $TopK$ vertices consists of two steps. The first step is to transform the $TopK$ vertices to have the same degree. The second step is to extract the 1-neighborhood of the $TopK$ vertices and add edges to make them isomorphic. The challenge is that making a group of vertices isomorphic may break the isomorphism of some previously processed vertices. Thus, the algorithm has to add the affected vertices, denoted by $AffectedV$, back to $VList$. This process repeats until $VList$ becomes empty. The details of the two steps, namely $SmoothingDegree$ (Line 5) and $MakeIsomorphic$ (Line 6), are described as follows.

A. Degree Smoothing

Given k disjointed vertices, denoted by $TopK$, that are sorted by degree in descending order, the goal of this step is to make them having the same degree by adding edges. Algorithm 2 describes the general idea of this procedure. Let v_0 be the first vertex of $TopK$, i.e., the one with the largest degree among the k vertices. For each vertex v_i in $TopK$, the procedure computes the number of edges, denoted by d, required to be added to v_i, and heuristically selects d vertices with the least degrees from V. Vertices with low degrees are preferable because they can be efficiently obtained from the end of the $VList$, and they are relatively easy to smoothen, if necessary, in later iterations. Due to the power law degree distribution [8], it is very likely that more than d vertices have the same least degree. The question is how to select the vertices from these candidates for adding edges with minimal impacts on the frequent spatterns.

Adding edges increases the support of some spatterns. Consequently, some spatterns that were not frequent before the anonymization may become frequent after the anonymization,

Algorithm 2 $SmoothingDegree(TopK)$

```
Input: $TopK$ sorted by degrees in descending order;
Output: $TopK$ with the same degree;
1: $v_0 = TopK.popfirst()$;
2: while $TopK \neq \emptyset$ do
3:     $v_i = TopK.popfirst()$;
4:     $d = degree(v_0) - degree(v_i)$;
5:     Add $d$ edges to $v_i$ based on minimum $Cost$;
6: end while
```

resulting in some false frequent spatterns. Thus, the heuristic function for selecting the target vertices should minimize the increase of the support. In other words, the function selects a vertex v_j with a label that has minimal overlap with the label of vertex v_i:

$$Cost(v_i, v_j) = |L(v_i) \cap L(v_j)| \quad (1)$$

where $L(v_i)$ and $L_(v_j)$ denote the labels of v_i and v_j, respectively. If more than d vertices share the same least degree and $Cost$, the algorithm randomly chooses d of them.

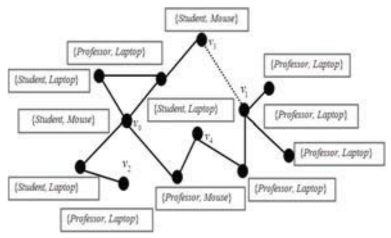


Fig. 4. Degree smoothing with $k = 2$

Example III.1. Consider Figure 4 with $k = 2$. After sorting all vertices in degree descending order, v_0 has the largest degree $degree(v_0) = 4$, v_1 is the vertex with the second largest degree, with $degree(v_1) = 3$, that is not connected with v_0. Thus, $d = degree(v_0) - degree(v_1) = 1$, and one edge has to be added between v_1 and an another vertex, which has the least degree. In this example, both v_2 and v_3 has degree 1; therefore, we choose the one minimum overlap in their labels: $Cost(v_1, v_2) = |\{Professor, Laptop\} \cap \{Professor, Laptop\}| = 2$ and $Cost(v_1, v_3) = |\{Professor, Laptop\} \cap \{Student, Mouse\}| = 0$. Since $Cost(v_1, v_3) < Cost(v_1, v_2)$, we add an edge between v_1 and v_3. ■

