Abstract:
Large amount of personal social information is collected and published due to the rapid development of social network technologies and applications, and thus it is quite essential to take privacy preservation and prevent sensitive information leakage. Most of current anonymizing techniques focus on the preservation to privacies, but cannot provide accurate answers to utility queries even at a high price. To solve the problem, a novel anonymizing approach, called splitting anonymization, is introduced in this paper to point against the contradiction of privacy and utility. This approach provides a high level preservation to the privacy of social network data that is unknown to attackers, which avoids the low utility caused by the enforced noises on knowledge that is already known to the attackers. Social network processed by splitting anonymization can refuse any direct attack, and these strategies are also safe enough to indirect attacks which are usually more dangerous than direct attacks. Finally, strict theoretical analysis and large amount of evaluation results based on real datasets verified the design of this paper.
Splitting Anonymization: A Novel Privacy Preserving Approach of Social Network

Yongjiao Sun*, Ye Yuan*, Guoren Wang* and Yurong Cheng*
*Northeastern University, China

Received: 12 Apr 2013/ Revised: 05 May 2015/Accepted: 21 May 2015

Abstract

Large amount of personal social information is collected and published due to the rapid development of social network technologies and applications, and thus it is quite essential to take privacy preservation and prevent sensitive information leakage. Most of current anonymizing techniques focus on the preservation to privacies, but cannot provide accurate answers to utility queries even at a high price. To solve the problem, a novel anonymizing approach, called splitting anonymization, is introduced in this paper to point against the contradiction of privacy and utility. This approach provides a high level preservation to the privacy of social network data that is unknown to attackers, which avoids the low utility caused by the enforced noises on knowledge that is already known to the attackers. Social network processed by splitting anonymization can refuse any direct attack, and these strategies are also safe enough to indirect attacks which are usually more dangerous than direct attacks. Finally, strict theoretical analysis and large amount of evaluation results based on real datasets verified the design of this paper.

Keywords social network, privacy preservation, attack, anonymization, predictable error

1 Introduction

To meet the requirement of research on data mining and data analysis, social network data usually should be available for publication [1][2][3]. However, as there is large amount of user’s privacy information contained in social network data, it is necessary to make some modification on real data in order to ensure the security of user’s privacy. If the modification is too much, the utility of social network data will be reduced. On the other hand, if the modification is not enough, the privacy information will not be well protected. That is to say, a tradeoff exists between privacy and utility. Thus, anonymization which can retain the character of real data as much as possible with the premise of privacy protection are exactly what researchers try to work out.

Social network data is usually presented in graph \(G = \{V,E,U\}\) and the detailed definition is shown in Definition 2.1. The existing anonymization algorithms can be classified into four classes, simple anonymizing, anonymizing based on vertex clustering, anonymizing based on edge clustering, and anonymizing based on bigraph.

Simple anonymizing uses a method of changing the labels of vertices to hide the privacy information. That is, after the process of simple anonymizing, \(G = \{V,E,U\}\) will be changed into \(G_H = \{V,E,U'_H\}\), where \(U'_H = \{u'_1,u'_2,...,u'_n\}\) is a set of meaningless labels generated randomly. Then, some privacy information in original data sets is protected. This method is apparently easy to realize, but it cannot provide sufficient protection in active attack [4]. Furthermore, in respect of utilization, as large quantity of labels are deleted, it is hard to do clustering analysis to vertices.

The main idea of anonymizing based on vertex or edge clustering [5][6][7] is to generalize the privacy information. Specifically, in this method, it is preferable to use collective information instead of individual information. For example, some vertices or edges with similar labels are replaced by one new vertex or edge with a new general label. The advantage of this method is that it does not change the framework of original graph data and reduce the storage space.
However, this method inversely loses some detailed information of original data, which will lead to an inaccurate result of query in respect of utilization.

Anonymizing based on bigraph is to partition the vertex set $V$ into $V_0$ and $V_1$, and make sure that $E \subseteq \{(v_0, v_1) \mid v_0 \in V_0, v_1 \in V_1\}$. In other words, the original graph should be changed into a bigraph before anonymizing process. However, this method can only be applied in some special graphs. If the modification during the above changing process is too much, it will greatly reduce the utility of original graph. Thus, this method cannot be applied into all kinds of social network data sets.

Briefly, existing anonymizations usually have the shortcomings of poor anti-attack capability and low utility, which mainly contains the unavailability and high extraction cost of the part of original data without privacy information. Thus, in this paper, we propose a novel anonymization, Splitting Anonymization, which is different of the above four existing methods. Our new method takes the advantages of the above four methods but makes some improvement. Firstly, to the information attached in the vertices in original graph, it conducts some process of trivialization (removing identity information), classification and integration, which ensure a high level protection to passive attack. Then, based on the splitting process to the vertices in original graph, it makes the information on vertices copy scrambled and the information on edges splitting stored in original graph through a serious of conversion. This phase ensures the original graph and its substructure cannot be traced in anonymous graph, which provides a high level protection to active attack. Noted that our method hid all privacy information, and can return a result with high accuracy for utility query.

To summarize, our contributions are listed as follows.

1. We present a novel anonymization approach.
2. We propose the concept of "Knowledge Information", and apply it into our anonymizing model, which reduce the constraint to unnecessary protected information and anonymization. Moreover, we propose the new concepts of "predictable error" and "approximate constraint error", which provides a more accurate query result.
3. The splitting anonymization can provide reliable privacy protection to all the known passive attacks and active attacks, and enhance the speed and accuracy of utility query.
4. Strict theoretical analysis proves that our splitting anonymization can satisfy the requirement of privacy protection and has a high utility. Large amount of experiment also verifies our design.

The rest of this paper is organized as follows. In Section 2, we introduce some basic concepts and give out the problem definition. In Section 3, we describe our principles and phases of splitting anonymizing in detail. In Section 4, we discuss the techniques of dynamic anonymization with the update of social network. The theoretical analysis for the protection effect of splitting anonymizing when facing different kinds of attacks is elaborated in Section 5, followed by the utility analysis in Section 6. Section 7 and 8 show the experiment results and the related work, respectively. Finally we conclude our paper in Section 9.

## 2 Problem Definition

In this section, we first give out some basic definitions on privacy, and then propose our problem.

**Definition 2.1. Social Network.** Data can be presented by a graph $G = (V, E, U)$. $V = \{v_1, v_2, ..., v_n\}$ is the vertex set presenting all the vertices in the graph corresponding to participators in the social network. $E \subseteq \{(v_i, v_j) \mid 1 \leq i, j \leq n, i \neq j, v_i, v_j \in V\}$ presents all the edges in the graph, which means the relationship between the participators. $U = \{u_1, u_2, ..., u_n\}$ is the set of vertex labels, and $u_i$ presents the label attached to vertex $v_i$ for $1 \leq i \leq n$. We use $u_i^t$ to present the value of $t$ item of label $u_i$.

**Definition 2.2.** The third party that tries to access the sensitive data without authority is called an **Attacker**. The method that an attacker only exploits the information and leak in anonymizing data to get the sensitive data is called **Passive Attack**. The method that an attacker needs to use other information to get the sensitive data except for the anonymizing data is called **Active Attack**.
As shown in Figure 1, two vertices of class \( a \) and one vertex on class \( b \) are grouped into one set. Although it is only known that vertex \( c \) is related to two vertices in this set, the attacker can still infer that \( c \) is related to at least one class \( a \) vertex. Through this way, the privacy information of vertex \( c \) is divulged.

