Privacy preservation for social networks sequential publishing

Safia Bourahla, Maryline Laurent, Yacine Challal

Privacy Preservation for Social Networks Sequential Publishing

Safia Bourahla[a,b], Maryline Laurent[c], Yacine Challal[a,d]

[a]*Laboratoire des Méthodes de Conception des Systémes, Ecole nationale Supérieure d'Informatique BP 68M, 16309, Oued-Smar, Alger, Algérie. http://www.esi.dz*
[b]*Université Blida 2, Blida, Algeria*
[c]*SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, France*
[d]*Centre de Recherche sur l'Information Scientifique et Technique, CERIST, ALger, Algérie*
Email: sa_bourahla@esi.dz, maryline.laurent@telecom-sudparis.eu, y_challal@esi.dz

Abstract

The proliferation of social networks allowed creating a big quantity of data about users and their relationships. Such data contain much private information. Therefore, anonymization is required before publishing the data for data mining purposes (scientific research, marketing, decision support, etc). Most of the anonymization works in social networks focus on publishing one instance while not considering the need for anonymizing sequential releases. However, many cases show that sequential releases may infer private information even though individual instances are anonymized. This paper studies the privacy issues of sequential releases and proposes a privacy preserving solution for this case. The proposed solution ensures three privacy requirements (users' privacy, groups' privacy and edges' privacy), and it considers the case where many users and groups may share the same profiles. Some experiments over some complex queries show that the utility of the released data is better preserved than other solutions, with regard to the privacy of users, groups and edges.

Keywords: Privacy preserving, Social networks, Anonymization, Sequential releases

Email address: sa_bourahla@esi.dz (Safia Bourahla)

1. INTRODUCTION

Nowadays, social networks are becoming an integral part of daily interactions of modern life. Facebook claims over 1.4 billion monthly and 900 million daily active users [1]. By creating personal profiles, that contain demographic information, social networks allow users to create and join groups which have different interests across political, economic, and geographic borders. The affiliation of users to groups is considered as rich information that can be used by network researchers, sociologists, application designers and others for data mining tasks. For example, authors in [2] propose new metrics, namely the dispersion and the monopoly coefficients to refining the study of bipartite structures, particularly, when there is a community neighborhood overlapping. These two metrics are used to capture the intricate patterns observed in real social networks.
As a result, the full publication of this relationship information meets the need of data miners and allows them to perform good data mining tasks. Social networks include two types of data publishing:

1. One release: when a single instance of the social network is published.
2. Sequential releases: when several instances of the same social network are published over time to reflect its evolution.

However, the full publication of these social network data violates the users' privacy because an adversary can infer the affiliation links that the victim would like to keep private. To overcome this problem, researchers have proposed several techniques [3] [9] [18][20] to anonymize the data before their publication so that the privacy of users is preserved and the needs of data miners are satisfied. These techniques deal with different privacy risks which are [26]:

- Identity disclosure: the adversary can identify the victim from the published graph.
- Content disclosure: the sensitive attributes are identified and associated to the victim in the published graph.
- Social link disclosure: a sensitive relationship between two users is revealed.
- Affiliation link disclosure: the adversary can identify whether the victim belongs to a particular group.

One drawback of these techniques is that they consider the "one release case" only. However, to better answer the needs of data analyzers, it is recommended to publish sequential anonymized releases for the same social network to reflect its evolution in time. Applying directly these techniques to publishing sequential releases, by anonymizing each release independently, leads to privacy breaches because by comparing the different published releases, an attacker can infer private information and violate the user privacy. Thus, the sequential releases case is more challenging to ensure the privacy adequate level while preserving the data utility for the analysis tasks. Indeed, the proposed solution should consider the previous published releases when it tries to anonymize the current data.

This paper designs a solution to the problem of privacy preserving sequential releases of social networks based on a two-step methodology. First, "one release case" is considered for a social network being represented as labeled bipartite graph, and it leads to identifying a *safely partitioning condition*, detected as insufficient for the sequential releases case. Indeed, the *safely partitioning condition* enables to anonymize the bipartite graph by grouping nodes (users and groups) into classes and masking the true mapping between nodes and attribute values. However, in "the sequential releases case", where nodes can change classes from one release to another, an attacker can still identify the targeted individual, and/or her/his affiliations by comparing in between various releases. Finally, the paper proposes a solution for the sequential releases case, for which a *safely permuting condition* is introduced to guarantee both privacy and utility of the published bipartite graph.

The rest of the paper is organized as follows: Section 2 reviews related works. Section 3 presents the data model, attacker's knowledge, the privacy and the utility requirements and the social network privacy preserving (SNPP) problem of sequential releases. To well understand the SNPP problem for sequential releases, Section 4 proposes a solution to anonymizing one instance of the social network and shows that this solution fails to give the required privacy level in the case of sequential releases. Section 5 proposes our sequential releases SNPP solution. Section 6 reports experimental results. Section 7 ends up this work with conclusions and future directions.

2. RELATED WORKS

There are several works dealing with privacy problem in the context of publishing social networks data. These works can be grouped into five cat-

egories, depending on the type of the published graph (output of the solution): *Anonymous graph, Disturbed graph, Clustered graph, Uncertain graph and Statistical results*. Most of the existing works deal with the problem of privacy preserving in the context of one release publication. We give examples of these works for each category:

Anonymous graph: In this category, the published anonymous version of the original graph ensures that an attacker, with a certain background, cannot identify an individual with a probability greater than $\frac{1}{k}$ in the anonymous graph. According to the attacker background, several techniques have been studied including:

- k-degree anonymity [3]: to prevent an attacker, whose background is the victim node's degree, to re-identify the victim node in the published graph.
- k-neighborhood anonymity [4]: to prevent the risks of identity disclosure. It considers an attacker having the victim node's neighborhood as background.
- k -automorphism anonymity [5][6]: to prevent an attacker, whose background is the subgraph constructed by the immediate neighbors of a victim node, to re-identify the victim node in the published graph. So, for any subgraph X, this technique is applied to construct an anonymous graph that contains at least k subgraphs isomorphic to X.

Other techniques have been proposed such as k-isomorphic graph [7], and probabilistic indistinguishability [8]. Each technique considers an attacker with a particular background.

Disturbed graph: The original graph is modified to produce the disturbed one. Authors in [9] propose a solution to prevent the risk of the identity and link disclosure. Their solution is based on a method that adds noise to the data in the form of random additions, deletions or switching of edges. So, the solution randomly deletes n edges from the graph and adds n fake edges to the graph. Authors in [10] argue that the approach proposed in [9] has impact on both real and spectral graph characteristics. Therefore, they propose a solution which considers the possible impact that a randomly selected edge will have on the graph's spectrum, in order to maintain the utility. Authors in [11] modify the original graph by using a method based on a randomized perturbation matrix.

Clustered graph: In this category, the graph is partitioned into clusters. First, n clusters are created so that each cluster contains at least k similar nodes. Then the nodes of each cluster are modified to become indistinguishable and the edges are generalized. Authors in [12] propose a solution to prevent the risk of identity disclosure for simple graphs. In [14], the case of labeled nodes has been considered, so the clusters are created, based on attributes and neighborhood similarity. Similar anonymization schemes are used in [18]. As said, authors in [18] propose a solution that first divides the graph of n nodes into several clusters; and each (cluster) contains at least k nodes. Then they replace the subgraph, which contains similar nodes into a cluster, by a super node. Also, in [30], Yu et al. propose a clustering algorithm to preserve privacy for social network. The proposed algorithm ensures the privacy of nodes attributes values and community structures simultaneously.
Uncertain graph: The original deterministic graph is converted into an uncertain graph by associating probabilities to edges (the probability represents the uncertainty level of the edge presence in the original graph). In [19], Bonchi et al. propose the k-obfuscation principle which ensures that an attacker cannot deduce the node identity in the published graph with a probability greater than $log_2 k$. In [20] Boldi et al. propose the principle (k, ϵ)-obfuscation which requires that the graph be k-obfuscated for at least (1-ϵ)n nodes. Other techniques have been proposed such as the MaxVar approach [21]. The MaxVar approach is an improvement over (k, ϵ)-obfuscation. It mitigates the problem of low anonymity. Another uncertain graph approach is proposed in [31].
Statistical results: In this category, both statistics and responses to queries over the social network are published. To ensure that an attacker cannot deduce the presence of an individual or a link, using these statistics and responses to queries, the *differential privacy* concept [28] needs to be satisfied. In social networks, it is said that an algorithm A is ϵ-differentially private, with $\epsilon < 1$, if it satisfies the following equation:

$$Pr(A(G) = s) \leq e^{\epsilon} Pr(A(\tilde{G}) = s) \quad (1)$$

where G and $\tilde{G}$ are two neighboring graphs and s represents an outcome produced by A.
According to the notion of neighboring graphs, we note two levels:

- Node level differential privacy [22]: two graphs are neighbors if they differ by a single node and its adjacent edges; its objective is to en-

sure that an attacker cannot deduce the presence or the absence of a particular node.

- Edge level differential privacy [23]: two graphs are neighbors if they differ by a single edge; its purpose is to ensure that an attacker cannot deduce the presence or the absence of a particular edge in the graph.

The case of sequential releases is less considered than that of one release. Moreover, there is little work concerning solutions that enable publishing sequential releases of social networks data while preserving the privacy. Hence, several privacy issues are still challenging in the case of sequential releases. In [24], Wang et al. study the problem of real-time spatio-temporal crowd-sourced social network data publishing over infinite streams. They propose a solution RescueDP "REal-time Spatio-temporal Crowd-soUrcEd Data Publishing with Differential Privacy" to protect any users' mobility trace. So, they group regions with small statistics together by considering the similarity of data change, and then they add a Laplace noise to each group. Their solution considers a rather different privacy issue of user's mobility than in our paper focusing on users', groups' and edges' privacy preserving. Also, their solution does not publish the whole social network, it allows publishing only statistics (user mobility) over the social networks. In [13], Bhagat et al. propose a solution which allows publishing anonymized versions of the social network in the context of sequential releases. Authors in [13] consider the problem of privacy in social networks modeled as a bipartite graph; they hide the mapping between a node and the corresponding entity by partitioning the set of nodes into groups of size k. They provide methods which use link prediction algorithms to model the evolution of the social network and to predict the future structure. The prediction is used to choose an anonymization which is expected to remain safe and useful for future releases. Their solution provides methods to anonymize a dynamic social network when new nodes and edges are added to the published social network.
In our work, we consider the problem of the identity and affiliation links privacy preserving in social networks sequential releases, and we propose a solution that produces a published bipartite graph which is both anonymous and clustered. Indeed, our solution uses techniques that group nodes into classes with size greater than or equal to k and achieves the $k-$anonymity principle. In our work we consider both cases, that of nodes/edges adding and that ofnodes/edges deleting; as well as the case where several nodes, users and groups, share the same attributes' list values.

3. SOCIAL NETWORK PRIVACY PRESERVING PROBLEM STATEMENT

This section defines the Social Network Privacy Preserving (SNPP) problem in the context of publishing sequential releases of the same evolving social network.
To represent the social network data, we use a labeled bipartite graph which has two types of nodes that represent users and groups. Each node has several attributes representing the node's profile. These attributes are called, in this paper, "the node attributes list". The bipartite graph has also a set of edges, representing the affiliation links between users and groups.

3.1. Bipartite graph model

We represent a time varying social network as a labeled bipartite graph $G_t = (V_t, W_t, E_t, L_{Vt}, L_{Wt})$, where:

- V_t : is a set of nodes which represents the users who are present in the social network at time t.
- W_t: is a set of nodes that represents the groups and other online contents of interest such as, photos, web pages.... These nodes are present in the social network at time t.
- E_t: is a set of edges. An edge $e(u_x,u_y)_t$ represents an affiliation link between a user u_x and a group u_y. E_t is the set of edges that are present in the social network at time t.
- L_{Vt}: is a set of attributes that describes a node $u_x \in V_t$.
- L_{Wt}: is a set of attributes that describes a group $u_y \in W_t$.

In practice, the data recipient is interested in a subset of the attributes offered by the data provider, for example: a pharmaceutical company needs only an attribute list with job, sex and age to find out, for example, a disease person [25], hence, the data provider publishes a tailored attribute list for each data recipient. So, as in the social network, there is a big number of nodes and the attributes list has a small number of possible values, we can find several nodes which share the same attributes list values. Indeed, for instance, if the data recipient is interested in an attributes list = (age, job, hobby), too many users are likely to share the same list (27, lawyer, reading)

in a real social network.
Let $\mathfrak{G} = (\bar{G}_1, \bar{G}_2, \bar{G}_3 \dots \bar{G}_T, \dots)$ be a sequence of published anonymous bipartite graphs representing one social network at time 1,2,... T,... respectively.

3.2. Considered actions

From one bipartite graph, $G_{t-1} = (V_{t-1}, W_{t-1}, E_{t-1}, L_{Vt-1}, L_{Wt-1})$, to the next one, $G_t = (V_t, W_t, E_t, L_{Vt}, L_{Wt})$, we consider the following actions:

1. **Act1 (Persisting nodes):** when a node N (user or group) is present in the current and previous social network sequence i.e. $N \in G_t$ and $N \in G_{t-1}$.
2. **Act2 (Persisting edges):** when an edge $e(u_x,u_y)_t$ is present in the current and previous social network sequence i.e. $e(u_x,u_y)_t \in G_t$ and $e(u_x,u_y)_t \in G_{t-1}$.
3. **Act3 (Adding nodes):** when a node N (user or group) is present in the current, but not in the previous, social network sequence i.e. $N \in G_t$ and $N \notin G_{t-1}$.
4. **Act4 (Deleting nodes):** when a node N (user or group) is present in the previous, but not in the current, social network sequence i.e. $N \in G_{t-1}$ and $N \notin G_t$.
5. **Act5 (Adding edges):** when an edge $e(u_x,u_y)_t$ is present in the current, but not in the previous, social network sequence i.e. $e(u_x,u_y)_t \notin G_{t-1}$ and $e(u_x,u_y)_t \in G_t$.
6. **Act6 (Deleting edges):** when an edge $e(u_x,u_y)_{t-1}$ is present in the previous, but not in the current, social network sequence i.e. $e(u_x,u_y)_{t-1} \notin G_t$ and $e(u_x,u_y)_{t-1} \in G_{t-1}$.

Note that in this paper, we do not consider the action of the modification of the node attribute lists values. But, it can be implemented, using deleting and then adding nodes i.e. **Act4** and **Act3**.

3.3. Attacker knowledge

This paper focuses on the attacks in where the attacker aims to re-identifying a target node and its affiliation links from the published bipartite graph. Contrary to work [13], where the attacker has no knowledge, we assume that the attacker knows only the node's (targeted individual or group) attributes list values and nothing else about the structure information (e.g. the node's degree). For example, the attacker knows that the targeted individual, Alice (who can be her/his co-worker), has attributes list values (26

years old, engineer), or she/he knows that the targeted group has attributes list values (politics).