B. Making Isomorphic

After smoothing the degree of the $TopK$ vertices, the next step is to make them isomorphic. Specifically, the goal of this step is to add edges to the 1-neighborhood of $TopK$ vertices in order to make them isomorphic. Similar to the technique of *DFS Code* [9], we employ a technique called *BFS coding* to identify the missing edges. Algorithm 3 describes the steps. The general idea is to compare the 1-neighborhood of the first vertex, denoted by $N^1(v_0)$, with the 1-neighborhood of each of the remaining vertices, denoted by $N^1(v_x)$, in $TopK$, and compare their BFS codes to determine and to add the missing edges (Lines 5-6). The next task is to identify the previously k-anonymized vertices that are ruined by the newly added edges. In other words, these affected vertices, denoted by $AffectedV$, have to be put back to the $VList$ for re-anonymization. Lines 7-16 describe this detection process. A previously k-anonymized vertex is affected by the newly added edges if it satisfies one of the following conditions:

Algorithm 3 $MakeIsomorphic(TopK, AffectedV)$

```
Input: $TopK$ sorted by degrees in descending order;
Output: $TopK$ with isomorphic 1-neighborhood;
1: $v_0 = TopK.popfirst()$;
2: for $i := 1$ to $2$ do
3:     for each $v_x \in TopK$ do
4:         if $BFS(N^1(v_0)) \neq BFS(N^1(v_x))$ then
5:             Add edges to $N^1(v_0)$ based on $BFS(N^1(v_x))$;
6:             Add edges to $N^1(v_x)$ based on $BFS(N^1(v_0))$;
7:             for each $v_a \notin VList$ do
8:                 if $v_a \in N^1(v_0) \ \vee \ v_a \in N^1(v_x)$ then
9:                     $AffectedV.Add(v_a)$;
10:                    for each $v_y \in TopK$ do
11:                        if $v_y \in AnonymousGroup(v_a)$ then
12:                            $AffectedV.Add(v_y)$;
13:                        end if
14:                    end for
15:                end if
16:            end for
17:        end if
18:    end for
19: end for
```

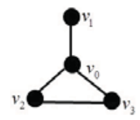


Fig. 5. BFS tree

1) the vertex is a neighbor of v_0 or a neighbor of v_x (Line 9), or
2) the vertex is in $TopK$ and shares the same k-anonymous group with another vertex v_a such that v_a is a neighbor of v_0 or a neighbor of v_x (Lines 10-14).

After the first round, the 1-neighborhood of v_0 is the supergraph of others. Then the algorithm runs the same steps once again to ensure the structure of the 1-neighborhood of all vertices in $TopK$ are copies of the 1-neighborhood of v_0. In the rest of this section, we focus on how to compute the BFS code the 1-neighborhood of a given vertex, and how to compute two BFS codes in order to determine the missing edges.

To facilitate the comparison of the structure of graphs, we use a *breath-first search tree* (*BFS-tree*) to encode the two graphs and compare their BFS codes. The general idea is to traverse the vertices using a breath-first search by following the subscripts of the vertices. Consider Figure 5 as an example. We start the BFS coding from the vertex with the largest degree, which is v_0, followed by v_1, v_2, v_3 and finally the edges between v_2 and v_3. Thus, The 1-neighborhood BFS Code of v_0, denoted by $BFS(N^1(v_0))$, is (01020323).

Next, we can determine the missing edges between two subgraphs by comparing their BFS codes. The following example illustrates the idea.

Example III.2. Consider Figure 6. The encoding always starts from 0, so $v_5 - v_8$ in Figure 6(b) become $v_0 - v_4$. The BFS Codes of $N^1(v_0)$ and $N^1(v_5)$ are (010203041423) and (0102030434), respectively. By comparing the two BFS codes, we know that (14) and (23) are not in $N^1(v_5)$ and (34) are not in $N^1(v_0)$. Therefore, we add an edge between v_3 and v_4 in $N^1(v_0)$ and add two edges between v_6 and v_9 and between v_7 and v_8 in $N^1(v_5)$. After adding these edges, the two graphs become isomorphic. ■

The following example illustrates how to isomorphize the 1-neighborhood of three vertices.