**Definition 2.3.** An attacker utilizing the openness of social network, generate a substructure that is easy to be recognized before the publication of anonymizing data as background knowledge. This substructure is usually a special vertex or subgraph. After the anonymizing data is published, the attacker can acquire sensitive data through identifying this structure. The above method is called **Structured Attack**.

In Figure 2, the generated substructure that the attacker published before anonymzing data is shown as yellow vertices, which is a subgraph here. After the publication of anonymizing data, the attacker can identify another three target vertices in blue by identifying this substructure, and get the sensitive data of their relation.

**Definition 2.4. Anonymization** can be presented by a function \( f: G \rightarrow G_H \), where \( G \) is the original social network data and \( G_H = \{V_H, E_H, U_H\} \) is the data set after anonymizing.

The function \( f \) of anonymization often has the following three characters:

1. \( G_H \) should not contain the direct sensitive information in \( G \).
2. \( G_H \) should take the effect of privacy protection aiming at different kinds of attacks.
3. \( G_H \) should remain the useful information in \( G \) as much as possible under the premise of (1) and (2).

**Definition 2.5. Predictable Error** is the query error acquired through theoretic computation before the query result is accessed.

**Definition 2.6. (Problem Definition)** An anonymization \( f_S: G \rightarrow G_H \) needs to be work out so that \( G_H \) satisfies three constrains as follows. (1) For the queries that directly ask for privacy information, it can return unavailable or useless results. (2) The stricter the constraint is, the more ambiguous the query result is in order to avoid leakage of privacy. However, the error should be constrained in a predictive range. (3) For the queries with relaxed constrains, there should exist a query process that can return exact or approximate results, and also, the error should be constrained in a predictive range.

3 The Splitting Anonymization Algorithm

In this section, we describe our proposed algorithm, the Splitting Anonymization Algorithm in detail.

The Splitting Anonymization Algorithms can be divided into two parts, **vertex information anonymization** and **vertex splitting**, which is shown in Figure 3. The Vertex Information Anonymization contains three phases, **label trivialization** (which is shown in Definition 3.1 and Algorithm 1), **grouping trivialized labels** (which is shown in Definition 3.2 and Algorithm 2), and **exact grouping** (which is shown in Definition 3.3 and Algorithm 3). The vertex splitting part is realized by graph **strict m-order algorithm** (which is shown in Definition 3.4 and Algorithm 4).

The main purpose of vertex information anonymization is to unified group different vertex information within the range of permitted accuracy, which is presented in tolerance set \( \Delta \) and shown in Definition 3.2. Not only can it remain main information in vertex labels, but also avoid the leakage of privacy caused by some special labels.

Two main purposes of vertex splitting are as follows. Firstly, vertex information in original graph is scrambled by copies, which makes one vertex in the original graph presented by many vertices in the anonymizing graph. This method enhances the difficulty of attacker’s identification, and thus improves the privacy protection. Secondly, the edge information in the original graph is stored separately in the anonymizing graph. That is, split the useful information that may expose privacy before hiding them into anonymizing graph, which makes attackers cannot recover privacy.
information but users can quickly recombine required information in normal service. In this way, we can both protect the privacy and improve the utility of anonymizing graph.

**Definition 3.1** Label trivialization it to get $U_R = R(U)$ through a function $R$ from $U$, where $R(U) = \{u_{R1}, u_{R2}, ..., u_{Rn}\}$ is the trivialization set, and $u_{Ri}$ is $u_i$ without main identification information.

The main purpose of label trivialization is to remove the only identification information of the vertices in original graph, which makes attackers cannot identify the vertices. For example, if $u_i = \{id = 98792, name = Lucy, gender = f, age = 13\}$, and the main identification information is id and name, then $u_{Ri} = \{gender = f, age = 13\}$, where $u_i^{gender} = u_{Ri}^{gender} = f, u_i^{age} = u_{Ri}^{age} = 13$. The algorithm is shown in Algorithm 1.

**Definition 3.2** Grouping trivialized label is to get $U_G = G(U_R, \Delta)$ through a function $G$ from $U_R$. Here, $U_G = \{u_{G1}, u_{G2}, ..., u_{Gm}\}$ is the label set without main identification information after grouping with $1 \leq m \leq n$, and $\Delta = \{\delta_1, \delta_2, ..., \delta_r\}$ is the tolerance set of numeric label where $r$ is the number of numeric label in $u_{Gi}$. $\forall u_{Ri} \in U_R, \exists u_{gp} \in U_G$ satisfies that for any non-numerical label $t_c$, there exists $u_{Ri}^{t_c} = u_{gp}^{t_c}$, and for the $s^{th}$ numerical label, there exists $|u_{Ri}^{t_s} - u_{gp}^{t_s}| < \delta_s 1 \leq s \leq r$. If for some $u_{R0}, \exists u_{gp} \in U_G$ and satisfies the above conditions at the same time, then $p = q$, that is, $u_{gp} = u_{Gq}$. $u_{R0}$ is called trivialized labels of $u_{R0}$ after grouping.

The grouping trivialized labels algorithm is shown in Algorithm 2. Noted that the second line in this algorithm sort the elements in $U_R$, which avoids the trouble of query the corresponding group.

**Definition 3.3** Exact grouping the social network data is to change $G_G = \{V, E, U_G\}$ from $G$ according to the numeric label tolerance set $\Delta$. $U_G' = \{u_{G1}', u_{G2}', ..., u_{Gn}\}$, where $u_i$ is trivialized labels of $u_{Gi}$. $1 \leq i \leq n$ after grouping.

The main purpose of exact grouping is to unitize and normalize the label information of vertices within the tolerance range, which both remains the difference of original information and reduce the probability of being attacked. The detailed algorithm is shown in Algorithm 3.

**Table 1** Original Data of Instance for Exact Grouping

<table>
<thead>
<tr>
<th>$v$</th>
<th>$u$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>1 Alice f 22 UK</td>
</tr>
<tr>
<td>$v_2$</td>
<td>2 Bob m 27 UK</td>
</tr>
<tr>
<td>$v_3$</td>
<td>3 Charles m 23 USA</td>
</tr>
<tr>
<td>$v_4$</td>
<td>4 David m 25 USA</td>
</tr>
<tr>
<td>$v_5$</td>
<td>5 Eve f 26 USA</td>
</tr>
<tr>
<td>$v_6$</td>
<td>6 Francis f 29 USA</td>
</tr>
<tr>
<td>$v_7$</td>
<td>7 Gerald m 27 UK</td>
</tr>
<tr>
<td>$v_8$</td>
<td>8 Henna f 22 UK</td>
</tr>
<tr>
<td>$v_9$</td>
<td>9 Ike f 23 UK</td>
</tr>
<tr>
<td>$v_{10}$</td>
<td>10 Jack m 21 USA</td>
</tr>
</tbody>
</table>

**Table 2** Trivialization Labels

<table>
<thead>
<tr>
<th>Labels</th>
<th>Classes for Short</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>(20,25) UK A</td>
</tr>
<tr>
<td>f</td>
<td>(25,30) USA B</td>
</tr>
<tr>
<td>m</td>
<td>(20,25) USA C</td>
</tr>
<tr>
<td>m</td>
<td>(25,30) UK D</td>
</tr>
</tbody>
</table>