3.4. SNPP Problem statement

Definition 1. *privacy level: k*
The constant k is the privacy level if: each node N, in the published social network, is indistinguishable from $(\lambda_N * k) - 1$ *other nodes, where* $\lambda_N - 1$ *represents the number of nodes which:*

1. *Have the same type as node N (group or user).*
2. *Share the same attributes list values as node N.*
3. *Are published at the same time as node N.*

Definition 2. *SNPP problem:*
Let us consider $\mathfrak{G}$*, the sequence of published bipartite graphs, representing the social network at times 1,2,...* $t-1$*,* G_t *the bipartite graph, representing the social network at time t, k, the privacy level,* $\bar{G}_t$*, the anonymous version of* G_t*, and an attacker with the background described in section 3.3 and who has access to all published bipartite graphs: i.e.* $\mathfrak{G}$ *and* $\bar{G}_t$*.*
The SNPP problem, denoted by:

$$SNPP(\mathfrak{G}, G_t, k, attacker) \rightarrow \bar{G}_t$$

consists in producing the anonymous bipartite graph $\bar{G}_t$ *that ensures the following requirements:*

- $\boldsymbol{RQ_1}$ ***(Users' privacy)****: the attacker cannot re-identify the targeted individual (i.e. find which node is the targeted individual) with probability greater than* $\frac{1}{\lambda_x k}$*, where (*λ_x*-1) represents the number of users' nodes which have the same attributes list values and which are published at the same time as the targeted individual.*

- $\boldsymbol{RQ_2}$ ***(Groups' privacy)****: the attacker cannot re-identify the targeted group (find which node is the targeted group) with probability greater than* $\frac{1}{\lambda_y k}$*, where (*λ_y*-1) represents the number of groups' nodes which have the same attributes list values and which are published at the same time as the targeted group.*

- $\boldsymbol{RQ_3}$ ***(Edges' privacy)****: the attacker cannot determine the existence of a link between the targeted individual,* u_x*, and the targeted group,* u_y*, with probability greater than* $\frac{1}{k}$*.*

- ***RQ*$_4$ *(Utility requirement)***: *using the anonymous bipartite graph,* $\bar{G}_t$*, we should be able to answer different mining queries with high accuracy.*

We measure the utility of the published bipartite graph by computing the relative query error, as used in [15][16], for which some answers to queries are obtained from the anonymous bipartite graph and compared to the original bipartite graph. This error is defined as follows:

U: $Q \rightarrow R$

$$q \rightarrow \frac{|answer(q_G) - answer(q_{\bar{G}})|}{answer(q_G)} \tag{2}$$

Where Q is the set of queries, q is a query, answer(q_G) is the answer of the query q over the original bipartite graph and answer($q_{\bar{G}}$) is the answer of the query q over the anonymous bipartite graph.

In the following sections, we construct our privacy preserving sequential releases of social networks through anonymization techniques that group nodes into classes of size at least k, and we give formal proofs to ensure the requirements $\mathbf{RQ}_1$, $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$. We evaluate the utility accuracy of our solution in section 6.

4. SNPP-1RELEASE AND LIMITATIONS

To well understand the SNPP problem, in the case of sequential releases, we first study the case of only one release. First, we present an overview of an anonymization solution for a single bipartite graph, we define a *safely partitioning condition*, and we present the associated algorithm, before identifying the solution limitations.
In Table 1, we summarize the terminology and notations used in this paper.

4.1. Overview

To produce the published bipartite graph, we use anonymization techniques. So, we create classes for users and others for groups, where each class contains at least k nodes and we publish only the number of edges between classes. In order to keep the attributes lists values distribution unchanged, in the anonymous bipartite graph, we do not generalize all the attributes

Table 1: NOTATIONS AND TERMINOLOGY

Notations	***Meaning***
N1.attributes list values	The node N1 attributes list values
N1.attributes list values = N2.attributes list values	The nodes N1 and N2 have the same attributes list values
N1 = N2	N1 and N2 represent the same node
X	The partition containing the users' classes
Y	The partition containing the groups' classes
$\bar{E}$	The set of edges of the anonymous bipartite graph
C1.attributes lists values	The set of attributes lists values of nodes belonging to C1
| C1.attributes lists values ∩ C2.attributes lists values|	The number of nodes having the same attributes list values between C1 and C2
$\|C_x\|$	The class C_x size
L_{CX}	The set of attributes lists that describe a class C_X *in* X
L_{CY}	The set of attributes lists that describe a class C_Y *in* Y
C. PermutSet	The set containing classes: B∈ C. PermutSet if: |C.attributes list values ∩ B.attributes list values | $\geq \frac{k}{2}$
CondidatG	The set containing the nodes that can be added to the current class without violating the safely partitioning condition
CondC	The set containing classes to which a given node u can be added without violating the safely partitioning condition
Bipartite graph	A bipartite graph is a graph whose vertices can be divided into two disjoint and independent sets V and W so that every edge connects a vertex in V to one in W
λ_N	The frequency of the node's N attributes list values

lists values to be the same for a class of nodes. Instead, for each class, we publish all the attributes lists values of the nodes without revealing the true mapping. This method has the same effect as the partitioning approach proposed by Cormode et al. [15].

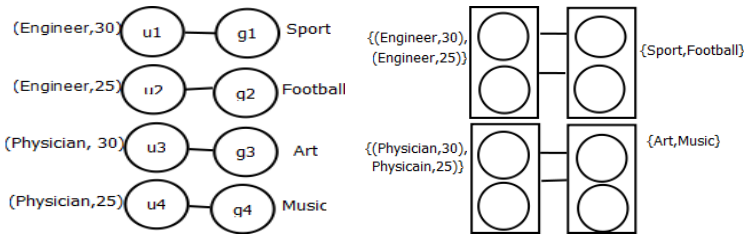
To ensure that this technique preserves the utility within classes, we group nodes into classes so that the utility cost would be lower. We measure the utility cost by quantifying the similarities between the nodes' attributes lists values of each class. It is obvious that the higher the similarity, the lower the utility cost. So, we group nodes which have similar attributes list values in the same class. For example, grouping a node which has an attributes list values = (Engineer, 30) with another node which has an attributes list values = (Engineer, 25) is better than grouping it with a node which has an attributes list values = (Physician, 40). To measure this similarity, we use the cosine distance.

Figure 1.b gives an example of the published bipartite graph, after anonymizing the bipartite graph in Figure 1.a: by taking the privacy level, $k = 2$. In the original bipartite graph, we have four users and four groups: u_1.attributes list values = (Engineer, 30) and she/he is member of the group g_1 which has as topic "Sport", u_2.attributes list values = (Engineer, 25) and she/he is member of the group g_2 which has as topic "Football", u_3.attributes list values = (Physician, 30) and she/he is member of the group g_3 which has as topic "Art", u_4.attributes list values= (Physician, 25) and she/he is member of the group g_4 which has as topic "Music". Note that the nodes' identifiers i.e. $u_1,\ldots, u_4$ and $g_1,\ldots, g_4$, are not published in the anonymous bipartite graphs, they are only used for explanation. We anonymize this bipartite graph by creating users' and groups' classes such as the size of each class is greater than or equal to 2 and without revealing the true mapping between nodes and attributes lists values. We also ensure that the grouping of nodes into classes is with the minimum utility cost. Finally, we publish the number of edges between any two users' class and groups' class:

To create the first users' class C_1, we put the node u_1 and we search the node which engenders the minimum utility cost among u_2, u_3 and u_4, we notice that the nodes u_2 and u_3 produce the same utility cost, as each node shares one value with u_1 (the value "Engineer" for u_2 and the values 30 for u_3). As in this anonymization technique, when there are several nodes which have the same utility cost, we select the first node. So, here the node to be added to the class C_1 is u_2. hence, C_1.attributes lists values = {(Engineer, 30), (Engineer, 25)}. Similarly, we construct the second users' class and the two

groups' classes, such as C_2.attributes lists values = {(physician, 30), (physician, 25)}, B_1.attributes lists values= {Football, Sport} and B_2.attributes lists values= {Art, Music} respectively.
Using the anonymous bipartite graph, an attacker cannot guess which node has which attributes list values, because the true mapping between nodes and attributes lists values is masked. Example: for C_1, an attacker cannot identify the node, among u_1 and u_2, that has (Engineer, 25) as attributes list values. The attacker cannot also know how nodes connect to each other. For example, the attacker cannot know how u_1 and u_2 connect to g_1 and g_2, as in the anonymous bipartite graph, there is only the total number of edges between C_1 and B_1.
Given an original bipartite graph, the problem of creating an anonymous bipartite graph $\bar{G}_t$, which meets the SNPP privacy requirements with the minimum utility cost is NP-hard.

Indeed, our problem can be reduced to a clustering problem. In fact, let us



(a) original bipartite graph (b) anonymous bipartite graph

Figure 1: Anonymization technique

consider that:

- each data item should be clustered as the node's attributes list values.
- the utility cost of adding the node u to the class C as the utility cost of adding the corresponding data item to the corresponding cluster.
- there are no affiliation relationships between users and groups.

The SNPP problem, in this case, is exactly the clustering problem. So, the SNPP problem is NP-hard.
This anonymization method enables creating only classes of size at least k. In what follows, we show that Property 1 is required to ensure the three SNPP privacy requirements ($\mathbf{RQ}_1$, $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$).

4.2. Safely partitioning condition

This subsection introduces the property *safely partitioning condition*, on which we build our SNPP1R algorithm (see algorithm 1) that preserves the three SNPP privacy requirements of the Definition 2 ($\mathbf{RQ}_1$, $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$).

Property 1. ***Safely partitioning condition***
Let us consider the SNPP problem:

$$SNPP(\mathfrak{G}, G_t, k, attacker) \rightarrow \bar{G}_t$$

with G_t = (V_t, W_t, E_t, L_{Vt}, L_{Wt}), $\bar{G}_t$=(X_t, Y_t, $\bar{E}_t$, L_{CXt}, L_{CYt}) and $\mathfrak{G} = \varnothing$: i.e. the attacker has only access to $\bar{G}_t$.
$\bar{G}_t$ satisfies the three SNPP privacy requirements if:

1. ***SPa*$_1$*:*** *$\forall C_{y_1}, C_{y_2} \in Y_t, \forall N_1 \in C_{y_1}, \forall N_2 \in C_{y_2}$ if N_1.attributes list values=N_2.attributes list values then $C_{y_1} \neq C_{y_2}$.*
2. ***SPa*$_2$*:*** *$\forall C_{x_1}, C_{x_2} \in X_t, \forall N_1 \in C_{x_1}, \forall N_2 \in C_{x_2}$ if N_1.attributes list values=N_2.attributes list values then $C_{x_1} \neq C_{x_2}$.*
3. ***SPa*$_3$*:*** *$\forall$Classes $C_x \in X_t$ and $C_y \in Y_t$, nb is the number of edges between C_x and C_y, nb $\leq k$.*
4. ***SPa*$_4$*:*** *$\forall$Classes $C_x \in X_t \cup Y_t$, $|C_x| \geq k$.*

We call these conditions the safely partitioning condition $\square$.

We give justifications and explanations of the conditions $\mathbf{SPa}_1$, $\mathbf{SPa}_2$, which require that no two group nodes (respectively two user nodes) with the same attributes list values belong to the same class in the anonymous bipartite graph, in section 5.1.

Proof:

First, we prove the probability that an attacker can re-identify the node u_x representing the targeted individual (the targeted group respectively) is less than or equal to $\frac{1}{\lambda_x k}$ ($\frac{1}{\lambda_y k}$ respectively). Let's suppose that:

- there are $(\lambda_x - 1)$ other nodes published at the same time as u_x having the same attributes list values like u_x. u_x and the other $\lambda_x - 1$ nodes belong to λ_x different classes (C_x,C_{x1},..., $C_{x\lambda_{x-1}}$) $\in X_t$, as required by the condition $\mathbf{SPa}_2$.

The probability an attacker knows that the node u_x is the node representing the targeted individual

is: P (u_x) = $\frac{1}{|C_x|+|C_{x1}|+....+|C_{x\lambda_{x-1}}|}$ as the size of each $C_{xi} \geq k$ (the condition **SPa**$_4$)
Then P(u_x)$\leq \frac{1}{\lambda_x k}$.
So, the *safely partitioning condition* guarantees the users' privacy requirement (**RQ**$_1$). Similarly, we prove that the groups' privacy (**RQ**$_2$) requirement is guaranteed by the *safely partitioning condition.*

Finally, we prove the probability that an attacker identifies that a link ex-

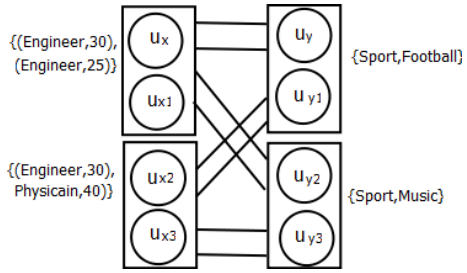


Figure 2: edges privacy

ists between a node u_x, representing the targeted individual, and a node u_y, representing the targeted group, is less than or equal to $\frac{1}{k}$. Let's suppose that:

- there are $(\lambda_x - 1)$ other nodes published at the same time as u_x and having the same attributes list values like u_x. u_x and the other $\lambda_x - 1$ nodes belong to λ_x different classes (C_x,C_{x1},..., $C_{x\lambda_{x-1}}$) $\in X_t$, as required by the condition **SPa**$_2$.

- there are $(\lambda_y - 1)$ other nodes published at the same time as u_y and having the same attributes list values as u_y. u_y and the other $\lambda_y - 1$ nodes belong to λ_y different classes (C_y,C_{y1},..., $C_{y\lambda_{y-1}}$) $\in Y_t$, as required by the condition **SPa**$_1$.

- In the worst case, all nodes in C_x,C_{x1},..., $C_{x\lambda_{x-1}}$ have links with nodes in C_y, C_{y1},..., $C_{y\lambda_{y-1}}$. As shown in Figure 2: by taking u_x.attributes list =(engineer, 30) and u_y.attributes list = Sport. Note that the nodes' identifiers i.e. u_x, u_{x1},..., u_{x3} and u_y, u_{y1},..., u_{y3}, are not published in the anonymous bipartite graphs, they are used only for explanation.

The probability an attacker knows that an edge e exists between u_x and u_y in the published bipartite graph is:

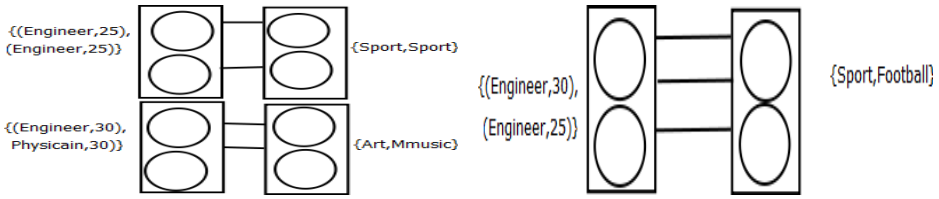
P(u_x, u_y,e) = $\frac{1}{|C_x|+|C_{x1}|+....+|C_{x\lambda_x-1}|} * \frac{1}{|C_y|+|C_{y1}|+....+|C_{y\lambda_y-1}|} * nbedge$
As the size of each $C_i \geq k$ (the condition $\mathbf{SPa}_4$) then:
P(u_x, u_y,e) $\leq \frac{1}{\lambda_x k} * \frac{1}{\lambda_y k} * nbedge$
where *nbedge* is the number of edges whose two endpoints might be u_x and u_y.

- As the targeted group, u_y, may belong to each λ_y classes (Cy,C_{y1},. . ., $C_{y\lambda_{y-1}}$),
- and the targeted individual, u_x, may belong to each λ_x classes (Cx,C_{x1},. . ., $C_{x\lambda_{x-1}}$),
- and the number of edges between any two classes, C_{xi} and C_{yi}, is $\leq k$ (according to the condition $\mathbf{SPa}_3$).

Then: $nbedge \leq k * \lambda_x * \lambda_y$.
So: P(u_x, u_y,e) $\leq \frac{1}{\lambda_x k} * \frac{1}{\lambda_y k} * \lambda_x \lambda_y k$. Therefore P($u_x$, u_y,e) $\leq \frac{1}{k}$
So, the *safely partitioning condition* guarantees the edges privacy requirement ($\mathbf{RQ}_3$).
Figure 3 shows via examples, the violation of the requirements $\mathbf{RQ}_1$, $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$ due to the dissatisfaction of the conditions $\mathbf{SPa}_1$, $\mathbf{SPa}_2$ and $\mathbf{SPa}_3$.

