



HAL
open science

Privacy preservation for social networks sequential publishing

Safia Bourahla, Maryline Laurent, Yacine Challal

► **To cite this version:**

Safia Bourahla, Maryline Laurent, Yacine Challal. Privacy preservation for social networks sequential publishing. *Computer Networks*, 2020, 170, pp.107106-1:107106-25. 10.1016/j.comnet.2020.107106 . hal-02921464

HAL Id: hal-02921464

<https://hal.science/hal-02921464>

Submitted on 25 Aug 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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 Systèmes, 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.

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

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

59 The rest of the paper is organized as follows: Section 2 reviews related works.
60 Section 3 presents the data model, attacker's knowledge, the privacy and
61 the utility requirements and the social network privacy preserving (SNPP)
62 problem of sequential releases. To well understand the SNPP problem for se-
63 quential releases, Section 4 proposes a solution to anonymizing one instance
64 of the social network and shows that this solution fails to give the required
65 privacy level in the case of sequential releases. Section 5 proposes our sequen-
66 tial releases SNPP solution. Section 6 reports experimental results. Section
67 7 ends up this work with conclusions and future directions.

68 2. RELATED WORKS

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

71 egories, depending on the type of the published graph (output of the solu-
72 tion): *Anonymous graph*, *Disturbed graph*, *Clustered graph*, *Uncertain graph*
73 *and Statistical results*. Most of the existing works deal with the problem of
74 privacy preserving in the context of one release publication. We give exam-
75 ples of these works for each category:

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

- 81 • k -degree anonymity [3]: to prevent an attacker, whose background is
82 the victim node's degree, to re-identify the victim node in the published
83 graph.
- 84 • k -neighborhood anonymity [4]: to prevent the risks of identity disclo-
85 sure. It considers an attacker having the victim node's neighborhood
86 as background.
- 87 • k -automorphism anonymity [5][6]: to prevent an attacker, whose back-
88 ground is the subgraph constructed by the immediate neighbors of a
89 victim node, to re-identify the victim node in the published graph. So,
90 for any subgraph X , this technique is applied to construct an anyo-
91 nous 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 $(1-\epsilon)n$ 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:

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

92 where G and \tilde{G} are two neighboring graphs and s represents an outcome
 93 produced by A .

94 According to the notion of neighboring graphs, we note two levels:

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

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

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

102 The case of sequential releases is less considered than that of one release.
103 Moreover, there is little work concerning solutions that enable publishing se-
104 quential releases of social networks data while preserving the privacy. Hence,
105 several privacy issues are still challenging in the case of sequential releases.
106 In [24], Wang et al. study the problem of real-time spatio-temporal crowd-
107 sourced social network data publishing over infinite streams. They propose a
108 solution RescueDP "REal-time Spatio-temporal Crowd-soUrcEd Data Pub-
109 lishing with Differential Privacy" to protect any users' mobility trace. So,
110 they group regions with small statistics together by considering the similar-
111 ity of data change, and then they add a Laplace noise to each group. Their
112 solution considers a rather different privacy issue of user's mobility than in
113 our paper focusing on users', groups' and edges' privacy preserving. Also,
114 their solution does not publish the whole social network, it allows publishing
115 only statistics (user mobility) over the social networks. In [13], Bhagat et al.
116 propose a solution which allows publishing anonymized versions of the social
117 network in the context of sequential releases. Authors in [13] consider the
118 problem of privacy in social networks modeled as a bipartite graph; they hide
119 the mapping between a node and the corresponding entity by partitioning
120 the set of nodes into groups of size k . They provide methods which use link
121 prediction algorithms to model the evolution of the social network and to
122 predict the future structure. The prediction is used to choose an anonymiza-
123 tion which is expected to remain safe and useful for future releases. Their
124 solution provides methods to anonymize a dynamic social network when new
125 nodes and edges are added to the published social network.

126 In our work, we consider the problem of the identity and affiliation links
127 privacy preserving in social networks sequential releases, and we propose a
128 solution that produces a published bipartite graph which is both anonymous
129 and clustered. Indeed, our solution uses techniques that group nodes into
130 classes with size greater than or equal to k and achieves the k -anonymity
131 principle. In our work we consider both cases, that of nodes/edges adding
132 and that of nodes/edges deleting; as well as the case where several nodes,
133 users and groups, share the same attributes' list values.

134 **3. SOCIAL NETWORK PRIVACY PRESERVING PROBLEM**
135 **STATEMENT**

136 This section defines the Social Network Privacy Preserving (SNPP) prob-
137 lem in the context of publishing sequential releases of the same evolving social
138 network.