**Table 3** The Result of Exact Grouping

<table>
<thead>
<tr>
<th>$v$</th>
<th>Classes for Short</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>A</td>
</tr>
<tr>
<td>$v_2$</td>
<td>D</td>
</tr>
<tr>
<td>$v_3$</td>
<td>C</td>
</tr>
<tr>
<td>$v_4$</td>
<td>C</td>
</tr>
<tr>
<td>$v_5$</td>
<td>B</td>
</tr>
</tbody>
</table>

**Algorithm 1** Label Trivialization

Input: label set $U$, main identification information set $I$
Output: Trivialization label set $U_R$

$U_R \leftarrow \emptyset$;

**Algorithm 2** Grouping Trivialized Labels

Input: trivialization label set $U_R$, tolerance set $\Delta$
Output: grouping label set $U_G$

$U_G \leftarrow \emptyset$;

**Algorithm 3** Exact Grouping

Input: label set $U$, tolerance set $\Delta$, grouping label set $U_G$
Output: exact grouping label set $U'_G$ from label set $U$
foreach $u_i$ in $U_R$
  $u_{R_i} \leftarrow \emptyset$;
  foreach $u_i$ in $u_i$
    if ($t \notin l$)
      $u_{R_i} \leftarrow u_i$;
    $U_R \leftarrow u_{R_i}$;

sort($U_R$);
foreach $u_i$ in $u_i$
  $u_{R_i} \leftarrow$ NULL;
foreach $u_i$ in $U_R$
  if (the corresponding group of $u_i$ is not $u_{R_i}$ according to $\Delta$)
    $u_{R_i} \leftarrow$ the corresponding group of $u_i$ according to $\Delta$;
  $U_R \leftarrow u_{R_i}$;

After exact grouping, the original graph $G$ is changed into $G_c = \{V, E, U'_c\}$. The exact grouping destroys the privacy information which groups the remained labels into several classes according to their similarity. Each vertex in $G_c$ can be grouped into one class, where $G_c$ only presents the relationship between the vertices in the same group and the vertices in different groups. For example, Table 1 shows an original vertex data in social network before exact grouping.

Table 2 and Table 3 together show the result of exact grouping, where Table 2 presents all labels of $U_c$ and corresponding classes for short after grouping label trivialization, and Table 3 presents the results of exact grouping based on Table 2. From Table 3, we can see that the original 10 vertices are grouped into 4 classes.

**Definition 3.4** The strict $m$-order splitting on a vertex is to use $m$ subvertices $v_1, v_2, ..., v_m$ to present vertex $v$ such that

1. all vertices have the same label with the original vertex $v$;
2. $\forall (v, v_\alpha) \in E$, there is only one corresponding $(v_i, v_\alpha)$, where $1 \leq i \leq m$.

If and only if any label $u_i$ in group $A$ of $U'_c$ satisfies $|v_i|_A - |v_j|_A| \leq 1$, we call it is strict for $m$-order splitting on vertices, where $|v_i|_A$ presents $\{((v_i, v_\alpha)|u_{v_\alpha} = u_i, (v_i, v_\alpha) \in E\}$.

The strict $m$-order splitting on Graph $G_c$ after exact grouping is the process to get $G_S = \{V_S, E_S, U_S\}$ through making strict $m$-order splitting on all the vertices in graph $G_c$, where $V_S$ is the subvertex set after splitting, $E_S$ is the edge set after splitting, and $U_S$ is the label set after splitting.

**Algorithm 4** Strict $m$-Order Splitting

Input: Exact grouping graph $G_c = \{V, E, U'_c\}$, splitting coefficient $m$
Output: Split graph $G_S = \{V_S, E_S, U_S\}$

foreach $v_i$ in $V$
  for ($j = 0$ to $m - 1$)
    $V_S \leftarrow V_E \leftarrow E; U_S \leftarrow \emptyset$;
    foreach $v_\alpha$ in $U'_c$
      $V_S = \emptyset$;
      $U_S \leftarrow u_{v_\alpha}$;
      foreach $u_i$ in $U'_c$
        $j \leftarrow 0$;
        while ($3(v_i, v_\alpha) \in E$)
          $E_S \leftarrow \{v_i \mid v_i \not\in E\}$;
          $j \leftarrow j + 1$;
          $E_S \leftarrow v_i$;
        $V_S \leftarrow \emptyset(v_i)$;

Figure 4 shows a simple instance of strict 2-order splitting. Assume that in original graph (a), there exists a class $A$, that is $U'_{c_1} = U'_{c_2} = U'_{c_3} = U'_{c_4} = u_A$. In (b), vertex $v_1$ in original graph is changed into two subvertices $u_{v_1}$ and $v_1$ by 2-order splitting. $(v_1, v_2)$ in original graph is replaced by $(v_{i_0}, v_2), (v_{i_1}, v_3)$ is replaced by $(v_1, v_4)$, and $(v_1, v_3)$ is replaced by $(v_5, v_6)$.
replaced by \((v_1, v_4)\). So, \(|v_1|_A = 1\) and \(|v_1|_A = 2\) satisfies \(||v_1|_A - |v_1|_A| = |1 - 2| = 1\). Similarly, the following (c),(d) and (e) describe the strict 2-order splitting process on vertices \(v_2, v_3, v_4\) respectively in original graph. Noted that different orders of vertex splitting results in different results. Thus we should consider the optimal order. Based on the splitting algorithm, we can prove that if we select the minimum number of edges to order all vertices, we can use the least costs to obtain a splitting result, which is just the optimal edge cover problem [25]. Thus we can use the corresponding greedy algorithm to compute the optimal order [25].

4 Dynamic Anonymization With the Update of Social Networks

When the original social network data is updated, our algorithm should be able to do corresponding adjustment to change the original anonymized data into the new anonymized ones. This process is called Dynamic Anonymization. As illustrated in Section 3, during the process of our Splitting Anonymization Algorithm, instead of processing the whole graph, what we need is only to process the vertices with some characteristics and their neighbor vertices, except for the exact grouping step, in which all vertices in the original graph are needed to determine the group label of each vertex. Thus, the Splitting Anonymization Algorithm has the intrinsic characteristic of dynamic anonymization with the update of social networks.

Dynamic Anonymization is a very important characteristic in our algorithm. The anonymization process in the existing anonymization algorithms usually needs to take a view of the whole graph before anonymization so as to ensure the security of privacy. Even worse, some anonymization algorithms have “butterfly effects”, which means that a small change of the original graph may lead to a privacy leakage when attacked. In addition, some anonymization algorithms apply a greedy method to solve the anonymization problem, which is usually NP-hard. To ensure the privacy, once some parts of the original graph change, the whole algorithms may need a recomputation. Thus, most anonymization algorithms do not have the dynamic anonymization characteristic due to its high computation complexity.

The basic operations of the update of social network data are vertex insertion, edge insertion, edge deletion, and vertex deletion. In follows, we will illustrate our Dynamic Anonymization algorithms according to the four basic operations.

4.1 Vertex Insertion

The anonymization process is very simple when inserting a new vertex to the original graph, which is shown in Algorithm 5. Firstly, we need to conduct the exact grouping process on the new added vertex, and replace its original label with the new group label. There are no edges connecting this vertex at present, so what we need to do is only to split this vertex. The new split graph \(G'_S\) is the result.