4.3. SNPP-1 Release (SNPP1R) algorithm

We propose a greedy algorithm (algorithm 1) that creates classes with at least k nodes. The SNPP1R algorithm relies on the concept of greedy safe k-grouping, used in [15][16][17], while minimizing the utility cost and considering the two types of nodes, user and group, and our new *safely partitioning condition*. It creates one groups' class then one users' class alternatively until all groups' nodes and users' nodes are grouped into classes. To partition nodes, the SNPP1R algorithm calls the procedure Safely partitioning(X, Y, Z, k), shown in algorithm 2, where X and Y are two partitions; Z is a set of nodes and k is the privacy level.
The procedure Safely partitioning takes each time the first node, not yet grouped, in Z, and it creates a new class C containing this node, line 1. Then it repeatedly adds nodes to C under the *safely partitioning condition* and with the minimum utility cost until the size of C reaches k, lines 2-12. If the size of the created class, C, cannot reach k, the algorithm 2 removes this class and groups its nodes in suitable classes, under the *safely partitioning*



(a) nodes privacy violation (b) edges privacy violation

Figure 3.a shows an example of the violation of nodes privacy requirements due to the dissatisfaction of the conditions $\mathbf{SPa}_1$ and $\mathbf{SPa}_2$. In fact, we notice that the probability to re-identify the targeted individual (Engineer, 25) is equal to $\frac{1}{2}$ which is greater than $\frac{1}{\lambda_x k} = \frac{1}{4}$. Where k is the privacy level and $\lambda_x - 1$ is the number of users' nodes which have the same attributes list values and which are published at the same time as the targeted individual (Engineer, 25) (in this example $k = 2$ and $\lambda_x - 1 = 1$ i.e. $\lambda_x = 2$). Similarly, we also notice that there is a violation of group privacy.

Figure 3.b shows an example of the violation of the edges privacy requirements due to the dissatisfaction of the condition $\mathbf{SPa}_3$. In fact, we notice that the probability to re-identify that the targeted individual (Engineer, 25) is member of the targeted group (Sport) is equal to $\frac{1}{2} * \frac{1}{2} * 3 = \frac{3}{4}$ which is greater than $\frac{1}{k} = \frac{1}{2}$.

Figure 3: Explanations of the Property 1 conditions

Algorithm 1: SNPP1R algorithm

Set W_t =groups' nodes ordered by creation time; Set V_t = users' nodes ordered by creation time; $X_t = \varnothing$;$Y_t = \varnothing$;
repeat
  if $|W_t| > 0$ **then**
    Safely partitioning (Y_t, X_t, W_t, k);
  end
  if $|V_t| > 0$ **then**
    Safely partitioning (X_t, Y_t, V_t, k);
  end
until $|W_t| = 0$ *and* $|V_t| = 0$;

Algorithm 2: Safely partitioning (X: partition, Y: partition, Z: set of nodes, k: privacy level)

U = the first node in Z; Z= Z- {U}; Class C = new class C {U}; X = X ∪ {C}; flag = true;
while $|C| < k$ *and flag* **do**
  Set CondidatG =∅ ; flag = false;
  for $T \in Z$ **do**
    if *adding T to C is under the safely partitioning condition* **then**
      Flag=true; T.cost = the cost to add T to C; CondidatG = CondidatG ∪ {T};
    end
  end
  if | *CondidatG* | > 0 **then**
    $\hat{U}$=t ∈CondidatG | $\forall \hat{t} \in$ CondidatG : t.cost< $\hat{t}$.cost; C = C ∪ {$\hat{U}$ }, Z = Z - {$\hat{U}$ };
  end
end
if $|C| < k$ **then**
  X = X - {C};
  for $u \in C$ **do**
    Set CondC = ∅;
    for $L \in X$ **do**
      if *adding u to L is under the safely partitioning condition* **then**
        L.cost = the utility cost of adding u to L; CondC = CondC ∪ {L};
      end
    end
    $\hat{L}$=t∈ CondC | $\forall \hat{t} \in$ CondC: t.cost < $\hat{t}$.cost; $\hat{L}$= $\hat{L}$∪ {u}
  end
end

Figure 4: SNPP-1 Release algorithms

condition and with the minimum utility cost, lines 13-24.
The procedure Safely partitioning is a greedy algorithm that creates classes with at least k nodes under the *safely partitioning condition* which depends on the sparse property of social networks [17]. In fact, the social networks graphs are relatively sparse [17], i.e. each user is typically member in only a small fraction of all groups, and each group holds only a few users. As a consequence, classes are easier to be found, based on the conditions in property 1, i.e. $\mathbf{SPa}_1$, $\mathbf{SPa}_2$, $\mathbf{SPa}_3$ and $\mathbf{SPa}_4$.

4.4. Complexity of the algorithm SNPP1R

The algorithm SNPP1R cost is in the worst case less than $O(N^2)$, where N ($N = N_{user} + N_{group}$) is the number of nodes, N_{user} is the number of user nodes, and N_{group} is the number of group nodes.
Let's consider adding a node to a class as an elementary operation.
To create a user class, the algorithm SNPP1R has to browse all the user nodes not yet grouped to choose the node that satisfies the *safely partitioning condition* with the minimum utility cost and adds it to the class to be created. So, to add the second user node to the first user class we have to do $N_{user} - 2$ operations (i.e. verify all user nodes not yet grouped), then $N_{user} - 3$ for the third user node etc. To add the second user node to the second user class, we need $N_{user} - (k+2)$ operations etc (k is the class size). So, in total, we have (N_{user}^2) operations. Similarly, we have (N_{group}^2) operations to create groups' classes. So, the complexity of the algorithm is $O(N_{user}^2 + N_{group}^2)$, which is less than $O(N^2)$.
Note that, the algorithm SNPP1R complexity does not depend on the attributes lists values of nodes, as it does not generalize the attributes lists values to anonymize the graph, and it only performs comparison between these attributes lists values to preserve the utility. The operation of "comparison between attributes lists values" is dominated by the operation "adding node to a class ".
The complexity of the algorithm SNPP1R can be enhanced to reach O(N), if at no substantial utility cost (section 4.1). In that case, the algorithm SNPP1R does not browse all the nodes not yet grouped, but instead, as soon as a node is found satisfying the *safely partitioning condition*, it is added directly to the class to be created.

- In the best case, the first visited node satisfies the *safely partitioning condition* (it is the node that will be added to the created class). In

this case, we have in total $O(k|X+Y|)$ operations (X: is the partition containing the users' classes and Y: is the partition containing the groups' classes and k: is the size of class). We have $k|X+Y| = N$ (all the nodes are grouped in classes, $|X+Y|$ is the number of classes and k is the size of a class). As a result, in this case, the complexity of the algorithm SNPP1R is $O(N)$.

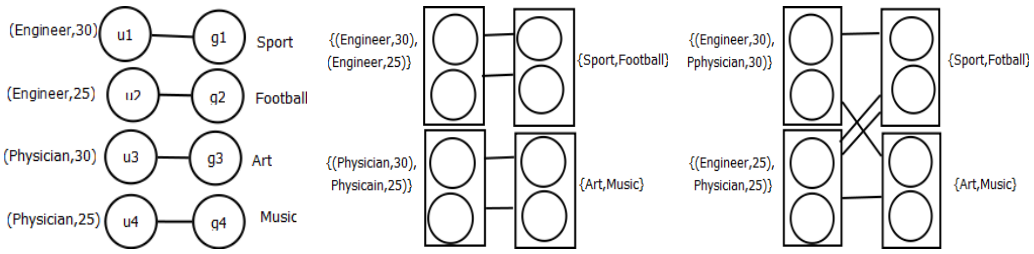
- In the worst case, the node that satisfies the *safely partitioning condition* is the last visited node. In this case, we must go through all the non-grouped nodes, so we come across the case of the algorithm SNPP1R with cost utility function. In this case, the complexity of the algorithm SNPP1R is $O(N^2)$.

Since social networks satisfy "the sparse property", the *safely partitioning condition* has become easy to be satisfied, and the complexity of the SNPP1R algorithm without utility cost function is $O(N)$.

4.5. Inadequacy to sequential releases

Using the SNPP1R algorithm to produce sequential anonymous versions, harvested at different times of the same social network, can lead to privacy violation. That is due to the generation of each anonymous version being made separately, thus leading to nodes changing classes from one release to another, and having their classes' attributes lists values modified. Hence, an attacker making comparison of classes' attributes lists values between releases, can violate the privacy.
As we order the nodes in the SNPP1R algorithm in the same way (by the creation time). So, there are three reasons that lead nodes changing their classes from one release to another:

1. The violation of the condition $\mathbf{SPa}_3$ (**Act5**): adding new edges between old users, who belong to the same class; and old groups, which belong to the same class, can lead to a number of edges between two classes greater than k. As that does not satisfy the condition $\mathbf{SPa}_3$, then nodes must leave their classes until the condition $\mathbf{SPa}_3$ is satisfied again.
2. Deleting nodes (**Act4**): deleting nodes can produce classes with less than k nodes; hence other nodes must be added to these classes until their size is greater than or equal to k.
3. Adding new nodes (**Act3**): the old nodes can be grouped with the new ones if they better preserve the data utility.



(a) original bipartite graph (b) anonymous bipartite graph at T_1 (c) anonymous bipartite graph at T_2

Figure 5: Example of the SNPP1R algorithm's privacy violation in the case of sequential releases

Example: Figure 5 gives an example of privacy violation with the privacy level $k = 2$, the targeted individual = u_2 and her/his attributes list values = (Engineer, 25). At T_1, there are four users and four groups, as depicted in Figure 5.a. Note that the nodes' identifiers i.e. u_1,..., u_4 and g_1, ..., g_4, are not published in the anonymous bipartite graphs, they are only used for explanation. The anonymous bipartite graph $\bar{G}_{T1}$, produced by the SNPP1R algorithm, is presented in Figure 5.b. At T_2, the targeted individual, u_2, joins the group g_1 and the anonymous bipartite graph $\bar{G}_{T2}$, produced by the SNPP1R algorithm, is presented in Figure 5.c.
the attacker is assumed to know that the nodes do not change classes unless there is a violation of the condition $\mathbf{SPa}_3$, or some nodes's adding or deleting. Thus by comparing $\bar{G}_{T1}$ and $\bar{G}_{T2}$, the attacker notices that the targeted individual (Engineer, 25), has changed her/his class and that is due to the violation of the condition $\mathbf{SPa}_3$. Then the attacker can infer that the targeted individual, (Engineer, 25), is member of the groups with topics "Sport" and "Football", which violates privacy.

5. SNPP-SEQUENTIAL RELEASES

This section proposes a privacy preserving solution for publishing sequential releases of the same social network, where addition or deletion of nodes or edges might happen from one release to another.
As discussed in subsection 4.5, the privacy threats are coming from changes occurring among the classes' attributes lists values, in particular, when one of the three following events occur:
(i) the violation of the condition $\mathbf{SPa}_3$ due to newly added edges between old users' classes and old groups' classes (**Act5**). (ii) newly deleted nodes

(**Act4**). (iii) newly added nodes (**Act3**).
To avoid these privacy threats, we need to keep the attributes lists values of each class invariant over time, so the attributes lists values of each class do not change from one release to another, and an attacker is unable to re-identify the targeted individual (group) and her/his affiliation links. This idea is inspired from the notion of $m-$invariance proposed by Xiao et al. [32] for dynamic databases. The $m-$invariance ensures that the sensitive values of each anonymized group (Quasi-Identifier) remain the same over publications.
Those three sensitive actions (i.e. the violation of the condition $\mathbf{SPa}_3$, newly deleted nodes and newly added nodes) are next provided with technical privacy preserving solutions.

5.1. Safe permutation

Safely permuting condition, hereafter defined as property 2, removes the violation of the condition $\mathbf{SPa}_3$ issue identified in Section 4.5. A violation of the condition $\mathbf{SPa}_3$ between two classes can be removed by decreasing the number of edges between these two classes, and permuting nodes belonging to one of these classes with other nodes of the other class, but under the condition that the *safely permuting condition* is respected.
Example: Figure 6 gives an example of a permutation that ensures the requirements $\mathbf{RQ}_1$, $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$ with the privacy level $k = 2$, the targeted individual = u_2 and her/his attributes list values = (Engineer, 25). At T_1, there are four users and four groups: As depicted in Figure 6.a, note that the nodes identifier i.e. $u_1,\ldots,u_4$ and $g_1, \ldots,g_4$, are not published in the anonymous bipartite graphs. They are only used for explanation. The anonymous bipartite graph $\bar{G}_{T1}$ is presented in Figure 6.b. It contains two users' classes, C_1.attributes lists = {(Engineer, 30), (Engineer, 25)} and C_2.attributes lists = {(Engineer, 25), (Physician, 25)}, and two groups' classes, B_1.attributes lists = (Sport, Football), and B_2 .attributes lists = (Art, Music). At T_2, the user u_2 joins the group g_1, hence, there is a violation of the condition $\mathbf{SPa}_3$ as the number of edges between C_1 and B_1 is $3 > k$. To remove this violation of the condition $\mathbf{SPa}_3$ without violating the targeted individual privacy we should:

- Search for two nodes of the same type that have the same attributes list values and belong to different classes.

- Permute these two nodes if that removes the violation of the condition $\mathbf{SPa}_3$ and does not create other ones.

The two nodes that satisfy these requirements are u_2 and u_3. We permute between these two nodes and we produce the anonymous bipartite graph $\bar{G}_{T2}$, presented in the Figure 6.c. We assume that the attacker knows our strategy, then by comparing $\bar{G}_{T2}$ and $\bar{G}_{T1}$, the attacker assumes:

- With probability equals to $\frac{1}{2}$ that the difference between $\bar{G}_{T2}$ and $\bar{G}_{T1}$ is due to the evolution of the social network, because users can join or leave groups over time. In this case, the requirements $\mathbf{RQ}_1$, $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$ are satisfied.

- With probability equals to $\frac{1}{2}$ that the difference between $\bar{G}_{T2}$ and $\bar{G}_{T1}$ is due to removing the violation of the condition $\mathbf{SPa}_3$ between C_1 and B_1. In this case, the groups' privacy requirement is satisfied, we do not permute between groups. For the requirement $\mathbf{RQ}_1$, the attacker identifies that the two users' nodes permuted are u_2 and u_3, so the probability that the targeted individual is u_2 (respectively u_3) is equal to $\frac{1}{2} * \frac{1}{2} = \frac{1}{\lambda_{u_2} k}$,

  1. where the first $\frac{1}{2}$ is the probability that this difference between $\bar{G}_{T2}$ and $\bar{G}_{T1}$ is due to the suppression of the violation of the condition $\mathbf{SPa}_3$,
  2. the second $\frac{1}{2}$ is the probability that the targeted individual is u_2 (respectively u_3),
  3. k is the privacy level, in this case k=2,
  4. and $\lambda_{u_2} - 1$ is the number of nodes which have the same attributes list values and which are published at the same time as the targeted individual ,u_2, in this case $\lambda_{u_2} - 1 = 1$ (u_3).
  5. Hence the users' privacy requirement is satisfied too.

  If the targeted individual is u_2, the probability of affiliation link identification is equal to $\frac{1}{2} * \frac{1}{2} * \frac{1}{2} * 2$, where the first $\frac{1}{2}$ is the probability that this difference between $\bar{G}_{T2}$ and $\bar{G}_{T1}$ is due to the suppression of the violation of the condition $\mathbf{SPa}_3$; the second $\frac{1}{2}$ is the probability that the targeted individual is u_2; the third $\frac{1}{2}$ is the probability that the group selected is the targeted group and 2 is the number of edges between u_2 and B_1.