Example III.3. Consider the 1-neighborhoods of v_0, v_5, and v_{10} in Figure 7(a). To make them isomorphic, we start from $N^1(v_0)$ and iteratively compare it with $N^1(v_5)$ and $N^1(v_{10})$. By comparing $BFS(N^1(v_0))$ with $BFS(N^1(v_5))$ and $BFS(N^1(v_{10}))$, we add an edge between v_1 and v_2 and another edge between v_2 and v_3 as shown in Figure 7(b). Yet, the three 1-neighborhoods are not isomorphic yet because $N^1(v_5)$ and $N^1(v_{10})$ are different. Since $N^1(v_0)$ must be a supergraph of $N^1(v_5)$ and $N^1(v_{10})$. We once again compare $BFS(N^1(v_0))$ with $BFS(N^1(v_5))$ and $BFS(N^1(v_{10}))$, add an edge between v_7 and v_8 as depicted in Figure 7(c). ■

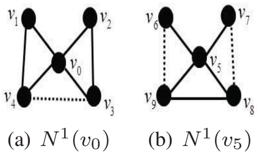
C. Analysis and Discussion

In this section, we analyze the computational complexity of the aforementioned procedures and discuss the limitations of our proposed algorithm.

In the $SmoothingDegree$ algorithm, the heuristic function first selects the vertices with the lowest degree and then computes the impact on spatterns. The computational complexity of the algorithm is $O(knlogn)$, where k is the anonymization threshold, n is the number of the vertices with the lowest degree. In the $MakeIsomorphic$ algorithm, we use BFS code to encode the 1-neighborhood a given vertex. We also need to find those affected vertices and put them into $VList$ again. Considering the worst case, the computational complexity of the algorithm is $O(k^3 \times |V| + k \times |V|^2)$, where k is the anonymization threshold and $|V|$ is the number of vertices in $N^1(v_i)$, where $v_i \in TopK$.

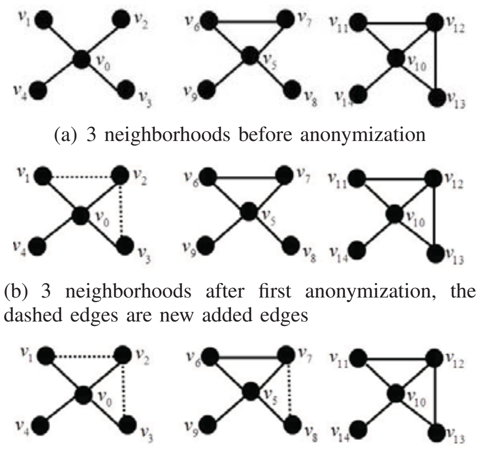
An alternative solution to tackle the problem is to first extract the frequent spatterns from the raw social network graph. Then at each iteration, the method chooses a vertex for adding edge with a heuristic function that minimizes the impact on the frequent spatterns. This alternative solution suffers from two shortcomings:

1) Extracting frequent spatterns is computationally expensive and doing so will significantly increases the complexity of the anonymization algorithm.



(a) $N^1(v_0)$ (b) $N^1(v_5)$

Fig. 6. Making isomorphic



(a) 3 neighborhoods before anonymization

(b) 3 neighborhoods after first anonymization, the dashed edges are new added edges

(c) 3 neighborhoods after second anonymization, the dashed edges are new added edges

Fig. 7. Example of 3-neighborhood anonymization

2) The notion of frequent spatterns depends on the user-specified minimum support threshold. In real-life data publishing, it is difficult for the data holder to determine an appropriate minimum threshold in advance on behalf of the data recipient. Also, the evaluation must then depend on the specified minimum support threshold.

Supported by the experimental results, we would like to emphasize that our proposed algorithm can effectively preserve the (maximal) frequent itemsets although the algorithm does not actually extract the frequent itemsets from the social network.

We would also like to provide a justification on why we choose the vertices with the lowest degree in the $SmoothingDegree$ algorithm. First, adding edges between vertices with large degrees may affect the previously anonymized vertices and increase the chance of re-anonymization, which degrades the efficiency and affects the diameter of the social network [2]. Second, since the BFS coding technique can only deal with the disjointed vertices, adding edges between vertices with large degrees will corrupt the disjointed vertices and increase the difficulty to achieve k-anonymity.