<table>
<thead>
<tr>
<th>Algo. 5 Dynamic Anonymization for Vertex Insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: the split graph (G_S = {V_S, E_S, U_S}),</td>
</tr>
<tr>
<td>New vertex (v), new vertex label (u)</td>
</tr>
<tr>
<td>Output: the new split graph (G'_S = {V'_S, E'_S, U'_S})</td>
</tr>
<tr>
<td>(u \leftarrow) the exact grouping label of (v);</td>
</tr>
<tr>
<td>(for(i = 1:m))</td>
</tr>
<tr>
<td>(V'_S, add(v);)</td>
</tr>
<tr>
<td>(U'_S, add(u);)</td>
</tr>
</tbody>
</table>

As shown in Fig.5, insert the vertex \(v_5\), and the original graph (a) becomes a new graph (b). (a’) and (b’) are the strict 2-order split graphs of (a) and (b) respectively. We can see that only two vertices, \(v_5\) and \(v_5\), are added to (b’) during the process from (a’) to (b’). Thus, the computation complexity is \(O(1)\).

4.2 Edge Insertion

The anonymization for edge insertion is a litter more complex than vertex insertion. As the number of edges of the
anonymization data is the same with that of the original data, for the final anonymization data, it should also be added with one edge. But the problem is, attributively adding one edge to its corresponding sub-vertex may lead that the absolute difference between the degree of sub-vertices produced by splitting of one vertex and the degree of some groups of vertices becomes to 2. This does not meet the requirement of the definition of strict splitting anonymization. So when inserting a new edge into the original graph, we should consider more than more vertices to decide the place for the inserted edge. The detailed algorithm is shown in Algorithm 6.

From Algorithm 6, we can see that the decision process is not difficult. To solve the problem of edge insertion, we only need to connect the sub-vertices whose degree is the smallest with respect to the degree of some group of vertices. The complexity of the process of finding the smallest degree is linear, which is $O(n)$. Note that it is not necessary to traverse all the vertices to find out $v_{min}$. Because the original graph satisfies the requirement of strict splitting, once we find two different degrees of vertices, the difference between them must be 1, and the smaller one between them must be the smallest one in the graph.

---

<table>
<thead>
<tr>
<th>Algo. 6 Dynamic Anonymization for Edge Insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: the split graph $G_s = {V_s, E_s, U_s}$, the new edge $(v_i, v_j)$</td>
</tr>
<tr>
<td>Output: the new split graph $G'_s = {V'_s, E'_s, U'_s}$</td>
</tr>
</tbody>
</table>

According to the label of the graph that $v_j$ is in, find the vertex $v_{min}$ whose degree is the smallest among the neighbor vertices of $v_i$; According to the label of the graph that $v_i$ is in, find the vertex $v_{min}'$, whose degree is the smallest among the neighbor vertices of $v_j$;

$E_s' \cup \{(v_{min}, v_{min}')\}$;

As shown in Fig.5, from (b) to (c), we add the edge $(v_2, v_3)$, and in the corresponding process of the anonymization graph, from (b') to (c'), we choose $v_{2_0}$ and $v_{5_0}$, and connect them with edge $(v_{2_0}, v_{5_0})$. Before the edge insertion, the degrees of $v_{2_0}$ and $v_{2_1}$ are both 1, and the degrees of $v_{5_0}$ and $v_{5_1}$ are both 0. So we can attributively select vertices between them and insert a new edge. However, from (c) to (d), the inserted edge is $(v_1, v_5)$, and in the corresponding anonymization graph, we choose $v_{1_0}$ and $v_{5_1}$ and add edge $(v_{1_0}, v_{5_1})$. Before inserting the new edge, the degrees of $v_{1_0}$ and $v_{1_1}$ are 1 and 2 respectively, and $v_{5_0}$ is smaller. Correspondingly, the degrees of $v_{5_0}$ and $v_{5_1}$ are 1 and 0 respectively, and $v_{5_1}$ is smaller. Thus, we can only choose $(v_{1_0}, v_{5_1})$ as the new edge.
4.3 Edge Deletion

The process of edge deletion in the original graph is much more complex than edge insertion. As the number of edges of the anonymization data is the same with that of the original data, for the final anonymization data, it should also delete one edge. But the problem is that deleting the corresponding edge in the anonymization graph may lead that the degree difference between two sub-vertices after splitting and some group of vertices becomes to 2, which does not meet the requirement of the strict splitting anonymization definition. Thus, we need to process these vertices after deleting the corresponding edge in the anonymization graph. That is, we should spare the additional edges to the vertices whose edges are fewer. The detailed algorithm is shown in Algorithm 7.

In this algorithm, when deleting the edge \((v_i, v_j)\) in the original graph, we should first delete the corresponding edge \((v_{i'}, v_{j'})\) in the anonymization graph. Then, we compare the sub-vertices that are influenced by \(v_i\) and \(v_j\), and determine whether they are contrary to the strict splitting definition. If so, there only exist the following two situations,

\[|v_{i_{\max}}| - |v_{i_{\min}}| = 2\]

or

\[|v_{j_{\max}}| - |v_{j_{\min}}| = 2\]

Once the above problem happens, we can solve it by sparing one edge of the vertex whose degree is the largest to the vertex whose degree is the smallest. Although Algorithm 7 is more complex than Algorithm 6, it only involves the vertex whose degree is the largest and the vertex whose degree is the smallest. Thus, the complexity of Algorithm is also \(O(n)\).

<table>
<thead>
<tr>
<th>Algo. 7 Dynamic Anonymization for Edge Deletion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: the split graph (G_S = {V_S, E_S, U_S}),</td>
</tr>
<tr>
<td>the deleted edge ((v_i, v_j))</td>
</tr>
<tr>
<td>Output: the new split graph (G'_S = {V'_S, E'_S, U'_S})</td>
</tr>
<tr>
<td>Find the corresponding edge ((v_{i'}, v_{j'})) in the</td>
</tr>
<tr>
<td>anonymization graph of edge ((v_i, v_j));</td>
</tr>
<tr>
<td>(E_S.remove((v_{i'}, v_{j'})));</td>
</tr>
<tr>
<td>According to the label of the graph that (v_j) is in, find</td>
</tr>
<tr>
<td>the vertex (v_{i_{\min}}) whose degree is the smallest and</td>
</tr>
<tr>
<td>the vertex (v_{i_{\max}}) whose degree is the largest among</td>
</tr>
<tr>
<td>the neighbor vertices of (v_i);</td>
</tr>
<tr>
<td>According to the label of the graph that (v_i) is in, find</td>
</tr>
<tr>
<td>the vertex (v_{j_{\min}}) whose degree is the smallest and</td>
</tr>
<tr>
<td>the vertex (v_{j_{\max}}): whose degree is the largest among</td>
</tr>
<tr>
<td>the neighbor vertices of (v_j);</td>
</tr>
<tr>
<td>if ((</td>
</tr>
<tr>
<td>Find a neighbor vertex (v_{i_{\max}}) of (v_{i_{\max}}) that</td>
</tr>
<tr>
<td>has (v_j)'s group label;</td>
</tr>
<tr>
<td>(E_S.remove((v_{i_{\max}}, v_{i_{\min}})));</td>
</tr>
<tr>
<td>(E_S.add((v_{i_{\min}}, v_{i_{\max}})));</td>
</tr>
<tr>
<td>if ((</td>
</tr>
<tr>
<td>Find a neighbor vertex (v_{j_{\max}}) of (v_{j_{\max}}) that</td>
</tr>
</tbody>
</table>
has $v'_i$'s group label;

\[ E_S.\text{remove}\left([v_{i_{\max}}, v_{i_0}]\right); \]
\[ E_S.\text{add}\left([v_{i_{\min}}, v_{i_0}]\right); \]