If the targeted individual is u_3, the probability of affiliation link identification is equal to $\frac{1}{2}*\frac{1}{2}*\frac{1}{2}*1$. So, in both cases the probability of affiliation link identification is less than ($\frac{1}{k} = \frac{1}{2}$). Hence, the edges' privacy requirement is also satisfied.

Property 2. ***safely permuting condition***
A permutation between two nodes of the same type, N_1 and N_2, that belong to two different classes, C_x and C_{zi}, does not violate the requirements $\boldsymbol{RQ}_1$, $\boldsymbol{RQ}_2$ and $\boldsymbol{RQ}_3$ and removes the violation of the condition $\boldsymbol{SPa}_3$ between C_x and C_y if:

1. $\boldsymbol{SPe}_1$*:*$N_1 \in C_x$, $N_2 \in C_{zi}$, $|C_x.attributes\ list\ values \cap C_{zi}.attributes\ list\ values| \geq \frac{k}{2}$, *$k$ is the privacy level.*
2. $\boldsymbol{SPe}_2$*:* $N_1.attributes\ list\ values = N_2.attributes\ list\ values$.
3. $\boldsymbol{SPe}_3$*: The permutation between N_1 and N_2 decreases the number of edges or removes the violation of the condition $\boldsymbol{SPa}_3$ between C_x and C_y and does not create other violations of the condition $\boldsymbol{SPa}_3$.*

We permute nodes under these three conditions ($\boldsymbol{SPe}_1$,$\boldsymbol{SPe}_2$, $\boldsymbol{SPe}_3$), until the violation of the condition $\boldsymbol{SPa}_3$ between C_x and C_y is removed.
We call these conditions: safely permuting condition.□

We justify $\mathbf{SPa}_1$ and $\mathbf{SPa}_2$ (see Property 1): The condition $\mathbf{SPe}_1$ requires that the two nodes which must be permuted, belong to two different classes, and the condition $\mathbf{SPe}_2$ requires that these two nodes should have the same attributes list values. So, to ensure that these two conditions ($\mathbf{SPe}_1$ and $\mathbf{SPe}_2$) are satisfied, we should not put the nodes that have the same attributes list values in the same class, when creating classes. That is why, conditions $\mathbf{SPa}_1$ and $\mathbf{SPa}_2$ have to be satisfied when grouping nodes into classes.
The proof of this property is shown in Appendix A.

5.2. Safely permuting algorithm

Algorithm 3 permutes nodes, when the number of edges increases (with the arrival of new edges) between old classes, to remove the violation of the condition $\mathbf{SPa}_3$. It takes a group class C and for each user class T such as the number of edges between C and T is greater than (k). It permutes nodes so that this number of edges is less than or equal to (k). It calls the procedure safe_class, line 3, which returns: a group class that verifies with the class C, the *safely permuting condition*, if such group class cannot

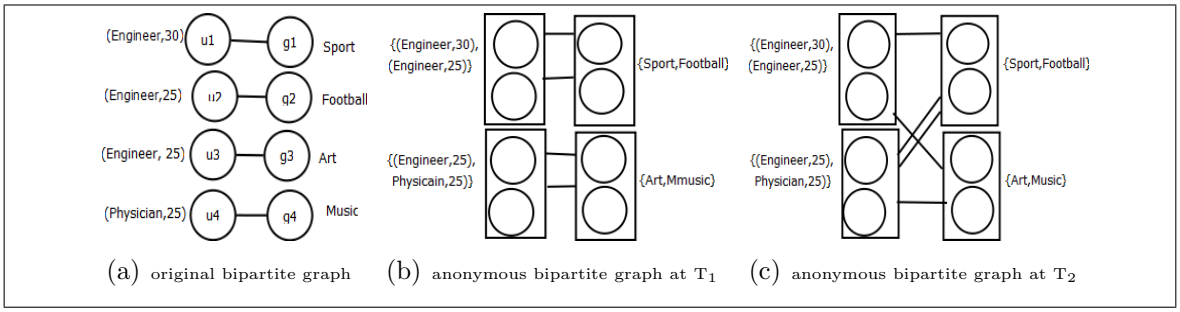


Figure 6: A permutation that preserves the privacy in the case of sequential releases

be found, it returns a user class that respects with T the *safely permuting condition*, if such user class cannot be found, it creates a new group class using the new group nodes as the created class and C comply with the *safely permuting condition*, if this class cannot be created, safe_class creates a new user class using the new user nodes, such as the created class and T check the *safely permuting condition*. If this class cannot be created, it creates a noise group class, using noise nodes, that verifies with the class C, the *safely permuting condition*. Finally, algorithm 3 permutes nodes between C or T and the returned class, lines 4-10. The algorithm 3 repeats this procedure until the violation of the condition $\mathbf{SPa}_3$ is removed.

Procedure of noise groups' class creation: we create a noise group class, by using noise nodes, to remove the violation of the condition $\mathbf{SPa}_3$ between a group class C and a user class T. For the sake of simplicity, we create k noise nodes such as: $\frac{k}{2}$ noise nodes have the same attributes lists values as $\frac{k}{2}$ different nodes in C, and the attributes lists values of the other $\frac{k}{2}$ noise nodes are assigned randomly.

5.3. Dynamic actions: deleting and adding nodes

Deleting old nodes

When a node N_1 leaves the social network, we delay its suppression until a new node N_2 arrives, such as N_1.attributes list values = N_2.attributes list values and N_2 and N_1 have the same type (user or group nodes), then we replace N_1 with N_2.

Indeed, replacing the old node that should be deleted with a new one that has the same attributes list values and the same type, does not induce a change in the classes' attributes lists values. So, an attacker will be unable to guess if the node is deleted or not.

Algorithm 3: Safely permuting (C: group' class, $\hat{X}$: users' partition, k : privacy level)

while *C violate the condition* $\boldsymbol{SPa_3}$ **do**
  T = the first node in $\hat{X}$;
  B = safe_class(C, T, C.PermutSet, T.PermutSet) ;
  if *B.type = C.type* **then**
    Let $N_1 \in$ C and $N_2 \in$ B | N_1.attributes list values = N_2.attributes list values;
    C=C-{ N_1} ; C=C ∪{ N_2} ; B=B-{ N_2} , B=B ∪{ N_1};
  else
    Let $N_1 \in$ T and $N_2 \in$ B | N_1.attributes list values = N_2.attributes list values;
    T=T-{ N_1} ; T=T ∪{ N_2} ; B=B-{ N_2} , B=B ∪{ N_1};
  end
  $\hat{X}$= x $\in X_t$ | the nbr of edges between x and C $> k$;
end

Algorithm 4: SNPP-Sequential Releases algorithm

W_{ti} = set of groups' nodes ordered by the frequency of the nodes attributes lists values; V_{ti} = set of users' nodes ordered by the frequency of the nodes attributes lists values; $X_{ti} = X_{ti-1}$; $Y_{ti} = Y_{ti-1}$; $\hat{W}$:= W_{ti} - W_{ti-1} ; $\hat{V} = V_{ti}$ - V_{ti-1} ; A_W := set of groups' nodes that their deletion is delayed to T_{i-1}; A_V:= set of users' nodes that their deletion is delayed to T_{i-1};
Safely deleting (A_W,W_{ti-1}, W_{ti}) ;
Safely deleting (A_V , V_{ti-1}, V_{ti}) ;
for $C \in Y_{ti}$ **do**
  set $\hat{X}$ = x $\in X_{ti}$ | the nbr of edges between x and C $> k$;
  Safely permuting (C, $\hat{X}$, k);
end
if $|\hat{W}| < k$ **then**
  W_{ti}:= W_{ti}- $\hat{W}$; $\hat{W}$= ∅;
end
if $|\hat{V}| < k$ **then**
  V_{ti}:= V_{ti}- $\hat{V}$; $\hat{V}$= ∅;
end
if $|\hat{W}| \geq k$ *or* $|\hat{V}| \geq k$ **then**
  repeat
    if $|\hat{W}| > 0$ **then**
      Safely partitioning (Y_{ti}, X_{ti}, $\hat{W}$, k);
    end
    if $|\hat{V}| > 0$ **then**
      Safely partitioning (X_{ti}, Y_{ti}, $\hat{V}$, k);
    end
  until $|\hat{W}| = 0$ *and* $|\hat{V}| = 0$;
end

Figure 7: SNPP-Sequential Release algorithms

We call this procedure the Safely deleting procedure.

Adding new nodes

Unless the new nodes replace the old ones, that must be suppressed, the new nodes are never grouped with the old ones:

- If the number of new nodes is less than k, we delay the publication.
- Else, we group the new nodes together in new classes, under the *safely partitioning condition.*

We call this procedure the adding nodes procedure.

5.4. SNPP solution

This subsection, proposes a property (*SNPP solution*) that ensures the three SNPP privacy requirements of Definition 2 for the sequential releases case (i.e. where addition or deletion of nodes or edges might happen from one release to another).

Property 3. ***SNPP solution:***
Let:

$$SNPP(\mathfrak{G}, G_t, k, attacker) \rightarrow \bar{G}_t$$

with G_t *=* (V_t , W_t , E_t , L_{Vt} , L_{Wt}), $\bar{G}_t$=(X_t, Y_t, $\bar{E}_t$, L_{CXt}, L_{CYt}) *and* $\mathfrak{G}$ =$\bar{G}_1$,...,$\bar{G}_{t-1}$. *Assume that the attacker has access to* $\mathfrak{G}$ *and* $\bar{G}_t$.
$\bar{G}_t$ *satisfies the requirements* $\boldsymbol{RQ}_1$, $\boldsymbol{RQ}_2$ *and* $\boldsymbol{RQ}_3$ *if:*

1. *the grouping of new nodes together in classes of size at least k, satisfies the adding nodes procedure.*
2. *the old nodes deletion satisfies the Safely deleting procedure.*
3. *the removal of the violation of the condition* $\boldsymbol{SPa}_3$ *satisfies property 2.*

5.5. SNPP-Sequential Releases (SNPPnR) algorithm

We propose a greedy algorithm (algorithm 4) that permits publishing sequential releases of the same social network over time. This algorithm ensures that the requirements $\mathbf{RQ}_1$, $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$ are preserved. In this algorithm, we need only to keep the last original and published bipartite graph and

the set of nodes that their deletion is delayed in the previous releases when preparing the current data to be published. First, this algorithm calls the Safely deleting procedure. This procedure permits deleting nodes form the published bipartite graph without causing privacy leak. For each old node u that should be deleted (u $\in$ G_{ti-1} and u $\notin$ G_{ti}), Safely deleting searches a new node, T (T $\in$ G_{ti} and T $\notin$ G_{ti-1}), that has the same attributes list values and the same type as u. If such node is found, the Safely deleting algorithm deletes the old node and replaces it with the new one. In the opposite case (where the node T cannot be found), the Safely deleting algorithm delays the deletion of the old node, lines 2-3. Then, the algorithm SNPPnR calls repeatedly the procedure Safely permuting, see algorithm 3, to remove the violation of the condition $\mathbf{SPa}_3$, lines 4-7. Finally, the algorithm SNPPnR groups the new nodes. If the number of new nodes is less than k it delays their publication by suppressing them from the current set of nodes (W_{ti} ,V_{ti}) in order to be considered in the next releases, lines 8-13. Else it calls the procedure Safely partitioning, see algorithm 2, to group these nodes in classes of size at least k under the *safely partitioning condition*. In order to create classes under the condition $\mathbf{SPe}_1$ i.e. classes that share at least $\frac{2}{k}$ attributes lists values, we order the nodes by their attributes list values frequency. We then modify the utility cost in algorithm 2, by taking the node's utility cost equals to the inverse of the frequency of this node's attributes list value in the bipartite graph (i.e. N.cost $=\frac{1}{N.freq+1}$, where $N.freq$ is the number of nodes that have the same attributes list values as N), lines 14-23. Indeed, this allows creating classes that share a large number of attributes lists values, as we take each time the node that has the greatest attributes list values frequency, not yet grouped, and we create a new class C containing this node. Then we add repeatedly nodes to C until its size reaches k, such as the next node that will be added to C is the node that has the greatest attributes list values frequency, among other ones, that satisfy the *safely partitioning condition*.

5.6. Complexity of the algorithm SNPPnR

The algorithm SNPPnR cost equals to the cost of deleting old nodes plus the cost of removing the violations of the *safely partitioning condition* to which we add the cost of grouping new nodes.

The cost of grouping new nodes:

The cost of grouping new nodes is $O(N_{new}^2)$, where N_{new} is the number of new nodes. It can reach $O(N_{new})$, if we remove the utility cost function (see

section 4.4).

The cost of deleting old nodes:

Let's consider "deleting node as an elementary operation". The cost of deleting old nodes is:

- In the best case: the first new node tested has the same attributes lists values as the node that should be deleted. So, the cost of deleting nodes, in this case, is $O(N_{old})$; where N_{old} is the number of old nodes to be deleted.

- In the worst case: there is no new node with the same attributes lists values as the node to be deleted. So, for each user node to be deleted, we should browse all new user nodes. So, we need $N_{user-old} * N_{user-new}$ operations to delete user nodes; Where $N_{user-old}$ is the number of old user nodes to be deleted and $N_{user-new}$ is the number of new user nodes. Similarly, we need $N_{group-old} * N_{proup-new}$ operations for deleting group nodes; Where $N_{group-old}$ is the number of old group nodes to be deleted and $N_{group-new}$ is the number of new group nodes. So, in total, we need $N_{user-old} * N_{user_new} + N_{group-old} * N_{group-new}$ operations to delete group and user nodes. In this case, the cost of deleting nodes is less than $O(N_{old} * N_{new})$; where $N_{old} = N_{user-old} + N_{group-old}$ is the number of old nodes (user and group) to be deleted, and $N_{new} = N_{user-new} + N_{group-new}$ is the number of new nodes.

The cost of removing the violations of the *safely partitioning condition*:

Let's consider "removing a *safely partitioning condition* violation" as an elementary operation.

So as to suppress the violation of *safely partitioning condition*, the Safely permuting algorithm tries: to find a group class, a user class, to create a group class, a user class or a noise group class (see algorithm 3), consequently the cost of removing the violations of the *safely partitioning condition* is:

- In the best case, the first group class tested can remove the violation of the *safely partitioning condition*. So, in this case, the cost of removing the violations of the *safely partitioning condition* is $O(N_{violation})$. Where $N_{violation}$ is the number of violation of the *safely partitioning condition*.

- In the worst case, neither old nor new group/user classes can remove the violations of the *safely partitioning condition*. So, in this case, we create a noise group class. The cost of removing the violations of the *safely partitioning condition* is $O(N_{violation} * (|Old_{test}| + T_{Cnew} + T_{Cnoise}))$. Where $N_{violation}$ is the number of violations of the *safely partitioning condition*, $|Old_{test}|$ is the number of old group or user classes that are tested but that cannot remove the violation of the *safely partitioning condition*, T_{Cnew} is the time to test if a new group or user class can be created to remove the violation of the *safely partitioning condition*, and T_{Cnoise} is the time for creating a noise group class that removes the violation of the *safely partitioning condition*.

The results in appendix B show that the utility cost function has a low impact on the utility accuracy (because the utility accuracy depends more on the data itself). Thus, it is not aberrant to remove the utility cost function from the SNPPnR algorithm, which allows us to enhance its complexity. Indeed, this enhances the cost of grouping new nodes to be $O(N_{new})$. So, the SNPPnR algorithm without utility cost function is better suited for large social networks as it is faster. In section 6.11, we study the utility of the algorithm SNPPnR without utility cost function using large social networks.