Though k-anonymity technique is effective to thwart neighborhood attack on social network, our approach has some limitations. First, our approach can only deal with 1-neighborhood, if an adversary has the background knowledge beyond 1-neighborhood, the k-anonymous social network may still suffer from neighborhood attacks. Second, we assume that the adversary has the background knowledge of the structure of the social network. If the adversary has both the structural background knowledge of the social network and the partial label information of the target victim, our approach is insufficient for this kind of attack.

IV. Experimental Evaluation

The objective of the experiments is to evaluate the performance of the proposed algorithm with respect to the data quality of the anonymous social network. The experiments

Algorithm 4 Relabel($v, VList$)

```
Input: A vertex $v$ and a list of vertices excluding $v$
Output: A relabeled social network
1: for $i = 1$ to $M$ do
2:     for each $x \in N^1(v)$ do
3:         if $L_i(x) == L_i(v)$ then
4:             $labelNum(L_i(x)) \leftarrow labelNum(L_i(v))$;
5:             Relabel($x, VList - x$);
6:         end if
7:     end for
8: end for
```

were conducted on a PC with Core i7 2GHz CPU with 8GB memory running on Windows 7.

A. Datasets

We conducted the experiments on three real-life datasets, namely *Gnutella05*[1], *Gnutella08*[2] [10], and *Adult*[3]. *Gnutella05* and *Gnutella08* are snapshots of the Gnutella peer-to-peer file sharing network in August 2002. In both datasets, vertices represent host computers and the edges represent the connections. *Gnutella05* has 8,846 vertices and 31,839 edges. *Gnutella08* has 6,301 vertices and 20,777 edges. We converted the original directed graphs into indirected graphs for our experiment. As the two datasets have no labels, we used the *Adult* dataset, which has been previously employed in [11][12], to synthesize the vertex labels. *Adult* has 45,222 records on 8 categorical attributes.

As the numbers of records in *Adult* are different from the number of vertices in *Gnutella05* and *Gnutella08*, we sequentially associated each record in *adult* with *Gnutella05* and *Gnutella08* based on the order given in the raw datasets. The numerical attributes in *Adult* dataset were removed.

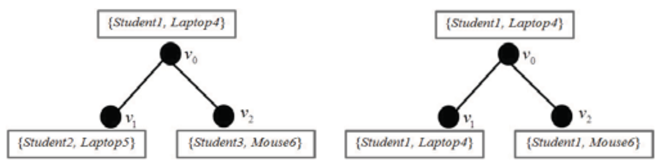
B. Frequent Spatterns Extraction

To evaluate the data utility on frequent spatterns, we measure the change of the frequent spatterns before and after anonymization. We use a tool called *MAFIA* [13] to extract the frequent spatterns.

In frequent itemsets mining, the support of an itemset is simply the number of transactions containing the itemset. However, in frequent spatterns mining, we *cannot* simply treat the label of each vertex as a transaction because the support of a spattern is the number of vertices in a maximal subgraph of the spattern. (See Definition II.3.) In other words, even two disjoint vertices have the same label, the support of the label is only 1. Thus, we need to first relabel the vertex labels such that two labels share the same label number only if they have the same label and their vertices are connected.

Suppose the label of each vertex is sorted in alphabetical order. Let $L_j(v)$, a *sub-label* of $L(v)$, be the j^{th} label of vertex v. For example, v_{Toby} has $L_1(v_{Toby}) = \{Student\}$ and $L_2(v_{Toby}) = \{Laptop\}$ in Figure 1(a). The first step is

[1]http://snap.stanford.edu/data/p2p-Gnutella05.html
[2]http://snap.stanford.edu/data/p2p-Gnutella08.html
[3]http://archive.ics.uci.edu/ml/datasets/Adult



(a) sub-label after assignment (b) relabeled graph with spatterns

Fig. 8. Relabeling

to assign a temporary distinct sub-label number to each sub-label of every vertex, denoted by $labelNum(L_i(v))$, and then call the depth-first recursive function $Relabel(v, V-v)$, where v can be any vertex in V, as described in Algorithm 4. The general idea is iterate through each sub-label of every neighbor of a given vertex v and copy the sub-label number from v to its neighbor x if their sub-labels are the same. To avoid relabeling the same sub-label more than once, we use a boolean flag to skip the visited vertices.