(a) \hspace{1cm} (b) \hspace{1cm} (c) \hspace{1cm} (d)

Fig. 6 Dynamic Splitting Example of Vertex and Edge Deletion

For example, from Fig. 6 (a) to (b), we delete edge $(v_1, v_4)$ from the original graph. Correspondingly, in the anonymization graph (a') and (b') of (a) and (b) respectively, we only need to delete $(v_{1_0}, v_{4_1})$. This is because after deletion, the degrees of $v_{1_0}$ and $v_{4_1}$ are changed into 1 and 2 respectively, which meets the requirement of strict splitting anonymization. Similarly, the degrees of $v_{4_0}$ and $v_{4_1}$ are changed into 1 and 0 respectively, which also meets the requirement of strict splitting anonymization. However, from Fig. 6 (b) to (c), we delete edge $(v_2, v_3)$ from the original graph. Correspondingly, in the anonymization graph (b') and (c'), we not only delete the edge $(v_{2_1}, v_{3_1})$, but also replace the original edge $(v_{2_0}, v_{3_0})$ with edge $(v_{2_1}, v_{3_0})$. This is because the degrees of $v_{2_0}$ and $v_{2_1}$ are changed into 2 and 0 respectively, which is contrary to the definition of strict splitting anonymization. Additionally, the degrees of $v_{3_0}$ and $v_{3_1}$ are changed into 1 and 1 respectively, which meets the requirement of strict splitting anonymization definition, and there is no need to do any more processes.

4.4 Vertex Deletion

Algorithm 8 shows the anonymization algorithm of vertex deletion.

---

Algorithm 8 Dynamic Anonymity for Vertex Deletion

**Input:** the split graph $G_S = \{V_S, E_S, U_S\}$, the deleted vertex $v$

**Output:** the new split graph $G'_S = \{V'_S, E'_S, U'_S\}$

Find all edges connected to $v$, and delete them according to Algorithm 7.

Delete all the sub-vertices of $v$ in the anonymization graph.
In this algorithm, we first delete all the edges connected to the vertex \( v \) using Algorithm 7. Then we delete vertex \( v \) and all its corresponding sub-vertices in the anonymization graph. This process needs to traverse all the edges connecting to \( v \), and then delete these edges, so the computation complexity is \( O(n^2) \). For the process that deletes all the sub-vertices in the anonymization graph, because its order is fixed, the complexity is \( O(1) \), which can be ignored.

For example, from Fig.6 (c) to (d), in the original graph, we delete vertex \( v_3 \), and there are two edges \( (v_1, v_3) \) and \( (v_3, v_4) \) connected to it. The corresponding anonymization graphs, from (c') to (d'), we need to delete \( (v_1, v_3) \) and \( (v_3, v_4) \), and no further processes are needed to meet the requirement of strict splitting anonymization definition. After deleting these edges, we can delete the two sub-vertices of \( v_3 \), \( v_{30} \) and \( v_{31} \).

5 Analysis of Security

In this section, we analyze the security of our algorithm aiming at three normal attacks, “passive attack”, “single vertex neighbors attack”, and “subgraph attack”.

Definition 4.1 Knowledge Information is the published information or the information that can be accessed through statistical approach in anonymizing data.

The security of anonymization algorithms is an important standard to evaluate an algorithm. In this paper, we analyze the security of our algorithm based on three premises as follows.

1. Knowledge information is not privacy information, that is, knowledge information accessed by attackers is not privacy leakage.
2. The process that statics privacy information inferred directly or indirectly by attackers only based on knowledge information is not privacy leakage.
3. Attackers can only recognize the vertices they have known, such as knowledge information, but cannot access any privacy information of other vertices inferred from the known vertices. This process is not privacy leakage.

For example, in anonymization data, attackers have known that 95% of people are from China, and only 5% of people are from other countries. The above information is knowledge information. If this is known by attacker, it is not privacy leakage. Given another example, based on the above information, attackers can infer that the probability that a user is from China is 95%. Though it is an estimate over privacy information, it is not privacy leakage. If attackers access the literacy rate of the Chinese is 72% from government statistic data in 2010, furthermore infer that the probability which this user is literate is 95% \( \times \) 72% = 68.4%. This process is also not privacy leakage.

After exact grouping and strict splitting, the original social network graph \( G \) is changed into anonymization graph \( G_S \), which can be directly published. We analyze the anti-attack ability of \( G_S \) as follows.

5.1 Analysis for the Resistance of Passive Attack

Theorem 5.1 There is no privacy leakage in anonymization graph acquired from exact grouping when it encounters passive attack.

Proof. After exact grouping, there is no main identification information in vertex labels, so it is impossible for attackers to identify any actual entity presented by corresponding vertex. What they can access is only the relation information of intra-group or inter-group. Moreover, the relation information of intra-group or inter-group is statistic data. Thus, Theorem 5.1 is proved. □

5.2 Analysis of the Resistance of Single Vertex Neighbors Attack

The single vertex neighbors attack is an attack method that attackers only have some knowledge information of a single vertex, and after identifying this vertex in anonymization data, perform privacy attack to its neighbors. As social network data is a kind of big data, different vertices have complex relationship and variety types. Thus, it is very difficult to construct a vertex with completely the same summation of information on this vertex and its corresponding edges after exact grouping if the global information is not known. The generated vertex and the copy of other vertices by exact splitting make so much trouble for identifying this knowledge vertex in anonymization graph.

Theorem 5.2 If there are totally \( n \) vertex labels and the edge information (containing the knowledge vertex) is the same
with the knowledge vertex, the probability that attackers can completed identify this knowledge vertex in anonymization graph is no more than $1/C_{nm}^m$.

**Proof.** If there are totally $n$ vertex labels and edge information is the same with the knowledge vertex, there will be at least $mn$ vertex labels and edge information can become the subvertices of knowledge vertex. As there is no other auxiliary information, there should be $C_{nm}^m$ construction methods according to combination principle. Thus, the probability of completely identification of knowledge vertex is $1/C_{nm}^m$. Moreover, after exact grouping, there may exist such vertices that vertex label information is the same with knowledge vertex but edge information is different, and thus the subvertices of these vertices may also become the subvertices of knowledge vertex. Thus, the probability of completely identification of knowledge vertex is no more than $1/C_{nm}^m$.

**Theorem 5.3** In the strict $m$-order split anonymization graph, after completely identify one knowledge vertex, attackers can get at most $1/m$ its neighbor vertex information.

**Proof.** After strict $m$-order splitting, for the sub vertices of one pair neighbor vertices in original graph, there exist one and only one pair o if sub-vertices are still neighbors. Thus, after completely identify one knowledge vertex, attackers can get at most $1/m$ its neighbor vertex information.

**Theorem 5.4** The anonymization graph can provide $C_{nm}^m$-order protection to single vertex neighbors attack after strict $m$-order splitting.