6. EXPERIMENTS: DATA UTILITY AND PRIVACY MEASUREMENTS

While previous sections are focused on privacy preservation, Section 6 proves the utility of the published data and gives practical measurement of the privacy.

6.1. Utility evaluation

For the labeled graphs, people use the change of certain queries' results to measure the utility [15][16][17]. In this paper, two kinds of queries are used to evaluate the utility of our solution:

- Attribute properties on one side only (Type 1): This kind of queries takes into account only the attributes properties of one type of nodes. E.g. find the average degree of lawyers' users.

- Attributes properties on both sides (Type 2): the second kind of queries takes into account the attributes properties of the two types of nodes,

user and group. E.g. find the number of lawyers' users who are members of groups whose topic is politics.

6.2. Querying anonymized data

The result of our anonymization technique is a bipartite graph which masks the true mapping between nodes and attributes list values. It associates to each node a set of possible attributes list values, and publishes only the number of edges between classes. Thus, to answer the two types of queries, introduced in section 6.1, we can use sampling consistent graph techniques [17] [15]. These techniques allow the data miner to randomly sample a graph which is consistent with the published data (i.e. assign attributes list values and edges to nodes for which the assignment is consistent with the anonymized graph). Then, we perform analysis. Therefore, the query can be evaluated over the sampled graph. An "expected" answer can be evaluated by computing the average of query results on several consistent sampling graphs. The utility of the published data is measured using the equation 2 (see Definition 2) and by taking the "expected" answer as answer($q_{\bar{G}}$).

6.3. Experimental model

To evaluate the utility of our SNPPnR algorithm, we adapt the solution proposed in [13] to our SNPP problem and then we compare our SNPPnR algorithm to the adapted solution. We also compare the SNPPnR algorithm with the SNPP1R algorithm.
The solution proposed in [13] aims to preserve the privacy of the relationships between users (user-user). First, it uses the prediction links algorithms to predict the evolution of these relationships. Then, the solution proposed in [13] partitions new nodes together, using this prediction, such as this partitioning remains safe in the future releases. As our interest is for the affiliation relationship between users and groups, we cannot use these algorithms to predict the affiliation relationship evolution. So, for the sake of simplicity, we group the new nodes together without using the prediction algorithm. We notice that this solution does not take into account the case where several nodes share the same attributes list values. To ensure the requirements $\mathbf{RQ}_1$, $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$, we use our *safely partitioning condition* to group new nodes. We also notice that the case of deleting nodes is not considered, thus we use our Safely deleting procedure to remove the nodes from the social network.
The algorithms SNPPnR, SNPP1R and the algorithm of the adapted solution

(A5), proposed in [13], were implemented in Python using NetworkX. NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and function of complex networks.
We used the random data generator to generate data sets, i.e. bipartite graphs. In our experiments, we used the data that are generated randomly. Consequently, our results are valid for any given social network that satisfies the sparse property. The first original social network bipartite graph has 1000 nodes, and 3660 edges. To generate the next bipartite graphs, we take the previous bipartite graph and we randomly remove and add nodes and edges, so that the number of nodes and edges raises from one release to another by rate between 10% and 20% (these rates are observed in the data set used in [29]).
In the experiments, for the sake of simplicity, we consider two users' attributes, i.e. user's attributes list = (age, job), and two groups' attributes, i.e. group's attributes list = (topic, subtopic).
We use the algorithm SNPPnR, the algorithm SNPP1R, and the adapted solution, A5, proposed in [13] to publish sequential releases based on the first original bipartite graph.

6.4. The data set

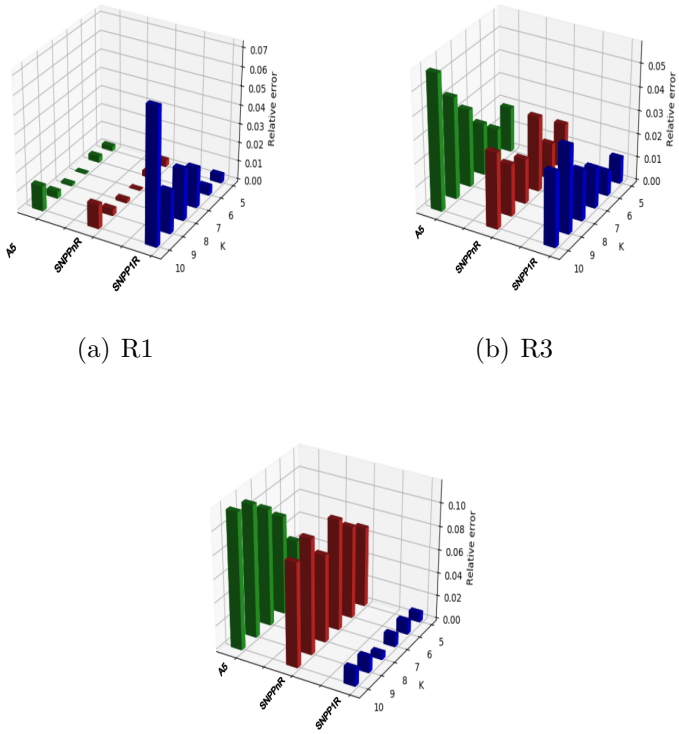
Table 2 presents the number of edges and nodes in each release.

Table 2: Bipartite Graphs in different releases

Releases	**Nodes' number**	**Edges' number**
R1	1000	3660
R3	1330	4700
R5	1710	6340

6.5. Utility accuracy

Figures 8 and 9 show the average relative errors of type1 and type2 queries for releases R1, R3 and R5, respectively. Each point in the figures is the expected answer of type1 and type2 queries, on the corresponding anonymous bipartite graph of 6 privacy levels using the algorithms SNPP1R, A5 and SNPPnR. For each anonymous bipartite graph and for each privacy level,

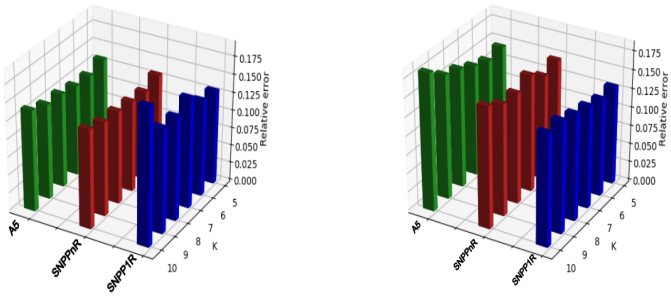


(a) R1

(b) R3

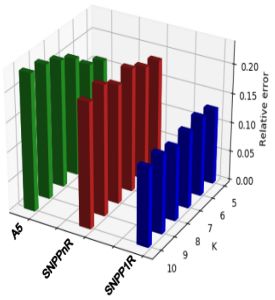
(c) R5

Figure 8: Relative error for type1. **a**: Average relative errors (type1) of the release R1, **b**: Average relative errors (type1) of the release R3, **c**: Average relative errors (type1) of the release R5.



(a) R1

(b) R3



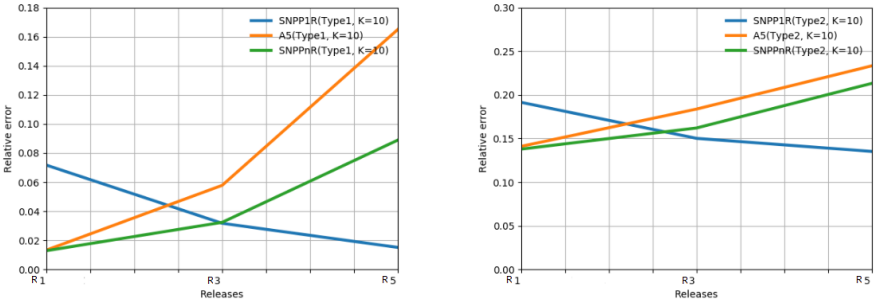
(c) R5

Figure 9: Relative error for type2. **a**: Average relative errors (type2) of the release R1, **b**: Average relative errors (type2) of the release R3, **c**: Average relative errors (type2) of the release R5.

we generate 20 consistent sampling bipartite graphs to get the "expected" result. We choose to use 20 consistent sampling bipartite graphs because this guarantees the reproducibility of the results . In fact, we calculated the margin of error,of the 20 consistent bipartite graphs, using a confidence interval with 99% and we obtained a margin of error of 1.5% for type1 queries and 3.7% for type2 queries. The average relative query error, type2, obtained by our solution, algorithm SNPPnR, ranging between 12% and 22% is in the same range as existing solutions based on the safe k-grouping concept [15]. From the results, we can observe that there is no significant difference on the average relative error between the anonymous bipartite graphs, produced by the algorithm SNPPnR, and the corresponding anonymous bipartite graphs produced, by algorithm SNPP1R, and the ones, produced by adapted solution A5. One reason of the little variation between them is the random characteristics when sampling consistent bipartite graphs. Another reason is the effect of the noise nodes added by the algorithm SNPPnR. The algorithm SNPPnR preserves the utility because it adds noise nodes as the number of edges increases, therefore adding noise nodes may degrade the utility; while using a bipartite graph, which contains a large number of edges, allows answering queries with more accuracy. However we notice that (especially in Figure 8.c) the algorithm SNPPnR and A5 give an average relative error more important than the algorithm SNPP1R, one reason is that the utility cost function, used in the algorithm SNPP1R to group the nodes, preserves the utility better than the one used in the algorithm SNPPnR and A5. The other reason is that the algorithm SNPP1R anonymizes each release independently, unlike the algorithms A5 and SNPPnR that consider the previous releases, when anonymizing the current one.

We also find that the average relative error increases with k, which satisfy the intuition, since sampling consistent bipartite graphs from an anonymous one with large k, leads to more significant error than the one with small k. However, at several points, such as the point $k = 6$ in Figure 9.b, the average relative error suddenly decreases a little. One reason is the algorithm random characteristics, used for sampling consistent bipartite graphs, which might give better result for a large k.

Figures 10.a and 10.b show the average relative error of type1 and type2 queries, respectively, over the different releases and by taking $k = 10$. The results show that the average relative error for algorithms A5 and SNPPnR increases with the number of releases. The reason is the algorithms A5 and SNPPnR produce the current release based on the previous one. The other



(a) Relative error for type1 (b) Relative error for type2

Figure 10: Relative error for different releases. **a**: Average relative error of type 1 queries over the different releases (R1, R3 and R5), **b**: Average relative error of type 2 queries over the different releases (R1, R3 and R5).

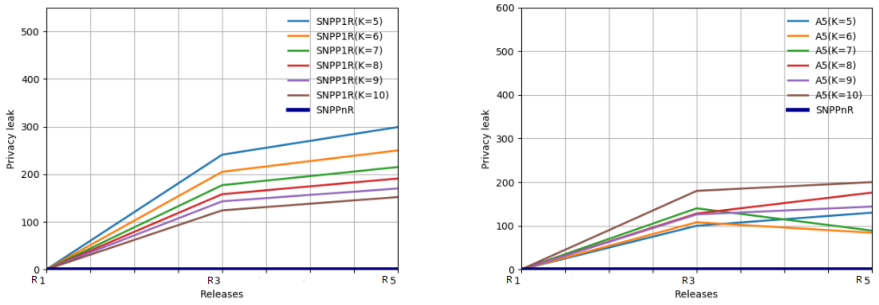
reason is the effect of the noise nodes created by the algorithm SNPPnR to ensure the privacy. However, we notice that the average relative error produced by the algorithm SNPP1R does not increase with the number of releases, because the algorithm SNPP1R anonymizes each release independently (without taking into consideration the previous releases).

6.6. Privacy leakage evaluation

We measure the amount of information that the attacker can obtain by comparing the different releases (R1, R3,and R5), produced by the algorithm SNPP1R and A5.
We measure the privacy leak, that the algorithm SNPP1R generates by calculating the number of non safe operations: NSafe defined as follows:

$$Nsafe = NSafePermut + NSafeNew + NSafeDelet \tag{3}$$

Where $NSafePermut$ is the number of permutations carried out without respecting the *safely permuting condition*, $NSafeNew$ represents the number of classes at $\bar{G}_{T-1}$ that are modified in $\bar{G}_T$ because of grouping new nodes with the old ones and $NSafeDelet$ is the number of removed nodes and replaced by other ones that don't have the same attributes list values.
We also measure the privacy leak engendered by the adapted solution proposed in [13]. As in the adapted solution proposed in [13], we have used our



(a) SNPP1R (b) A5

Figure 11: Privacy leak. **a**: Privacy leak engendered by the algorithm SNPP1R, **b**: Privacy leak engendered by the algorithm A5.

Safely deleting procedure to remove the old nodes. Therefore the only privacy risk is the violation of the condition $\mathbf{SPa}_3$ between classes. We measure the privacy leak named, Vleakage, as follows:

$$Vleakage = Vclasses * 2k \tag{4}$$

Where $Vclasses$ is the number of classes that violate the condition $\mathbf{SPa}_3$ in the published social network.

Figures 11.a and 11.b show the privacy leakage engendered by algorithm SNPP1R and A5, respectively. The results show that the privacy leak, engendered by the adapted solution A5 and the algorithm SNPP1R, increases with releases which satisfy the intuition, since the bipartite graph evolves with time. So the bipartite graph at T5 is more different than the bipartite graph at T3. However, with $k = 7$ and $k = 6$, we notice that the privacy leakage engendered by the algorithm A5 decreases with releases. One reason, is that the grouping of nodes in classes in the release R4 resists better to the social networks evolution (the edges adding) in the R5. Unlike the grouping of nodes in the release R2 that does not resist well to the social networks evolution in the R3.

Figure 11.a shows also that the privacy leak engendered by the algorithm SNPP1R decreases with k, because large k allows less non safe operations, especially less violation of the *safely permuting condition*, as large k allows large number of edges with no need of safe permutations.

Figure 11.b shows that the privacy leak engendered by the adapted solution

A5 increases with k, because the privacy leakage measures the number of nodes (user and group) in each class their privacy is affected. As the size of classes increases with k, then the number of nodes their privacy is affected increases. However, at several points, such as the point $k = 9$, the privacy leakage decreases suddenly a little. The reason, is that the grouping of nodes in classes, in the previous releases, with large k resists better to the social networks evolution (the edges adding) than the nodes grouping with small k.
Notice that the algorithm SNPPnR depicts no privacy leakage due to comparison of different releases, as it proved in section 5.