139 To represent the social network data, we use a labeled bipartite graph which
140 has two types of nodes that represent users and groups. Each node has sev-
141 eral attributes representing the node’s profile. These attributes are called, in
142 this paper, ”the node attributes list”. The bipartite graph has also a set of
143 edges, representing the affiliation links between users and groups.

144 *3.1. Bipartite graph model*

145 We represent a time varying social network as a labeled bipartite graph
146 $G_t = (V_t, W_t, E_t, L_{V_t}, L_{W_t})$, where:

- 147 • V_t : is a set of nodes which represents the users who are present in the
148 social network at time t .
- 149 • W_t : is a set of nodes that represents the groups and other online con-
150 tents of interest such as, photos, web pages. . . . These nodes are present
151 in the social network at time t .
- 152 • E_t : is a set of edges. An edge $e(u_x, u_y)_t$ represents an affiliation link
153 between a user u_x and a group u_y . E_t is the set of edges that are present
154 in the social network at time t .
- 155 • L_{V_t} : is a set of attributes that describes a node $u_x \in V_t$.
- 156 • L_{W_t} : is a set of attributes that describes a group $u_y \in W_t$.

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

166 in a real social network.

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

169 3.2. Considered actions

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

- 172 1. **Act1 (Persisting nodes)**: when a node N (user or group) is present
173 in the current and previous social network sequence i.e. $N \in G_t$ and N
174 $\in G_{t-1}$.
- 175 2. **Act2 (Persisting edges)**: when an edge $e(u_x, u_y)_t$ is present in the
176 current and previous social network sequence i.e. $e(u_x, u_y)_t \in G_t$ and
177 $e(u_x, u_y)_t \in G_{t-1}$.
- 178 3. **Act3 (Adding nodes)**: when a node N (user or group) is present in
179 the current, but not in the previous, social network sequence i.e. $N \in$
180 G_t and $N \notin G_{t-1}$.
- 181 4. **Act4 (Deleting nodes)**: when a node N (user or group) is present in
182 the previous, but not in the current, social network sequence i.e. $N \in$
183 G_{t-1} and $N \notin G_t$.
- 184 5. **Act5 (Adding edges)**: when an edge $e(u_x, u_y)_t$ is present in the cur-
185 rent, but not in the previous, social network sequence i.e. $e(u_x, u_y)_t \notin$
186 G_{t-1} and $e(u_x, u_y)_t \in G_t$.
- 187 6. **Act6 (Deleting edges)**: when an edge $e(u_x, u_y)_{t-1}$ is present in the
188 previous, but not in the current, social network sequence i.e. $e(u_x, u_y)_{t-1}$
189 $\notin G_t$ and $e(u_x, u_y)_{t-1} \in G_{t-1}$.

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

193 3.3. Attacker knowledge

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

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

203 *3.4. SNPP Problem statement*

204 **Definition 1.** *privacy level: k*

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

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

211 **Definition 2.** *SNPP problem:*

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

217 *The SNPP problem, denoted by:*

$$218 \quad \text{SNPP}(\mathfrak{G}, G_t, k, \text{attacker}) \rightarrow \bar{G}_t$$

219 *consists in producing the anonymous bipartite graph \bar{G}_t that ensures the fol-*
 220 *lowing requirements:*

- 221 • **RQ₁ (Users' privacy):** *the attacker cannot re-identify the targeted*
 222 *individual (i.e. find which node is the targeted individual) with prob-*
 223 *ability greater than $\frac{1}{\lambda_x k}$, where $(\lambda_x - 1)$ represents the number of users'*
 224 *nodes which have the same attributes list values and which are published*
 225 *at the same time as the targeted individual.*
- 226 • **RQ₂ (Groups' privacy):** *the attacker cannot re-identify the targeted*
 227 *group (find which node is the targeted group) with probability greater*
 228 *than $\frac{1}{\lambda_y k}$, where $(\lambda_y - 1)$ represents the number of groups' nodes which*
 229 *have the same attributes list values and which are published at the same*
 230 *time as the targeted group.*
- 231 • **RQ₃ (Edges' privacy):** *the attacker cannot determine the existence*
 232 *of a link between the targeted individual, u_x , and the targeted group, u_y ,*
 233 *with probability greater than $\frac{1}{k}$.*

- 234 • **RQ₄ (Utility requirement)**: using the anonymous bipartite graph,
 235 \bar{G}_t , we should be able to answer different mining queries with high ac-
 236 curacy.