**Proof.** The proof can be seen from Theorem 4.2 together with [10].

### 5.3 Analysis of the Resistance of Subgraph Attack

The definition of subgraph attack can reference from Definition 2.3 and Figure 2.

**Theorem 5.5** A substructure with $n$ vertices and $d$ average degree, there exists

$$\left(\prod_{i=0}^{m-1} C_{d-i|m}^{d} \right)^n$$

splitting results after strict $m$-order splitting.

**Proof.** Split a vertex $v_i$ with $d$ through strict $m$-order splitting into $v_{i1}, v_{i2}, \ldots, v_{im}$. These $m$ subvertices satisfy $|v_i| = [d/m]$ or $|v_i| = [d/m]|i = 1, 2, \ldots, m$. We use $|v_i| = [d/m]$ for convenience in the following. Allocate the edges according to their corner mark from small to large, and there exist

$$C_{d-(i-1)|m}^{d}$$

kinds of allocation methods. Thus, for each vertex, there exist

$$\left(\prod_{i=0}^{m-1} C_{d-i|m}^{d} \right)$$

kinds of splitting methods. Thus, for substructures, there exist

$$\left(\prod_{i=0}^{m-1} C_{d-i|m}^{d} \right)^n$$

kinds of splitting results.

In general social network data, there should usually be about 30 vertices in an effective substructure. However, in social network data, the average degree of each vertex is probably from 100 to 1000. Table 5 shows the splitting result when the number of vertices is 10 in substructure. Moreover, for small query graph, it is almost impossible to do so much subgraph query over such a large social network data under the existing computation ability, even when the query substructure is as large as 30 vertices and more than 100 average degrees. Thus, under the condition of existing computation ability, it is impossible to perform subgraph attack on anonymization graph after strict $m$-order splitting.
6 Utility Analysis

**Definition 6.1 (Utility)** Clustering queries can be grouped into two classes. One is only perform clustering queries on the existing constraints of vertices, without considering edges, which is called **First Class Query (FCQ)**. The other one considers edges, but there exist and only exist numeric constraints on edges, which is called **Second Class Query (SCQ)**. It usually uses SCQ to enumerate utility.

For example, the queries such as “how many users are of the age between 20 and 25 in social network” and “how many users are Chinese men or Japanese men in social network” belong to FCQ. The queries such as “how many Americans have Chinese friends over 20 years old in the social network” and “how many females over 20 years old have no male friends” belong to SCQ.

**Theorem 6.1** Anonymization data after strict m-order splitting can answer FCQ without error in the tolerance difference \( \Delta \).

As FCQ only refers to non-privacy information on vertices, and non-privacy information on vertices in anonymization data after strict m-order splitting does not change within the tolerance difference \( \Delta \), it can directly return the number of vertices \( n \) satisfying query constraints. Furthermore, the query result is \( \frac{n}{m} \).

**Theorem 6.2** Anonymization data after strict m-order splitting can answer SCQ accurately with predictive error in the tolerance difference \( \Delta \).

We divide SCQ over numeric constraints on edges into 3 classes, exact constraint, bilateral constraint, and unilateral constraint. Exact constraint is given an exact number of edges. For instance, the number of edges is 10 or 5. Bilateral constraint is given upper bound and lower bound of edges at the same time. For instance, the number of edges is between 10 and 20. Unilateral constraint is given only upper bound or lower bound of edges. For instance, the number of edges is less than 10 or more than 5. The proof of Theorem 6.2 will be processed in 3 parts separately according to the above 3 classes of SCQ.

**Proof.** In this paper, we only show the proof of strict 2-order splitting of the query that vertices in Class A satisfies the constraints on edge numbers of vertices in Class B. Note that \( |A^p_B| = |\{v \in A, |v|_B = n\}| \), that is, the number of vertices with \( n \) edges in Class A and Class B.

1. When the number of edges is even number, \( p \), in exact constraint:

   Find out \( n \) vertices in Class A satisfies that the edge number of vertices in Class B is \( \frac{p}{2} \), and note the query result as \( \frac{n}{2} \). It can be known that,

   \[
   \frac{n}{2} = \frac{|A^{p-1}_B|}{2} + \frac{|A^{p+1}_B|}{2} + |A^p_B|
   \]

   where \( |A^p_B| \) is the true value of result, and \( \frac{|A^{p-1}_B|}{2} + \frac{|A^{p+1}_B|}{2} \) is the error of result. As the error is only respect to \( |A^{p-1}_B| \) and \( |A^{p+1}_B| \), and can be prospected before query, it is called prospective error.

2. When the number of edges is between even number \( p \) and \( q \) in bilateral constraint:

   Find out \( n \) vertices in Class A satisfies that the edge number of vertices in Class B is between \( \frac{p}{2} \) and \( \frac{q}{2} \), and note the query result as \( \frac{n}{2} \). It can be known that,

   \[
   \frac{n}{2} = \frac{|A^{p-1}_B|}{2} + \frac{|A^{p+1}_B|}{2} + \sum_{i=p}^{q} |A^i_B|
   \]

   The error analysis is similar to that of (1).

3. When the number of edges is more than even number, \( p \), in unilateral constraint:
Find out \( n \) vertices in Class A satisfies that the edge number of vertices in Class B is more than \( \frac{p}{2} \), and note the query result as \( \frac{n}{2} \). It can be known that,

\[
\frac{n}{2} = \frac{|A^{p-1}_B|}{2} + \sum_{i=p}^{\infty} |A^i_B|
\]

The error analysis is similar to that of (1). □

Noted that exact constrain may lead to privacy leakage when the number of results satisfying constraints is small. However, it does not need exact constraint query in social network data analysis. Thus, predictive error is larger compared with query result. On the other hand, the error approximately satisfies the constraints, and actually, the query result should be the number of vertices with about \( p \) edges. The bilateral constraint and unilateral constraint are similar to exact constraint. The percentage of predictive error becomes smaller with the increase of the number of query results, and satisfies the constraints all along. Moreover, in exact constraint query, we can make an assumption that \( |A^{p-1}_B| \approx |A^p_B| \approx |A^{p+1}_B| \approx |A_B| \), and use \( \frac{n}{4} \) to correct result, which can make query more accurate. Similar are the correct methods of other kinds of queries.

### 6.1 The Correction of Usability Query

The predictable errors, as mentioned above, makes the users be able to predict the reason why these errors happen, and the place where these errors happen, and even predict the proportion of the errors according to some distribution function. Thus, unlike the existing methods that provide the query answers only according to the returned query results, we can provide the users a corrected result based on the analysis of the characteristics of the anonymization data or the original data. With this method, we can further improve the accuracy of the queries.

Because the query results of the first class usability query provided by the splitting anonymization algorithm are exact results within the error range of \( \Delta \), the accuracy rate is 100%. This is proved by former theorems, so there is no need for any result correction. In the following, we will discuss how to correct the predictable errors produced by the splitting anonymization data in the second class of queries.

There are so many methods of result correction of the first class of usability queries, but the final effect is not the same. In general, there are three phases to correct the predictable errors, which are illustrated as follows.

1. Analyze the distribution of query variables in the original data or anonymization data;
2. Compute the place where the part matching the query conditions is;
3. Compute the proportion of the error terms in the query and correct them.