6.7. SNPPnR privacy evaluation measurement

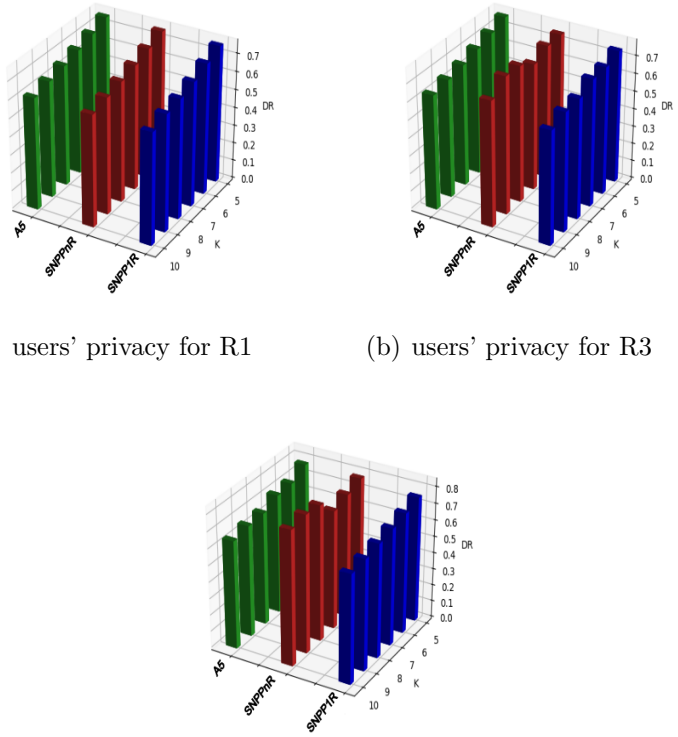
This subsection evaluates the privacy of the current anonymous data, without considering the information leakage, obtained by examining the previous published data. So, we measure the resistance of the current anonymous graph against the de-anonymization attacks by using the Discrimination Rate (DR) metric proposed by Sondeck et.al in [27] and defined by:

$$DR_X(Y) = 1 - \frac{H(X/Y)}{H(X)} \tag{5}$$

Where H(X) is the entropy of X and H(X/Y) is the conditional entropy.
The $\mathrm{DR}_X(\mathrm{Y})$ measures the capability of the attribute Y to identify the attribute X. It is scaled between 0 and 1. The greater the $\mathrm{DR}_X(\mathrm{Y})$, the greater the identification, and lower the privacy. For example, if $\mathrm{DR}_X(\mathrm{Y})$ equals to 1, it means that the values of the attribute Y permits to completely identify the values of the attribute X; hence the privacy of X is completely disclosed. We use the DR metric to measure the real users' privacy, groups' privacy and edges' privacy of the three solutions, SNPPnR, SNPP1R and A5. So, to measure the:

- users' privacy: we measure $\mathrm{DR}_X(\mathrm{Y})$ where X is the user node identifier and Y is the attributes list of the user node. We measure the identity user de-anonymization attack (as we measure the capability of the values of the attributes lists to identify the user node).

- groups' privacy: we measure $\mathrm{DR}_X(\mathrm{Y})$ where X is the group node identifier and Y is the attributes list of the group node. We measure the identity group de-anonymization attack (as we measure the capability of the values of the attributes lists to identify the group node).



(a) users' privacy for R1

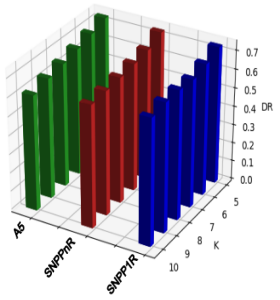
(b) users' privacy for R3

(c) users' privacy for R5

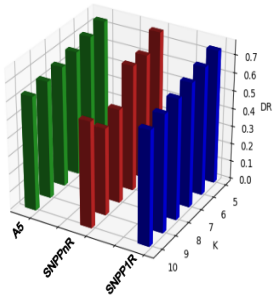
Figure 12: DR metric for users' privacy. **a**: DR metric of the release R1, **b**: DR metric of the release R3, **c**: DR metric of the release R5.

- edges' privacy: we measure $\mathrm{DR}_X(\mathrm{Y})$ where X is the attributes list of the group node and Y is the attributes list of the user node. We measure the link de-anonymization attack (as we measure the capability of the attributes lists values of the user nodes to identify the attributes lists values of the group nodes).

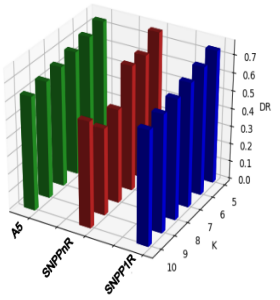
Figures 12, 13 and 14 show the users' privacy, the groups' privacy and the edges' privacy of the bipartite graphs R1, R3 and R5, produced by the algorithm SNPP1R, SNPPnR and A5 respectively. the DR metric of our solution, SNPPnR, which ranges between 41% and 86% is in the same range as the 15-anonymity instantiation and 10-anonymity [27]. From the results, we no-



(a) groups' privacy for R1

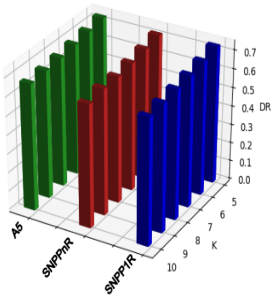


(b) groups' privacy for R3

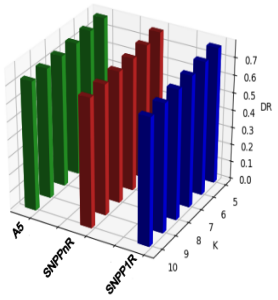


(c) groups' privacy for R5

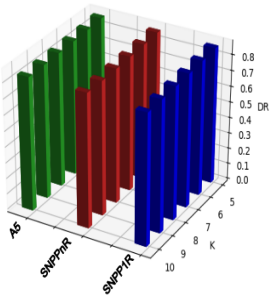
Figure 13: DR metric for groups' privacy. **a**: DR metric of the release R1, **b**: DR metric of the release R3, **c**: DR metric of the release R5.



(a) edges' privacy for R1



(b) edges' privacy for R3



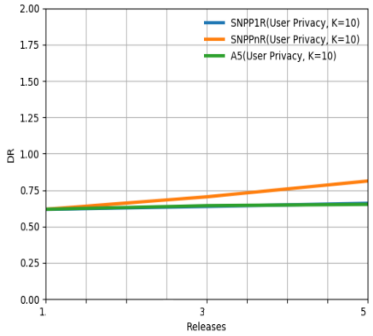
(c) edges' privacy for R5

Figure 14: DR metric for edges' privacy. **a**: DR metric of the release R1, **b**: DR metric of the release R3, **c**: DR metric of the release R5.

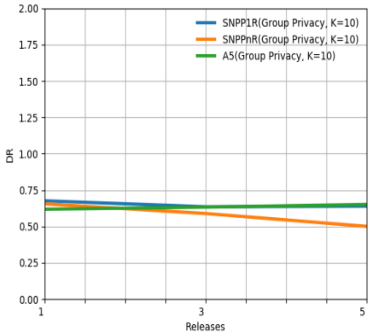
tice that the DR metric decreases with the increasing of k, which satisfies the intuition, as large k leads to more privacy. However, in Figures 12 and 13, we notice that the DR metric for users' and groups' privacy knows little increase with k (e.g. Figure 13.b, $k = 10$). The reason is the grouping of nodes in classes with large k preserves better the utility which leads to less privacy. The other reason is with large k, the algorithm SNPPnR creates less noise groups which lead to less privacy.

The results (Figures 12, 14, and 13.a) show also that there is no significant difference between the privacy level, offered by our solution, SNPPnR, and the others (SNPP1R and A5). The reason, is that we have used the same anonymization technique in the three algorithms. However in Figures 13.b and 13.c, that represent the groups' privacy in the release R3 and R5, respectively, we notice that our solution the algorithm, SNPPnR, outperforms the algorithms SNPP1R and A5. The reason is that the algorithm SNPPnR creates a noise groups to ensure the privacy in sequential releases (see the algorithm 3). As the groups' privacy is $\leq \frac{1}{\lambda_y k}$ (see $\mathbf{RQ}_2$) and as the created noise groups have the same attributes lists values like the real groups. Then, the average values of λ_y, in the anonymous bipartite graphs, created with the algorithm SNNPnR is larger than those created by the algorithm A5 and SNPP1R. So, we get a better groups' privacy.

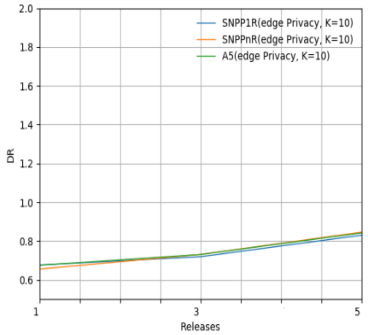
Figures 15.a, 15.b and 15.c show the users' privacy, the groups' privacy and the edges' privacy, over the different releases and by taking $k = 10$. Figures 15.a and 15.b show also that the DR metric (for users' privacy and groups' privacy), of the algorithms A5 and SNPP1R, does not vary a lot with the number of releases (R1, R3 and R5). One reason is that with the algorithm A5 and SNPP1R, we do not modify the number of nodes (groups and users). Figure 15.a shows also that the DR metric (for users' privacy) of the algorithm SNPPnR increases a little across releases. The reason is that the algorithm SNPPnR, when creating classes, groups similar user nodes in the same class which leads to less privacy. The Figure 15.b shows that the DR metric (for groups privacy)of the algorithm SNPPnR decreases with releases (which lead to more privacy). One reason is the effect of the noise groups, created by the algorithm SNPPnR to remove the violation of the condition $\mathbf{SPa}_3$. Figure 15.c shows that the DR metric for edges' privacy knows little increase with the number of releases. The reason is the evolution of the social networks, i.e. the bipartite graph in R5 contains more edges and nodes than the ones in R3 and R1. This leads to less privacy.



(a) users' privacy for k=10

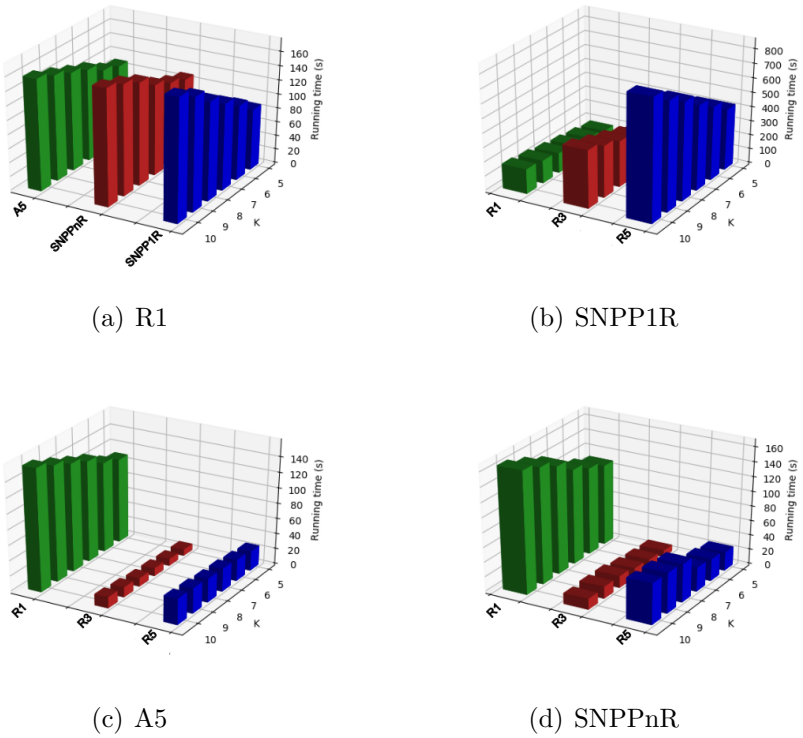


(b) groups' privacy for k=10



(c) edges' privacy for k=10

Figure 15: DR metric over the different releases. **a**: DR metric for users' privacy over the different releases (R1, R3 and R5), **b**: DR metric for groups' privacy over the different releases (R1, R3 and R5), **c**: DR metric for edges' privacy over the different releases (R1, R3 and R5).



(a) R1 (b) SNPP1R

(c) A5 (d) SNPPnR

Figure 16: Running time: **a**: Running time to produce the release R1, **b**: Running time of the algorithm SNPP1R, **c**: Running time of the algorithm A5, **d**: Running time of the algorithm SNPPnR.

6.8. Run time

Figure 16 shows the run time of the algorithms SNPP1R, SNPPnR and the adapted solution A5.

Figure 16.a shows the running time to produce the anonymous bipartite graph R1. From the results, we can see that the algorithm SNPP1R needs less running time than the algorithm SNPPnR and the algorithm A5. That is because the algorithms SNPPnR and A5 need to save the produced anonymous bipartite graph for the next releases. But in $k = 10$, we notice that the algorithm SNPPnR and A5 are faster than the algorithm SNPP1R. The reason is that the utility cost function used by the algorithm SNPP1R to group nodes needs more time than the one used by the algorithms SNPPnR

and A5. i.e. with large k, the algorithm SNPP1R needs to compare many nodes to choose the one which preserves well the utility. This needs more time.

Figures 16.b, 16.c and 16.d show the run time to produce the anonymous bipartite graphs R1, R3 and R5, using the algorithms SNPP1R, A5 and SNPPnR, respectively. From the results, we can notice that the running time of the algorithm SNPP1R increases with the releases which satisfy the intuition as the size of bipartite graphs increases with releases. The results show also that the running time needed by the algorithm SNPPnR and the algorithm A5 to produce the bipartite graphs R3 and R5 is shorter than the running time needed by the algorithm SNPP1R. For example for the point (R3, $k = 5$) the algorithm SNPPnR needs only 9.4 seconds and the algorithm A5 needs 8.3 seconds while the algorithm SNPP1R needs 204 seconds. The reason is that the algorithm SNPPnR and the algorithm A5 group only the new nodes while the algorithm SNPP1R groups all the nodes. We notice also that the algorithm A5 needs less time than the algorithm SNPPnR. The justification is that the algorithm A5 does not make any processing to remove the violation of the condition $\mathbf{SPa}_3$.

6.9. λ_x and λ_y impact on utility

Recall that $\lambda_x - 1$ represents the number of users' nodes that have the same attributes list values and which are published at the same time as node u_x and $\lambda_y - 1$ is the number of groups' nodes that share the same attributes list values and published at the same time as node u_y.

To study the impact of the values of parameters λ_x and λ_y on the utility, we generate two bipartite graphs, G1 and G2, with different values of λ_x and λ_y. We use the algorithm SNPPnR to produce the anonymous bipartite graphs. Table 3 gives the number of nodes, edges and the average values of λ_x and λ_y for each bipartite graph.

Figure 17 shows the average relative errors of type1 and type2 queries for bipartite graphs G1, and G2. Each point in the figure is the expected answer of type1 and type2 queries on the corresponding anonymous bipartite graph of 6 privacy levels, using the algorithm SNPPnR. The results show that the relative error in the bipartite graph G1 is larger than the one in the bipartite graph G2. The reason is that in the bipartite graph G2, the average values of λ_x and λ_y are larger than the ones in the bipartite graph G1, so, as we have large average values of λ_x and λ_y, in G2, we get better utility because we use a bipartite graph which contains several nodes that share the same

Table 3: BIPARTITE GRAPHS CHARACTERISTICS

Bipartite graphs	**Nodes' number**	**Edges' number**	**Average values of λ_x**	**Average values of λ_y**
G1	400	1670	1,05	1,1
G2	400	1670	3,29	3,30

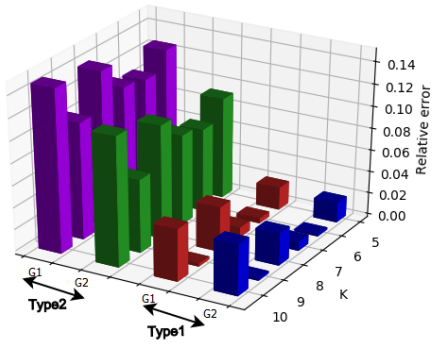
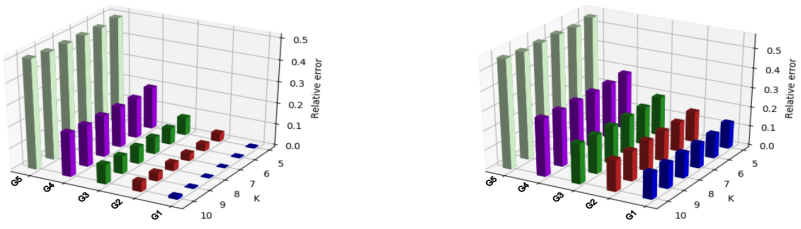


Figure 17: λ_x and λ_y impact on utility.

attributes list values.