Example IV.1. Consider Figure 8(a). The label in each vertex contains two sub-labels: $L_1(v)$ and $L_2(v)$. We first assign a distinct sub-label number to every sub-label. For example, $labelNum(L_1(v_2)) = 3$ and $labelNum(L_2(v_1)) = 5$. Next, we start a depth-first search on v_0 for L_1 since it has the lowest sub-label number. v_0, v_1, and v_2 are connected and $L_1(v_0)$, $L_1(v_1)$, and $L_1(v_2)$ are the same, so we reassign the sub-label numbers of $Student$ in v_1 and v_2 to $labelNum(L_1(v_1)) = 1$ and $labelNum(L_1(v_2)) = 1$, respectively. Similarly, we reassign the sub-label number of $Laptop$ in v_1 to $labelNum(L_2(v_1)) = 4$. Figure 8(b) depicts the relabeled graph. ■

After relabeling the vertices, each vertex is transformed into a transaction and its sub-label numbers are treated as transaction items. Then MAFIA is applied to extract the frequent spatterns.

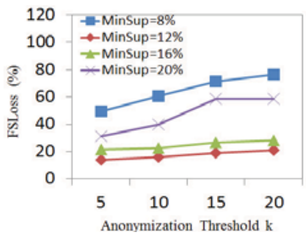
C. Data Utility on Frequent Spatterns

The first experiment is to evaluate the impact of anonymization on frequent spatterns. The utility loss is calculated by $FSLoss = \frac{A-B}{B}$, where B and A denotes the number of frequent spatterns extracted before and after anonymization, respectively. The value of $FSLoss$ is non-negative. The higher value of $FSLoss$ means the higher number of false positive frequent spatterns, implying higher utility loss.

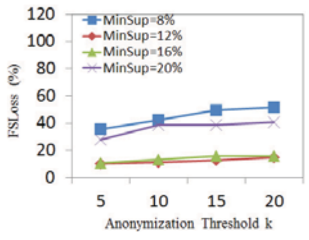
Figures 9 depicts the utility loss on frequent spatterns with anonymization threshold $5 \leq k \leq 20$, and minimum support $MinSup = 8\%, 12\%, 16\%, 20\%$ on *p2p-Gnutella08* and *p2p-Gnutella05*. For example, at $MinSup$ = 16%, $FSLoss = 21.3\%, 22.7\%, 26.7\%, 28\%$ for $5 \leq k \leq 20$, respectively. This result suggests that as k increase, more fake edges have to be added in order to achieve the k-anonymity requirement, resulting in higher $FSLoss$. Yet, the impact of anonymization on $FSLoss$ is mild.

V. Related Works

Privacy threats on social network data can be summarized into three types, namely *identity disclosure*, *attribute disclosure*, and *link re-identification*, depending on the adversary's



(a) *p2p-Gnutella08*



(b) *p2p-Gnutella05*

Fig. 9. Utility loss on frequent spatterns

background knowledge on the social network data. For identity disclosure, the attack goal is to identify the vertex that represents a target victim. For attribute disclosure, the attack goal is to identify or infer some sensitive information about a target victim. For link re-identification, the attack goal is to identify sensitive relationships of a target victim. We briefly review the related works on thwarting identity disclosures with anonymization technique in the enhanced model [1], which represents the social network data as a graph in which labeled vertices denote participants and their associated information such as jobs and purchased items and edges denote the relationships between participants.