For example, in a social network data, there are two groups of vertices, namely A and B. The number of edges connecting the vertices between A and B (one vertex of such edge is in A and the other is in B) is from 5 to 20. Here, the group A and B are the group labels after the exact grouping step. First, we need to analyze the relationship between the distributions of the original data and the split data.
Fig. 7 is the comparison distribution of vertices and edges between the distribution of an original data and its anonymization data after strict 2-order splitting when the queries are as mentioned above. Here, the dashed line shows the relationship between vertices and edges in the original data, where the vertical axis on the right side shows the number of vertices, and the horizontal axis on the top shows the number of edges. The solid line shows the relationship between vertices and edges in the split anonymization data, where the vertical axis on the left side shows the number of vertices, and the horizontal axis on the bottom shows the number of edges. For example, in the original data, the point (26,162) means in the original data, there are 162 vertices in Group A that connect 26 vertices in Group B.

From Fig 7 we can see that
(1) the number of x-coordinate in the anonymization data is half of that in the original data;
(2) the number of vertices in the anonymization data is nearly 4 times as large as that in the original data;
(3) the distribution trend is nearly the same between the anonymization data and the original data;
(4) the change in the anonymization data is flatter than that in the original data.

In the following, we will illustrate the correction of our observation.

We note the number of vertices in Group A that connects $i$ vertices in Group B in the original data as $|A^i_B|$, and note that number in the anonymization data as $|A^i_B'|$. Then, we have

$$|A^i_B'| = |A^{2i-1}_B| + 2|A^{2i}_B| + |A^{2i+1}_B|$$

(1) As each vertex in the anonymization data is produced by the strict 2-order splitting of the original data, and each kind of edges of each vertex are averagely allocated to the two sub-vertices, the number of x-coordinate in the anonymization data is half of that in the original data.

(2) When there is no large fluctuation in the original data, which means,

$$|A^{2i-1}_B| \approx |A^{2i}_B| \approx |A^{2i+1}_B|$$

we have

$$|A^i_B'| \approx 4|A^i_B|$$

That is why the number of vertices in the anonymization data is nearly 4 times as large as that in the original data.

(3) Because

$$|A^i_B'| = |A^{2i-1}_B| + 2|A^{2i}_B| + |A^{2i+1}_B|$$

we know that each point in the anonymization data can be and only can be linear expressed by the value of 3...
When the actual result is $\sum_{i=1}^{20} |A_B^i|$ or the actual result is $\sum_{i=1}^{20} |A_B^i|$, according to the analysis in Section 4.5.1, the predictable error is $\frac{1}{2} |A_B^3| + |A_B^4| + \frac{1}{2} |A_B^{21}|$.

Or the actual result is

$$n = \frac{1}{2} \sum_{i=3}^{10} |A_B^i|'$$

Similarly, when the predictable error is

$$\frac{1}{2} |A_B^{21}| - \frac{1}{2} |A_B^5|$$

the actual result is

$$n = \frac{1}{2} \sum_{i=3}^{10} |A_B^i|'$$

When $|A_B^{21}| \approx |A_B^5|$, $|A_B^{21}| / 2 - |A_B^5| / 2 \approx 0$. When no such results whose predictable errors nearly equals to 0 exist, we
can eliminate the influence of the predictable errors by calculating its proportion in the anonymization result. The simplest method is to assume $|A_B^i| = |A_B^j|$, $i \neq j$. If we can deduce the proportion of all the error items, $|A_B^{p-2}|$, $|A_B^{p-1}|$, $|A_B^{q+1}|$ and $|A_B^{q+2}|$, in the exact result $\sum_{i=p}^{q} |A_B^i|$, according to the distribution of anonymization data, the corrected result will be more accurate.

Fig. 8 is the data graph composed of the data related to the query result, the data not related to the query result, and the data related to the predictable errors, when the original graph receives a query about “the number of vertices in Group A whose number of edges connecting to the vertices in Group B is between 5 and 20”. Here, the data not related to the query result means the part of data that does not meet the requirement in the usability query, which are the vertices whose number of edges is not between 5 and 20. The data related to the query result means the part of data that meet the requirement in the usability query, which are the vertices whose number of edges is between 5 and 20. The predictable error means the vertices in the original data corresponding to the error items in the anonymization data.

Fig. 9 is the data graph composed of the data related to the query result, the data not related to the query result, and the data related to the predictable errors, when the anonymization graph receives a query about “the number of vertices in Group A whose number of edges connecting to the vertices in Group B is between 2 and 10”. Here, the data not related to the query result means the part of data that does not meet the requirement in the usability query, which are the vertices whose number of edges is not between 2 and 10. The data related to the query result means the part of data that meet the requirement in the usability query, which are the vertices whose number of edges is between 2 and 10. The predictable error means the vertices in the original data corresponding to the error items in the anonymization data.

Because the anonymization data has the same distribution with the original data, we can see that in Fig.8 and Fig.9, the numbers of each part of vertices are nearly the same. If we can correct the errors when calculating the final result by calculating the proportion that the predictable errors are of the query result in the anonymization data, we will largely reduce the errors and improve the accuracy.

7 Experiment

In this section, we use two representativesynthetic data and two real social network data to verify the algorithm proposed in this paper. The code is written in Visual C# on Microsoft SQL Server 2008. The experiment in preformed with CPU of Intel Core i3 3.20GHz, internal memory of 4GB, and hard disk of 500GB. We use SCQ to test the exact constraint query and unilateral constraint query.

The data in our experiments are:

(1) complete graph $K_{400}$, which is synthetic data. We divide the 400 vertices into 4 groups in our experiment, and
there exists and only exists one undirected edge between each two edges.

(2) mean graph $E_{20}$, which is synthetic data. The whole graph is divided into 4 groups, there are each 20 vertices with the connected degree of each group to other groups from 1 to 20.

(3) Wiki-Vote, which is real data. This data is generated from the case that Wikipedia vote for the election of administrator of each plate since the establishment of Wikipedia to the year 2008. Each vertex presents a user, and directed edge $\langle i, j \rangle$ presents the user $i$ trusts and votes to the user $j$.

(4) soc-Epinions, which is real data. This data is from the online social network, who-trust-whom, of the general consumers’ comments, website Epinions.com. Registered users can show whether he trust other users. Each vertex presents a user, and directed edge $\langle i, j \rangle$ presents the user $i$ trusts and votes to the user $j$.

The frequently-used characters of data (3) and (4) are shown in Table 4.

Firstly, perform three kinds of queries, exact constraint, unilateral upper bound constraint, and unilateral lower bound constraint with the constraint of $n$ edges. Then access the exact result on original graph $r_n$ and anonymization graph $r'_n$ given different values to $n$. Note $Re_n = |r_n - r'_n| / r_n$ as the relative error of query result with numeric constraint $n$, the edge number $n$ in exact constraint query as $n$, the edge number $n$ in unilateral upper bound constraint query as $\text{Bound}$. Finally, record the relative error $Re$ with the increase of $n$ and $\text{Bound}$, which is shown in Figure 5.

From the analysis of utility, if we make correction to query result, the of error is mainly caused by the large change of vertex numbers that approximately satisfy query constraints, which can be seen in Figure 5 (a) and (b). As in $K_{400}$, $|X^0| = 100$, and $\forall i \neq 100$, there is $|X^i| = 0$. Thus, in exact constraint query, the predicted error is large when $n = 100$. Similarly, in unilateral constraint query, the predicted error is large when $\text{Bound} = 100$. When query results stay at a value will never change, the predicted error will decrease to 0, which can be seen when $n \in [0,90]$ in Figure 5 (a) and $\text{Bound} \in [0,90]$ in Figure 5 (b). Figure 5 (c) presents the similar relation, the predicted error is large when $n = 0$ and $n = 20$.