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:

$$\begin{aligned}
 U: Q &\rightarrow R \\
 q &\rightarrow \frac{|\text{answer}(q_G) - \text{answer}(q_{\bar{G}})|}{\text{answer}(q_G)} \quad (2)
 \end{aligned}$$

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

240 In the following sections, we construct our privacy preserving sequential re-
 241 leases of social networks through anonymization techniques that group nodes
 242 into classes of size at least k , and we give formal proofs to ensure the require-
 243 ments **RQ₁**, **RQ₂** and **RQ₃**. We evaluate the utility accuracy of our solution
 244 in section 6.

245 4. SNPP-1RELEASE AND LIMITATIONS

246 To well understand the SNPP problem, in the case of sequential releases,
 247 we first study the case of only one release. First, we present an overview
 248 of an anonymization solution for a single bipartite graph, we define a *safely*
 249 *partitioning condition*, and we present the associated algorithm, before iden-
 250 tifying the solution limitations.

251 In Table 1, we summarize the terminology and notations used in this paper.
 252

253 4.1. Overview

254 To produce the published bipartite graph, we use anonymization tech-
 255 niques. So, we create classes for users and others for groups, where each class
 256 contains at least k nodes and we publish only the number of edges between
 257 classes. In order to keep the attributes lists values distribution unchanged,
 258 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 \cap 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: $B \in C$. PermutSet if: $ C.attributes list values \cap 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

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

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

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

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

297 groups' classes, such as C_2 .attributes lists values = {(physician, 30), (physi-
 298 cian, 25)}, B_1 .attributes lists values= {Football, Sport} and B_2 .attributes
 299 lists values= {Art, Music} respectively.

300 Using the anonymous bipartite graph, an attacker cannot guess which node
 301 has which attributes list values, because the true mapping between nodes
 302 and attributes lists values is masked. Example: for C_1 , an attacker cannot
 303 identify the node, among u_1 and u_2 , that has (Engineer, 25) as attributes
 304 list values. The attacker cannot also know how nodes connect to each other.
 305 For example, the attacker cannot know how u_1 and u_2 connect to g_1 and g_2 ,
 306 as in the anonymous bipartite graph, there is only the total number of edges
 307 between C_1 and B_1 .

308 Given an original bipartite graph, the problem of creating an anonymous
 309 bipartite graph \tilde{G}_t , which meets the SNPP privacy requirements with the
 310 minimum utility cost is NP-hard.

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

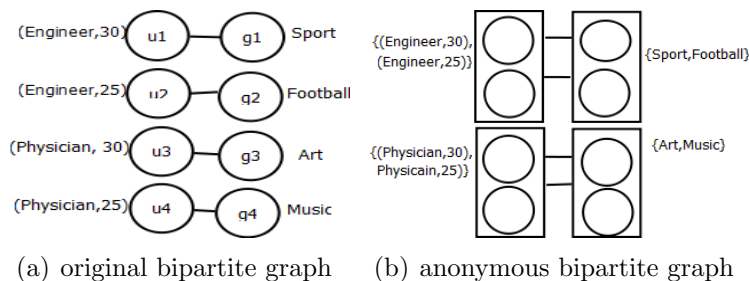


Figure 1: Anonymization technique

311

312 consider that:

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

317 The SNPP problem, in this case, is exactly the clustering problem. So, the
 318 SNPP problem is NP-hard.

319 This anonymization method enables creating only classes of size at least k .
 320 In what follows, we show that Property 1 is required to ensure the three
 321 SNPP privacy requirements (**RQ**₁, **RQ**₂ and **RQ**₃).

322 4.2. Safely partitioning condition

323 This subsection introduces the property *safely partitioning condition*, on
 324 which we build our SNPP1R algorithm (see algorithm 1) that preserves the
 325 three SNPP privacy requirements of the Definition 2 (**RQ**₁, **RQ**₂ and **RQ**₃).