Existing anonymization techniques for thwarting identity disclosure on social networks are primarily classified into three categories: adding or removing edges [2] [14] [15], generalization [16] [17], and randomization [18] [19]. However, generalization and randomization technique are not applicable to the problem of preserving frequent sharing patterns studied in this paper because the generalized graph is a transformation of the original graph and a randomized graph produces noised patterns. Our approach falls into the category of adding edges. Zhou and Pei [2] generalize node labels and inserted edges into the network to achieve k-neighborhood. Cheng et al. [14] introduce a k-isomorphism technique to thwart structural attack in social networks, ensuring that on social network, even if the adversary knows the information of an individual, or the relationship among the individuals, privacy will still be protected. Bonchi et al. [15] describe a k-obfuscation model which ensures that an adversary cannot infer the vertex in the obfuscated graph based on the vertex of its original graph. However, the aforementioned approaches employ traditional utility measures, such as graph topological properties, graph spectral properties, and aggregate network queries, to evaluate the information utility of the anonymized social network. In contrast, our proposed method aims at preserving frequent itemsets in the anonymization process and evaluates the information utility from a different aspect.

VI. Acknowledgement

The research is supported in part by the Discovery Grants (356065-2013) from the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu, "Privacy-preserving data publishing: A survey of recent developments," *ACM Computing Surveys*, vol. 42, no. 4, pp. 14:1–14:53, June 2010.

[2] B. Zhou and J. Pei, "Preserving privacy in social networks against neighborhood attacks," in *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, 2008, pp. 506–515.

[3] L. Sweeney, "k-anonymity: a model for protecting privacy," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[4] M. Fukuzaki, M. Seki, H. Kashima, and J. Sese, "Finding itemset-sharing patterns in a large itemset-associated graph," in *Proceedings of the 14th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining*, 2010, pp. 147–159.

[5] P. Samarati and L. Sweeney, "Protecting privacy when disclosing information: K-anonymity and its enforcement through generalization and suppression," SRI International, Tech. Rep., 1998.

[6] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam, "L-diversity: Privacy beyond k-anonymity," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, March 2007.

[7] K. Wang, B. C. M. Fung, and P. S. Yu, "Handicapping attacker's confidence: an alternative to k-anonymization," *Knowledge and Information Systems*, vol. 11, pp. 345–368, April 2007.

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 1999, pp. 251–262.

[9] X. Yan and J. Han, "gSpan: Graph-based substructure pattern mining," in *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, 2002, pp. 721–724.

[10] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Graph evolution: Densification and shrinking diameters," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, March 2007.

[11] N. Mohammed, B. C. M. Fung, and M. Debbabi, "Anonymity meets game theory: secure data integration with malicious participants," *Very Large Data Bases Journal (VLDBJ)*, vol. 20, no. 4, pp. 567–588, August 2011.

[12] N. Mohammed, B. C. M. Fung, P. C. K. Hung, and C.-K. Lee, "Centralized and distributed anonymization for high-dimensional healthcare data," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 4, no. 4, pp. 18:1–18:33, October 2010.

[13] D. Burdick, M. Calimlim, and J. Gehrke, "Mafia: A maximal frequent itemset algorithm for transactional databases," in *Proceedings of the 17th International Conference on Data Engineering*, 2001, pp. 443–452.

[14] J. Cheng, A. W.-C. Fu, and J. Liu, "K-isomorphism: Privacy preserving network publication against structural attacks," in *Proceedings of the ACM SIGMOD International Conference on Management of data*, 2010, pp. 459–470.

[15] F. Bonchi, A. Gionis, and T. Tassa, "Identity obfuscation in graphs through the information theoretic lens," in *Proceedings of the IEEE 27th International Conference on Data Engineering (ICDE)*, 2011, pp. 924–935.

[16] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis, "Resisting structural re-identification in anonymized social networks," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 102–114, 2008.

[17] A. Campan and T. M. Truta, "A clustering approach for data and structural anonymity in social networks," in *In Proceedings of the 2nd ACM SIGKDD International Workshop on Privacy, Security, and Trust in KDD Workshop*, 2008, pp. 1–10.

[18] M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava, "Anonymizing social networks," Computer Science Department, University of Massachusetts Amherst, Tech. Rep. 07-19, 2007.

[19] X. Wu, X. Ying, K. Liu, and L. Chen, *A Survey of Algorithms for Privacy-Preservation of Graphs and Social Networks.* Kluwer Academic Publishers, 2009, ch. Managing and Mining Graph Data.