<table>
<thead>
<tr>
<th>Table 4 Tested Data and Their Basic Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>Vertex Number</td>
</tr>
<tr>
<td>Edge Number</td>
</tr>
<tr>
<td>Maximum WCC Vertex Number</td>
</tr>
<tr>
<td>Maximum WCC Edge Number</td>
</tr>
<tr>
<td>Maximum SCC Vertex Number</td>
</tr>
<tr>
<td>Maximum SCC Edge Number</td>
</tr>
<tr>
<td>Average Clustering Coefficient</td>
</tr>
<tr>
<td>Triangle Numbers</td>
</tr>
<tr>
<td>Closed Triangular Score</td>
</tr>
<tr>
<td>Diameter</td>
</tr>
<tr>
<td>90% Effective Diameter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5 Instance of the result of substructure splitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d$</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 6 Instance of Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Bound}$</td>
</tr>
<tr>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>$1.2$</td>
</tr>
<tr>
<td>$1.3$</td>
</tr>
<tr>
<td>$1.4$</td>
</tr>
<tr>
<td>$2.3$</td>
</tr>
</tbody>
</table>

The change of error decreases with the increase of query results when query results change uniformly with the $\text{Bound}$ of unilateral constraint or bilateral constraint, which is shown in Figure 5(d).
Just as discussed in Section 5 of analysis of utility and error, the experiment results show that splitting anonymization method can make an unstable error for the exact constraint queries that may leak privacy, which can protect the privacy that may be leaked. For unilateral constraint query, a common query in data analysis, when the query result number is large, or in other words, the privacy data is relatively safe, the error almost equals to 0. Even though when the actual result is small, the relative error is less than 5%.

Finally, we verify the utility of our proposed algorithm in respect of query performance. Another important character of splitting anonymization algorithm is that the size of data never changes. Though after applying splitting anonymization, the number of vertices is twice the size of original graph, the total quantity of actual storage is not increased. This is because storage of graph largely depends on the edges of graph instead of vertices, and splitting anonymization never changes the number of edges in a graph. Figure 4 (a) and Figure 4 (e) can verify this character in detail, and its storage form is shown in Table 6. Although the number of vertices increase in the new graph, query complexity is not changed as the main cause of complexity (edge information) is not changed.

8 Related Work

Techniques for anonymizing social networks can be broadly classified into three categories: generalization based on clustering of vertices; deterministic alteration of the graph by edge additions or deletions; randomized alteration of the graph by addition, deletion or switching of edges.

In the first category, Campan and Truta [14] study the case in which vertices contain additional attributes, e.g., demographic information. They propose to cluster the vertices and reveal only the number of intra- and inter-cluster edges. The vertex properties are generalized in such a way that all vertices in the same cluster have the same generalized representation. Hay et al. [12, 13] propose to generalize a network by clustering vertices and publishing the number of vertices in each partition together with the densities of edges within and across partitions. Tassa and Cohen [15] consider a similar setting and propose a sequential clustering algorithm that issues anonymized graphs with higher utility than those issued by the algorithm of Campan and Truta.

Cormode et al. [16, 17] consider a framework where two sets of entities (e.g., patients and drugs) are connected by links (e.g., which patient takes which drugs), and each entity is also described by a set of attributes. The adversary relies upon knowledge of attributes rather than graph structure in devising a matching attack. To prevent matching attacks, their technique masks the mapping between vertices in the graph and real-world entities by clustering the vertices and the corresponding entities into groups. Zheleva and Getoor [18] consider the case where there are multiple types of edges, one of which is sensitive and should be protected. It is assumed that the network is published without the sensitive edges and the adversary predicts sensitive edges based on the observed non-sensitive edges.

In the second category of methods, Liu and Terzi [19] consider the case that a vertex can be identified by its degree. Their algorithms use edge additions and deletions in order to make the graph k-degree anonymous, meaning that for every vertex there are at least k − 1 other vertices with the same degree.

Zhou and Pei [20] consider the case that a vertex can be identified by its radius-one induced subgraph. Adversarial knowledge stronger than the degree is also considered by Thompson and Yao [21], who assume that the adversary knows the degrees of the neighbors, the degrees of the neighbors of the neighbors, and so forth. Zou et al. [22] and Wu et al. [23]
assume that the adversary knows the complete graph, and the location of the vertex in the graph; hence, the adversary can always identify a vertex in any copy of the graph, unless the graph has other vertices that are automorphically-equivalent. Some important graph queries [26-30] should consider the above anonymous strategies to prevent leakage.

In the last category of methods, Hay et al. [24] study the effectiveness of random perturbations for identity obfuscation. They concentrate on degree-based re-identification of vertices. Given a vertex v in the real network, they quantify the level of anonymity that is provided for v by the perturbed graph as (\(\max_u \{Pr(v | u)\}^{-1}\)), where the maximum is taken over all vertices u in the released graph and Pr(v | u) stands for the belief probability that u is the image of the target vertex v. By performing experimentation on the Enron dataset, using various values for the number h of added and removed edges, they conclude that in order to achieve a meaningful level of anonymity for the vertices in the graph, h has to be tuned so high that the resulting features of the perturbed graph no longer reflect those of the original graph.

9 Conclusion

In this paper, we first emphasize the importance of privacy protection in social network, and make a brief analysis on existing data anonymizing methods and point out the shortcomings of these works. Secondly, we propose our new data anonymization method based on vertex splitting, which is Splitting Anonymization. We strictly analyze the privacy protection theoretically aiming at all kinds of existing attacks. Furthermore, we elaborate the utility of anonymization data for utility verification queries. Finally, the reasonableness of our design is verified by large amount of experiments. In the future, we plan to propose targeted algorithms according to different characters of different data in vertex splitting process to satisfy the need of privacy protection and utility.

Acknowledgments

This work is supported in part by the NSFC (61202087, 61332006, U1401256, 61328202); the Fundamental Research Funds for the Central Universities (Grant nos. N120404012, N130504006); the Key Projects in the National Science & Technology Pillar Program no. 2014BAI17B02-03, and the Open Foundation of WUHAN University no. SKLSE2012-09-40.

References

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1342916&isnumber=29567

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4565015&isnumber=4565003

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4061343&isnumber=4061322

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5207644&isnumber=5207632

doi: 10.1109/WKDD.2009.34
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4771997&isnumber=4771855

doi: 10.1109/TKDE.2011.232
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6081867&isnumber=4358933

doi: 10.1109/ICCIS.2010.245

doi: 10.1109/GreenCom-CPSCom.2010.33
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5724922&isnumber=5724803

doi:10.1142/S0218479302001648
URL: http://arbor.ee.ntu.edu.tw/archive/ppdm/Anonymity/SweeneyKA02.pdf

doi: 10.1109/92.748202
URL:http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=748202&isnumber=16156


[22] L. Zou, L. Chen, and M. T. Ozsu. K-automorphism: A general framework for privacy preserving network publication. PVldb,


Supplementary material

Click here to download Supplementary material: Author Biographies and Photos.rar