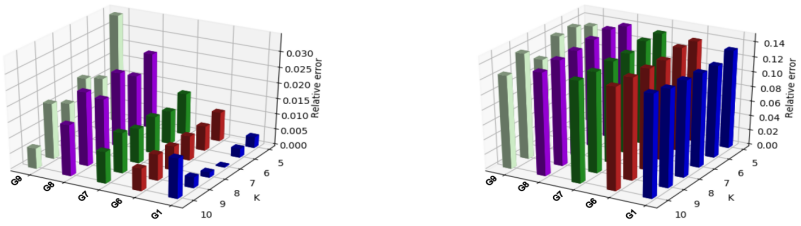
6.10. Nodes/edges addition/deletion impact on utility

Recall that the algorithm SNPPnR delays the publication of new nodes if the number of these new nodes is less than the privacy level, k. The algorithm SNPPnR delays the suppression of nodes if no new ones can replace them (new nodes that have the same attributes list values as those to be deleted). Also, the algorithm adds noise nodes to avoid the violation of the *safely partitioning condition* (due to the edges addition between old classes).
As the number of new nodes in large social networks is usually greater than k, this section studies the impact on the utility of the delayed deleted nodes and that of the noise nodes due to the edges addition between old classes.
To study the impact of delayed deleted nodes, we generate a graph G1, and we produce the graphs G2, G3, G4, and G5 by adding/deleting nodes/edges from G1. 5% nodes are deleted from G1 to G2, 10% nodes from G1 to G3, and 20% nodes from G1 to G4, and 50% nodes from G1 to G5. the characteristics of graphs G1, G2, G3, G4, and G5 are shown in Table 4.
Figure 18 shows the average relative errors of type1 and type2 queries for bipartite graphs G1, G2, G3, G4 and G5. Each point in the figure is the expected answer of type1 and type2 queries on the corresponding anonymous bipartite graph of 6 privacy levels using the algorithm SNPPnR. The results show that the relative error type1 and type2 increases with the number of deleted nodes (5% in G2, 10% in G3, 20% in G4, 50% in G5) which satisfy the intuition. The nodes that should be deleted remain in the next graphs (G2, G3, G4, G5), until the arrival of new nodes that can replace them. These delayed deleted nodes are considered as noise data, thus leading to less utility. Figure 18 shows that the relative error (type1 and type2) in graph G4 (where there are 20% deleted nodes) ranges between 19% and 29%, which is in the same range as existing solutions based on the safe k-grouping concept [15]. As a result, our solution can preserve the utility even though the number of deleted nodes from one release to the next one is higher (20% deleted nodes).
To study the impact of noise nodes, due to adding new edges between old classes), we generate a graph G1, and we produce the graphs G6, G7, G8, and G9 by adding/deleting nodes/edges from G1 as follows: 5% edges are added between old classes in G1 to produce G6, 10% edges between old classes in G1 to produce G7, 20% edges between old classes in G1 to produce G8, and 50% edges between old classes in G1 to produce G9. The characteristics of



(a) Relative error for type 1 (b) Relative error for type 2

Figure 18: The nodes deletion impact on the utility.



(a) Relative error for type 1 (b) Relative error for type 2

Figure 19: The edges addition impact on the utility

Table 4: BIPARTITE GRAPHS CHARACTERISTICS

Releases	**Nodes' number**	**Edges' number**
G1	1000	3660
G2	1170	3690
G3	1110	3280
G4	980	2620
G5	620	1020

Table 5: BIPARTITE GRAPHS CHARACTERISTICS

Releases	**Nodes' number**	**Edges' number**
G1	1000	3660
G6	1180	4020
G7	1180	4200
G8	1180	4600
G9	1180	4980

graphs G1, G6, G7, G8, and G9 are shown in Table 5.
Figure 19 shows the average relative errors of type1 and type2 queries for bipartite graphs G1, G6, G7, G8 and G9. Each point in the figure is the expected answer of type1 and type2 queries on the corresponding anonymous bipartite graph of 6 privacy levels, using the algorithm SNPPnR. The results show that the relative error type1 and type2 (especially type 2)does not vary between graphs (i.e. with the edges addition). The reason is that the algorithm SNPPnR adds noise nodes proportionally to the number of added edges (to suppress the violation of the *safely partitioning condition*). As a consequence, although the addition of noise nodes may decrease the utility, the use of graphs which contain a big number of edges leads to better utility (i.e. the big number of edges hides the impact of noise nodes). As such, our solution can preserve the utility even though the number of added edges from one release to the next one is high.
As a consequence, the SNPPnR algorithm preserves the utility in the case of high dynamic social networks.

6.11. Big social networks

Recall that the privacy preserving property of our solution, SNPPnR, is proved in section 5.
This subsection studies the utility, offered by our solution in the case of large social networks.
We used the random data generator to generate data sets. The first original social network bipartite graph has 101770 nodes, and 309980 edges. To generate the next bipartite graphs for the next releases, we take the previous bipartite graph and we randomly remove and add nodes and edges, so that the number of nodes and edges raises from one release to another by rate

between 10% and 20% [29] (see Table 6).
As these graphs are large, to generate the anonymous bipartite graphs, we use a modified version of the algorithm SNPPnR by removing the utility cost function from the algorithm SNPPnR for the following reasons:

- Tests on social networks of reasonable sizes show that taking into account the utility cost function does not bring much for the utility accuracy (see Appendix B).

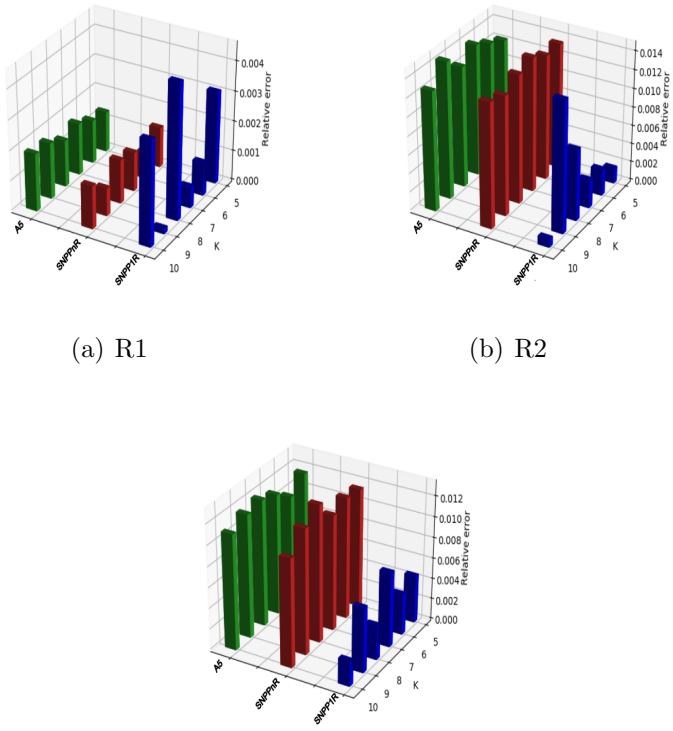
- The running time of the algorithm SNPPnR without the utility cost function is more efficient (see section 5.6). For example, the algorithm SNPPnR without the utility cost function requires only 16 seconds to anonymize a graph with 5000 nodes and 16900 edges while the algorithm SNPPnR with the utility cost function requires 278 minutes. Also, the algorithm SNPPnR without the utility cost function requires only 131 minutes to anonymize a graph with 101770 nodes and 309980 edges (see Appendix C).

As such, for testing large social networks, we decide to remove the utility cost function from the algorithm SNPPnR. We also use modified versions of the algorithms A5 and SNPP1R (Without utility cost function) to evaluate our solution (SNPPnR without utility cost function).

Table 6: BIPARTITE GRAPHS CHARACTERISTICS

Releases	**Nodes' number**	**Edges' number**
R1	101770	309980
R2	120410	368410
R3	145380	448670

Figures 20 and 21 show the average relative errors of type1 and type2 queries for releases R1, R2 and R3, respectively. Each point in the figures is the expected answer of type1 and type2 queries, on the corresponding anonymous bipartite graph of 6 privacy levels using the modified versions of the algorithms SNPP1R, A5 and SNPPnR (algorithms without the utility cost).
We can notice that the obtained results share the same logic with those obtained in small social networks using the algorithms (SNPPnR, SNPP1, A5)

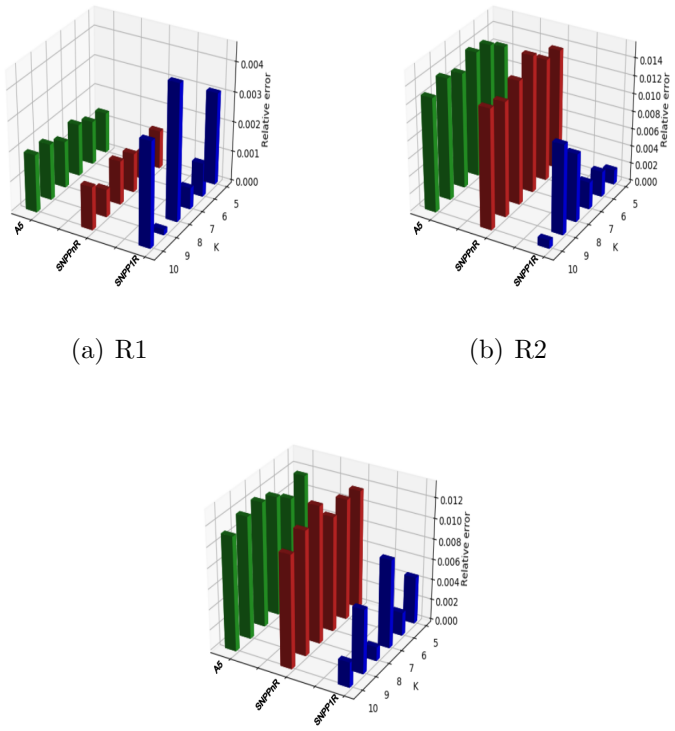


(a) R1

(b) R2

(c) R3

Figure 20: Average relative error for type 1. **a**: for release R1, **b**: for release R2, **c**: for release R3.



(a) R1

(b) R2

(c) R3

captionAverage relative error for type 2. **a**: for release R1, **b**: for release R2, **c**: for release R3.

with utility cost see section 6.5. Indeed, the results (especially in figures 20.b, 20.c, 21.b and 21.c) show that the modified version of the algorithm SNPP1R outperforms the modified versions of the algorithms A5 and SNPPnR. The reason is that the modified version of the algorithm SNPP1R anonymizes each release independently, unlike the modified versions of the algorithms A5 and SNPPnR that consider the previous releases, when anonymizing the current one. From the results, we notice also that the modified version of the algorithm A5 outperforms a little bit the modified version of algorithm SNPPnR. The reason is the effect of the noise nodes created by the modified algorithm SNPPnR to ensure the privacy. We notice that the effect of noise nodes is more important in big social networks than in small social networks, which can be explained by the high evolution (nodes/edges addition/deletion) in big social networks.
The results show that the relative error for type1 and type2, of the modified version of the algorithm SNPPnR, ranges between 0.05% and 2%, against the range 0.5% and 22% for small social networks using the SNPPnR with utility cost (see section 6.5). We can conclude that the modified version of the algorithm SNPPnR preserves well the utility in the case of large social networks despite the removal of the utility cost function which was originally introduced for preserving the utility within classes. The reason is that, in large graphs, the massive number of edges leads to better utility, the other reason is that the frequency of attributes lists values is higher in large social networks, i.e. the high probability to find several nodes with same attributes list values leads to fewer errors when sampling consistent bipartite graphs, thus resulting in high utility. As a consequence, our solution provides high utility accuracy in the case of big social networks, especially when the data receivers are interested only in a small number of attributes list.

7. CONCLUSION

In this paper, we propose a privacy preserving technique for labeled bipartite social networks. This technique allows publishing sequential releases of the same social network while ensuring that the published data meet the privacy requirements and remain useful for data mining tasks. This technique is based on anonymization method that groups nodes in classes and masks the true mapping between nodes and attributes values. We prove the effectiveness of the proposed technique through several experiments. The latter demonstrated the privacy preserving of sequential releases while maintaining

high utility of the published bipartite graphs and a reasonable overhead in terms, of running time. We propose this solution to encourage social network data publishing and helping new services useful for the whole society to emerge, especially in health or education sectors.

Appendix A. safely permuting condition proof

We prove that the permutation between two nodes of the same type that belong to two different classes, under the *safely permuting condition*, satisfies the $\mathbf{RQ}_1$, $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$.

Proof that the safely permuting condition satisfies the requirements $\mathbf{RQ}_1$ and $\mathbf{RQ}_2$:

We prove that the probability that an attacker can re-identify the node, u_x, in the published bipartite graph that represents the targeted individual (the targeted group, respectively) is less than or equal to $\frac{1}{\lambda_x k}$ ($\frac{1}{\lambda_y k}$ respectively) .

Let suppose that:

At T_i, we have a published bipartite graph $\bar{G}_{Ti}$ =(X_{Ti},Y_{Ti}, $\bar{E}_{Ti}$, L_{CXTi}, L_{CYTi}) such as:

- A node u_x is in class ${C_x}^{Ti} \in X_{Ti}$.
- ${C_y}^{Ti} \in Y_{Ti}$ is class of groups.

At T_{i+n}, we have a published bipartite graph $\bar{G}_{Ti+n}$ such as:

- C_{z1}, C_{z2},..., $C_{zn} \in X_{Ti+n}$ are classes that satisfy, with ${C_x}^{Ti+n}$, the condition $\mathbf{SPe}_1$.
- There are (λ_x-1) nodes published, at the same time as u_x, having the same attributes list values like u_x, and belonging to (λ_x-1) different classes, (C_{x1},..., $C_{x\lambda_{x-1}}$) $\in X_{Ti+n}$, as required by the condition $\mathbf{SPa}_2$.
- There is a violation of the condition $\mathbf{SPa}_3$ between ${C_x}^{Ti+n}$ and ${C_y}^{Ti+n}$, so to remove this violation, we must permute, under the property 2, between nodes belonging to ${C_x}^{Ti+n}$ and C_{zi}.

