Preserving Privacy and Frequent Sharing Patterns for Social Network Data Publishing

Benjamin C. M. Fung
CIISE, Concordia University
Montreal, QC, Canada
tfung@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
ljiamin@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 goal 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 the 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 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 [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
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), \( L(v_{\text{Toby}}) = \{\text{Student, Laptop}\} \) and \( L(v_{\text{Lily}}) = \{\text{Professor, Mouse}\} \). 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., \( N^1(v_{\text{Toby}}) \).

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 patterns as possible. The notions of \( k \)-anonymity and frequent patterns 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 \((k\text{-anonymous social network})\). 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 Patterns

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

Definition II.2 \((\text{Pattern})\). A sharing pattern, or simply pattern, \( p \) 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 pattern.

Fig. 2. 1-neighborhood of Toby
are not frequent spatterns. Maximal frequent spattern has been proven to be NP-hard [2]. Thus, we propose a subgraph.

Consider Figure 1(a). Vertex contains spatterns \{Student\}, \{Laptop\}, and \{Student, Laptop\}. Figure 3 depicts the two maximal subgraphs of a spattern \{Student, Laptop\}. The support of a 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 \(u \in N^1(v) \wedge u \notin G_s \rightarrow p \not\subseteq L(u)\) states that the subgraph containing the pattern \(p\) is maximal.

Example II.1. Consider Figure 1(a). Vertex \(v_{Top}\) 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 \(v_0\) 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 \(\min\sup\) be the minimum support threshold specified by the social network data holder. A spattern \(p\) is a frequent spattern in \(G\) if \(\sup(p) \geq \min\sup\).

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 \(\min\sup = 5\). \{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 \(\min\sup\), 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. \(V\text{-List} \leftarrow V\);
2. Sort \(V\text{-List}\) by degrees in descending order;
3. while \(V\text{-List} \neq \emptyset\) do
4. \(\text{TopK} \leftarrow \text{first } k\text{ disjointed vertices in } V\text{-List};
5. \Call{SmoothingDegree}{TopK};
6. \Call{MakeIsomorphic}{TopK, AffectedV};
7. \Call{VList.Remove}{TopK};
8. \Call{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 \(V\text{-List}\), iteratively processes the first \(k\) disjointed vertices in \(V\text{-List}\), denoted by \(\text{TopK}\), and then removes \(\text{TopK}\) from \(V\text{-List}\). Each iteration of processing the \(\text{TopK}\) vertices consists of two steps. The first step is to transform the \(\text{TopK}\) vertices to have the same degree. The second step is to extract the 1-neighborhood of the \(\text{TopK}\) 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 \(\text{AffectedV}\), back to \(V\text{-List}\). This process repeats until \(V\text{-List}\) becomes empty. The details of the two steps, namely \(\text{SmoothingDegree}\) (Line 5) and \(\text{MakeIsomorphic}\) (Line 6), are described as follows.

A. Degree Smoothing

Given \(k\) disjointed vertices, denoted by \(\text{TopK}\), that are sorted by degree in descending order, the goal of the first 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 \(\text{TopK}\), i.e., the one with the largest degree among the \(k\) vertices. For each vertex \(v_i\) in \(\text{TopK}\), the procedure computes the number of degrees, 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 \(V\text{-List}\), 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: \hspace{1em} $v_i = TopK.popfirst();$
4: \hspace{1em} $d = degree(v_0) - degree(v_i);$\hspace{1em} \hspace{1em} 
5: \hspace{1em} Add $d$ edges to $v_i$ based on minimum Cost;
6: \hspace{1em} end while

resulting in some false frequent patterns. 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 $v_i$: 

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

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 degree and Cost, the algorithm randomly chooses $d$ of them.

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

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

**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: \hspace{1em} for each $v_x \in TopK$ do
4: \hspace{2em} if $BFS(N^1(v_x)) \neq BFS(N^1(v_x))$ then
5: \hspace{3em} Add edges to $N^1(v_x)$ based on BFS($N^1(v_x)$);
6: \hspace{3em} Add edges to $N^1(v_x)$ based on BFS($N^1(v_x)$);
7: \hspace{1em} end if
8: \hspace{1em} end for
9: end for

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:

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 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 $N^1(v_5)$ and $N^1(v_5)$ are (01020341234) and (010203041423), 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 $N^1(v_5)$. Therefore, we add an edge between $v_3$ and $v_4$ in $N^1(v_5)$ 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 $O(nk\log n)$, 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 $O(kd \times |V| + k \times |V|^2)$, 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.

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
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, Gnutella08 [10], and Adult. 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 undirected 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, \( v_{\text{Enable}} \) has \( L_1(v_{\text{Enable}}) = \{ \text{Student} \} \) and \( L_2(v_{\text{Enable}}) = \{ \text{Laptop} \} \) in Figure 1(a). The first step is to assign a temporary distinct sub-label number to each sub-label of every vertex, denoted by \( \text{labelNum}(L_i(v)) \), and then call the depth-first recursive function \( \text{Relabel}(v, V\text{List} - v) \), where \( v \) can be any vertex in \( V \), as described in Algorithm 4. The general idea is to 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, \( \text{labelNum}(L_1(v_0)) = 3 \) and \( \text{labelNum}(L_2(v_0)) = 5 \). 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 \( \text{Student} \) in \( v_1 \) and \( v_2 \) to \( \text{labelNum}(L_1(v_1)) = 1 \) and \( \text{labelNum}(L_1(v_2)) = 1 \), respectively. Similarly, we reassign the sub-label number of \( \text{Laptop} \) in \( v_1 \) to \( \text{labelNum}(L_2(v_1)) = 4 \). 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 \( \text{FSLoss} = \frac{A - B}{A} \), where \( B \) and \( A \) denotes the number of frequent spatterns extracted before and after anonymization, respectively. The value of \( \text{FSLoss} \) is non-negative. The higher value of \( \text{FSLoss} \) 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 \( \text{MinSup} = 8\% \), \( 12\% \), \( 16\% \), \( 20\% \) on p2p-Gnutella08 and p2p-Gnutella05. For example, at \( \text{MinSup} = 16\% \), \( \text{FSLoss} < 21\% \), \( 22\% \), \( 26\% \), \( 28\% \) for \( 5 \leq k \leq 20 \), respectively. This result suggests that as \( k \) increase, more edge lines have to be added in order to achieve the \( k \)-anonymity requirement, resulting in higher \( \text{FSLoss} \). Yet, the impact of anonymization on \( \text{FSLoss} \) 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

\[ \text{Algorithm 4 Relabel}(v, V\text{List}) \]

**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. \( \text{labelNum}(L_i(x)) \leftarrow \text{labelNum}(L_i(v)) \);
5. \( \text{Relabel}(x, V\text{List} - x) \);
6. end if
7. end for
8. end for

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

Fig. 8. Relabeling

Examples include:

- **[Student, Laptop1]**
- **[Student, Laptop2]**
- **[Student, Laptop3]**
- **[Student, Laptop4]**
- **[Student, Myspace]**

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