We find two cases:

- The attacker assumes with probability $\frac{1}{2}$ that, the difference between $\bar{G}_{Ti+n}$ and $\bar{G}_{Ti}$ is due to the social network evolution, then in this case the probability that an attacker knows that the node u_x is the one

representing the targeted individual is: $P(u_x)= \frac{1}{2}* \frac{1}{\lambda_x}* \frac{1}{|C_x^{Ti+n}|}$ as the size of each
$C_{xi} \geq k$ then $P(u_x) \leq \frac{1}{2}*\frac{1}{\lambda_x k} < \frac{1}{\lambda_x k}$

- The attacker assumes, with probability $\frac{1}{2}$, that the difference between $\bar{G}_{Ti+n}$ and $\bar{G}_{Ti}$ is due to the suppression of the violation of the condition $\mathbf{SPa}_3$ between ${C_x}^{Ti+n}$ and ${C_y}^{Ti+n}$. So, she/he tries to find two nodes, u_1 and u_2, belonging to ${C_x}^{Ti+n}$ and C_{zi}, respectively, such as permuting these two nodes, u_1 and u_2, violates the condition $\mathbf{SPa}_3$ between ${C_x}^{Ti+n}$ and ${C_y}^{Ti+n}$ (or increases the number of edges between ${C_x}^{Ti+n}$ and ${C_y}^{Ti+n}$): so
  - If there is a node, u_k: $u_k \in C_{zi}$, such as u_k.attributes list values = u_x.attributes list values, and as:
    $|$ ${C_x}^{Ti+n}$.attributes lists values $\cap$ C_{zi}.attributes lists values $| \geq \frac{k}{2}$, then the probability that the permuted node u_1 is the targeted individual node, u_x, is less than or equal to $\frac{2}{k}$, so the probability that an attacker knows that the node u_x is the node representing the targeted individual is:
    $P(u_x) \leq \frac{1}{2}* \frac{1}{\lambda_x} *\frac{2}{k} \leq \frac{1}{\lambda_x k}$
  - If there is not a node, u_k: $u_k \in C_{zi}$ such as u_k.attributes list values = u_x.attributes list values, then the probability that an attacker knows that the node u_x is the targeted individual node is:
    $P(u_x) = \frac{1}{2}* (\frac{1}{\lambda_x}* \frac{1}{|C_x^{Ti+n}|-1})$.
    as the size of each $C_{xi} \geq k$ then
    $P(u_x) \leq \frac{1}{\lambda_x}* \frac{1}{2(k-1)} < \frac{1}{\lambda_x k}$ $\forall k \geq 2$ then $P(u_x) < \frac{1}{\lambda_x k}$

So the *safely permuting condition* guarantees that the requirement $\mathbf{RQ}_1$(users' privacy) is satisfied while allowing nodes permutation.
Similarly, we prove that the requirement $\mathbf{RQ}_2$ (groups' privacy) is guaranteed by the *safely permuting condition.*

Proof that the safely permuting condition satisfies the requirement $\mathbf{RQ}_3$:
We prove that the probability that an attacker re-identifies that a link exists between a node u_x, representing the targeted individual, and a node u_y, representing the targeted group, is less than or equal to $\frac{1}{k}$. Let suppose that:
At T_i, we have a published bipartite graph $\bar{G}_{Ti}$ =(X_{Ti},Y_{Ti}, $\bar{E}_{Ti}$, L_{CXTi}, L_{CYTi}) such as:

- A node u_x is in class ${C_x}^{Ti} \in X_{Ti}$.
- A node u_y is in class ${C_y}^{Ti} \in Y_{Ti}$.

At T_{i+n}, we have a published bipartite graph $\bar{G}_{Ti+n}$ such as:

- C_{z1}, C_{z2},..., $C_{zn} \in X_{Ti+n}$ are classes that satisfy, with ${C_x}^{Ti+n}$, the condition $\mathbf{SPe}_1$.
- There are (λ_x-1) nodes published, at the same time as u_x, having the same attributes list values like u_x, and belonging to (λ_x-1) different classes (C_{x1},..., $C_{x\lambda_{x-1}}$) $\in X_{Ti+n}$, as required by the condition $\mathbf{SPa}_2$.
- There are (λ_y-1) other nodes published at the same time as u_y, having the same attributes list values like u_y, belonging to (λ_y-1) different classes (C_{y1},..., $C_{y\lambda_{y-1}}$) $\in Y_{Ti+n}$ as required by the condition $\mathbf{SPa}_1$.
- all nodes in ${C_x}^{Ti+n}$,C_{x1},..., $C_{\lambda_{x-1}}$ have links with nodes in ${C_y}^{Ti+n}$, C_{y1},..., $C_{\lambda_{y-1}}$.
- There is a violation of the condition $\mathbf{SPa}_3$ between ${C_x}^{Ti+n}$ and ${C_y}^{Ti+n}$, so to remove this violation, we must permute, under the property 2, between nodes belonging to ${C_x}^{Ti+n}$ and C_{zi}.

We find two cases:

- The attacker assumes with probability $\frac{1}{2}$ that, the difference between $\bar{G}_{Ti+n}$ and $\bar{G}_{Ti}$ is due to the social network evolution, then in this case the probability that an attacker knows that a link, e, exists between node u_x and u_y is:
  $P(u_x, u_y, e) \leq \frac{1}{2} * \frac{1}{k} \leq \frac{1}{k}$: because the anonymization of the bipartite graph $\bar{G}_{Ti+n}$ is under the *safely partitioning condition*. So, the property 1 is verified i.e. the requirement $\mathbf{RQ}_3$ is satisfied (the probability of edge's identification is less than or equal to $\frac{1}{k}$).
- The attacker assumes, with probability $\frac{1}{2}$, that the difference between $\bar{G}_{Ti+n}$ and $\bar{G}_{Ti}$ is due to the suppression of the violation of the condition $\mathbf{SPa}_3$ between ${C_x}^{Ti+n}$ and ${C_y}^{Ti+n}$, so she/he tries to find two nodes, u_1 and u_2, belonging to ${C_x}^{Ti+n}$ and C_{zi}, respectively, such as permuting these two nodes, u_1 and u_2, violates the condition $\mathbf{SPa}_3$ between ${C_x}^{Ti+n}$ and ${C_y}^{Ti+n}$(or increases the number of edges between ${C_x}^{Ti+n}$ and ${C_y}^{Ti+n}$). In this case, the probability that an attacker knows that a link, e, exists between nodes u_x and u_y is:
  $P(u_x, u_y, e) = \frac{1}{2} * (\frac{1}{|C_x^{Ti+n}|+|C_{x1}|+....+|C_{x\lambda_{x-1}}|} * \frac{1}{|C_y^{Ti+n}|+|C_{y1}|+....+|C_{y\lambda_{y-1}}|}) * nbedge$

According to the condition $\mathbf{SPa}_3$, the number of edges between any two classes,C_{xi} and C_{yj}, must be $\leq k$ and as we permute two nodes between ${C_x}^{Ti+n}$ and C_{zi}, then in the worst case the replaced node in ${C_x}^{Ti+n}$ has k edges with ${C_y}^{Ti+n}$. Hence the number of edges between ${C_x}^{Ti+n}$ and ${C_y}^{Ti+n} \leq 2k$, so:

$nbedge \leq k(\lambda_x * \lambda_y + 1)$

Then: $P(u_x, u_y, e) \leq \frac{1}{2} * \frac{1}{\lambda_x k} * \frac{1}{\lambda_y k} * k(\lambda_x * \lambda_y + 1)$.

So: $P(u_x, u_y, e) \leq \frac{1}{2} * (\frac{1}{k} + \frac{1}{\lambda_x \lambda_y k})$, then $P(u_x, u_y, e) \leq \frac{1}{k}$

So the *safely permuting condition* guarantees the requirement $\mathbf{RQ}_3$(edges' privacy) while allowing nodes permutation.

Appendix B. The utility cost function impact on the utility

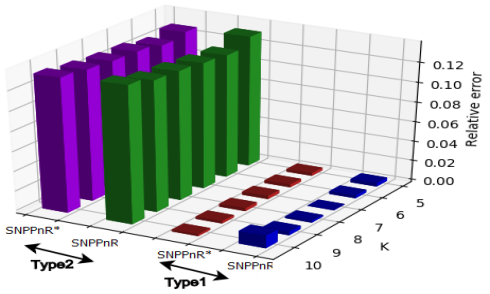


Figure B.21: The utility cost function impact on the utility accuracy

This section studies the impact of the utility cost function on the utility accuracy. So, we generate randomly a bipartite graph (G1) that contains 1000 nodes and 3360 edges and we use the algorithm SNPPnR and the algorithm SNPPnR without utility cost function (SNPPnR*) to produce the anonymous bipartite graph.

Figure 22.B shows the average relative errors of type1 and type2 queries for bipartite graph G1. Each point in the figure is the expected answer of type1 and type2 queries on the corresponding anonymous bipartite graph of 6 privacy levels, using the algorithm SNPPnR and the algorithm SNPPnR*. The results show that the utility cost function does not impact much on the utility as the relative error for type1 and type2 obtained by the algorithm SNPPnR

and SNPPnR*, is mainly similar (only a little variation). The reason is that the utility accuracy depends much more on the nature of the data (attributes lists values). Indeed, i) if in the graph, the attributes values frequency is high, then, even without considering the utility cost function, the probability of putting nodes that share a big number of attributes values in the same class is high. So, the algorithm SNPPnR* reaches a high utility accuracy. ii) if, in the graph, the attributes values frequency is low, then, even though the utility cost function is used, the utility accuracy will not enhance a lot, as finally, the algorithm SNPPnR can put small number of nodes that share attributes values together (as the attributes values frequency is low).

Appendix C. Running time of the algorithm SNPPnR without utility cost function

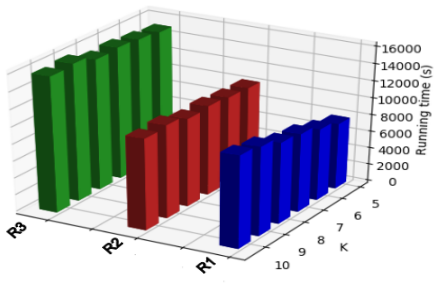


Figure C.22: the running time of the algorithm SNPPnR without utility cost function

This section studies the running time of the algorithm SNPPnR without utility cost function. So, we calculate the running time needed by the algorithm SNPPnR without utility cost function to produce the anonymous bipartite graphs of the original ones, presented in the section 6.11 (see Table 6).

Figure 23.C shows that the algorithm SNPPnR without utility cost function produces an anonymous bipartite graph that contains 101770 nodes and 309980 edges in only 131 minutes. Figure 22.C shows also that the running time increases with releases (R1, R2 and R3). The reason is that the bipartite graphs in R2 and R3 contain a big number of new nodes that should be grouped in new classes. The other reason is the running time needed

to remove the old nodes (nodes that should be deleted) and the running time to suppress the violation of the *safely partitioning condition* (i.e. in the large social networks, we need to test a big number of classes to suppress the violation of the *safely partitioning condition*).

References

[1] J.Abawajy, M. Izuan, H. Ninggal, and T. Herawan. "Privacy Preserving Social Network Data Publication", in IEEE Communications Surveys and Tutorials, 2016.

[2] R. Tackx , J.l. Guillaume, and F. Tarissan. "Revealing intricate properties of communities in the bipartite structure of online social networks", in IEEE Ninth International Conference on Research Challenges in Information Science (RCIS), 2015.

[3] M.Yuan, L.Chen, P.S.Yu, and T.Yu. "Protecting Sensitive Labels in Social Network Data Anonymization", in IEEE Transactions on Knowledge and Data Engineering, Volume 25, Issue 3 , pp. 633-647, 2013.

[4] B. Zhou and J. Pei. "Preserving privacy in social networks against neighborhood attacks", in IEEE 24th International Conference on Data Engineering (ICDE), pp. 506-515, 2008.

[5] Y. Wang, L. Xie, B. Zheng, and K. Lee. "High utility k-anonymization for social network publishing". in Knowledge and Information Systems, vol. 41, no. 3, pp. 697-725, 2014.

[6] R. Okada, C. Watanabe, and H. Kitagawa. "A k-anonymization algorithm on social network data that reduces distances between nodes", in Reliable Distributed Systems Workshops (SRDSW), 2014 IEEE 33rd International Symposium on . IEEE, pp. 7681, 2014.

[7] J.Cheng, A.W.Fu, and J.Liu, "K-isomorphism: privacy preserving network publication against structural attacks," in Pro-ceedings of the ACM International Conference on Management of Data (SIGMOD), pp. 459 470, 2010.

[8] Q. Liu, G. Wang; F. Li, S. Yang, and J. Wu, " Preserving Privacy with Probabilistic Indistinguishability in Weighted Social Networks", in IEEE

Transactions on Parallel and Distributed Systems, Volume: 28, Issue: 5, pp. 1417 - 1429, 2017.

[9] M. Hay, G. Miklau, D.Jensen, P. Weis, and S. Srivastava, "Anonymizing social networks," in Computer Science Department Faculty Publication Series. 180, pp. 1-17, 2007.

[10] X. Ying and X. Wu. "Randomizing Social Networks: a Spectrum Preserving Approach", in SIAM International Conference on Data Mining (SDM), pp. 739-750, 2008.

[11] P. Liu, L. Wang, and X. Li. "Randomized Perturbation for Privacy Preserving Social Network Data Publishing" , in the IEEE International Conference on Big Knowledge (ICBK), pp 208-2013, 2017.

[12] M. Hay, G. Miklau, D. Jensen, D. F. Towsley, and C. Li. "Resisting structural reidentification in anonymized social networks", in Very Large Database Journal, 2010.

[13] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. "Prediction Promotes Privacy In Dynamic Social Networks", in Workshop on Online Social Networks (WOSN), Boston, 2010.

[14] T. Tassa and D. Cohen. "Anonymization of centralized and distributed social networks by sequential clustering," in IEEE Transactions on Knowledge and Data Engineering, 2012.

[15] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. "Class-based graph anonymization for social network data", in VLDB , 2009.

[16] M. Yuan, L.Chen, and P.S. Yu. "Personalized Privacy Protection in Social Networks", in the VLDB Endowment, Vol. 4, 2011.

[17] G.Cormode , D.Srivastava ,T.Yu , and Q. Zhang, "Anonymizing bipartite graph data using safe groupings", in the VLDB Journal , 2010.

[18] H. Jiang, "A novel clustering-based anonymization approach for graph to achieve privacy preservation in social network", in International Conference on Advances in Mechanical Engineering and Industrial Informatics. Atlantis Press, 2015.

[19] F. Bonchi, A. Gionis, and T. Tassa. "Identity Obfuscation in Graphs Through the Information Theoretic Lens", in IEEE 24th International Conference on Data Engineering (ICDE), 2011.

[20] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa. "Injecting uncertainty in graphs for identity obfuscation," in Proceedings of VLDB Endowment, Volume 5, No 11, 1376-1387, 2012.

[21] H.H. Nguyen, A.Imine, and M. Rusinowitch. "Anonymizing Social Graphs via Uncertainty Semantics", in Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS), 2015.

[22] C. Borgs, J. T. Chayes and A. Smith. "Private Graphon Estimation for Sparse Graphs", in NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems, April 2015.

[23] J. Blocki, A. Blum, A. Datta, and O. Sheffet. "Differentially Private Data Analysis of Social Networks via Restricted Sensitivity", in Innovations in (Theoretical) Computer Science (ICS), 2013.

[24] Q.Wang, Y. Zhang, X.Lu , Z. Wang, Z. Qin, and K.Ren. "Real-time and Spatio-temporal Crowd-sourced Social Network Data Publishing with Differential Privacy", in IEEE Transactions on Dependable and Secure Computing, 2018.

[25] C. M. Fung, K. Wang, A. Wai-Chee Fu, and P.S. Yu. "Introduction to Privacy-Preserving Data Publishing Concepts and Techniques", in Chapman and Hall/CRC Data Mining and Knowledge Discovery Series, 2010.

[26] E.Zheleva and L.Getoor. "Privacy in social networks: A survey", in Springer Science+Business Media, LLC 2011.

[27] LP.Sondeck, M.Laurent, and V. Frey. "Discrimination rate: an attribute-centric metric to measure privacy", In: Annals of Telecommunications journal DOI:10.1007/s12243-017-0581-8, 2017.

[28] C.Dwork. "Differential privacy", in Automata, languages and programming, pp. 1-12. Springer Berlin Heidelberg, 2006.

[29] T.Wu, S.Yu, W.Liao, and C.Chang "Temporal Bipartite Projection and Link Prediction for Online Social Networks", in IEEE International Conference on Big Data, 2014.

[30] F.Yu, M.Chen, B.Yu, W.Li, L.Ma, and H.Gao. "Privacy preservation based on clustering perturbation algorithm for social network", in Journal Multimedia Tools and Applications. 2018.

[31] J.Yan, L.Zhang, Y.Tian, G.Wen, and J.Hu. "An Uncertain Graph Approach for Preserving Privacy in Social Networks Based on Important Nodes", in International Conference on Networking and Network Applications (NaNA). 2018.

[32] X.Xiao,and Y.Tao. "m-invariance: Towards privacy preserving republication of dynamic datasets", in Proceedings of the ACM SIGMOD Conference. ACM, New York. 2007.