326 **Property 1. Safely partitioning condition**

327 *Let us consider the SNPP problem:*

$$328 \quad \text{SNPP}(\mathfrak{G}, G_t, k, \text{attacker}) \rightarrow \bar{G}_t$$

329 *with $G_t = (V_t, W_t, E_t, L_{V_t}, L_{W_t})$, $\bar{G}_t = (X_t, Y_t, \bar{E}_t, L_{CX_t}, L_{CY_t})$ and $\mathfrak{G} = \emptyset$:*
 330 *i.e. the attacker has only access to \bar{G}_t .*

331 *\bar{G}_t satisfies the three SNPP privacy requirements if:*

- 332 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
 333 values= N_2 .attributes list values then $C_{y_1} \neq C_{y_2}$.
- 334 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
 335 values= N_2 .attributes list values then $C_{x_1} \neq C_{x_2}$.
- 336 3. **SPa**₃: \forall Classes $C_x \in X_t$ and $C_y \in Y_t$, nb is the number of edges
 337 between C_x and C_y , $nb \leq k$.
- 338 4. **SPa**₄: \forall Classes $C_x \in X_t \cup Y_t$, $|C_x| \geq k$.

339 *We call these conditions the safely partitioning condition \square .*

340 We give justifications and explanations of the conditions **SPa**₁, **SPa**₂, which
 341 require that no two group nodes (respectively two user nodes) with the same
 342 attributes list values belong to the same class in the anonymous bipartite
 343 graph, in section 5.1.

344 **Proof:**

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

- 348 • there are $(\lambda_x - 1)$ other nodes published at the same time as u_x having
 349 the same attributes list values like u_x . u_x and the other
 350 $\lambda_x - 1$ nodes belong to λ_x different classes $(C_x, C_{x_1}, \dots, C_{x_{\lambda_x-1}}) \in X_t$,
 351 as required by the condition **SPa**₂.

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

354 is: $P(u_x) = \frac{1}{|C_x|+|C_{x1}|+\dots+|C_{x\lambda_x-1}|}$ as the size of each $C_{xi} \geq k$ (the condition
355 **SPa₄**)
356 Then $P(u_x) \leq \frac{1}{\lambda_x k}$.
357 So, the *safely partitioning condition* guarantees the users' privacy require-
358 ment (**RQ₁**). Similarly, we prove that the groups' privacy (**RQ₂**) require-
359 ment is guaranteed by the *safely partitioning condition*.

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

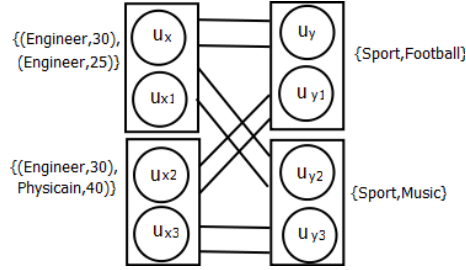


Figure 2: edges privacy

360
361 ist between a node u_x , representing the targeted individual, and a node u_y ,
362 representing the targeted group, is less than or equal to $\frac{1}{k}$. Let's suppose
363 that:

- 364 • there are $(\lambda_x - 1)$ other nodes published at the same time as u_x and
365 having the same attributes list values like u_x . u_x and the other
366 $\lambda_x - 1$ nodes belong to λ_x different classes $(C_x, C_{x1}, \dots, C_{x\lambda_x-1}) \in X_t$, as
367 required by the condition **SPa₂**.
- 368 • there are $(\lambda_y - 1)$ other nodes published at the same time as u_y and
369 having the same attributes list values as u_y . u_y and the other
370 $\lambda_y - 1$ nodes belong to λ_y different classes $(C_y, C_{y1}, \dots, C_{y\lambda_y-1}) \in Y_t$, as
371 required by the condition **SPa₁**.
- 372 • In the worst case, all nodes in $C_x, C_{x1}, \dots, C_{x\lambda_x-1}$ have links with nodes
373 in $C_y, C_{y1}, \dots, C_{y\lambda_y-1}$. As shown in Figure 2: by taking u_x .attributes
374 list = (engineer, 30) and u_y .attributes list = Sport. Note that the nodes'
375 identifiers i.e. $u_x, u_{x1}, \dots, u_{x3}$ and $u_y, u_{y1}, \dots, u_{y3}$, are not published in
376 the anonymous bipartite graphs, they are used only for explanation.

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

379 $P(u_x, u_y, e) = \frac{1}{|C_x|+|C_{x1}|+\dots+|C_{x\lambda_{x-1}}|} * \frac{1}{|C_y|+|C_{y1}|+\dots+|C_{y\lambda_{y-1}}|} * n_{\text{bedge}}$

380 As the size of each $C_i \geq k$ (the condition **SPa₄**) then:

381 $P(u_x, u_y, e) \leq \frac{1}{\lambda_x k} * \frac{1}{\lambda_y k} * n_{\text{bedge}}$

382 where n_{bedge} is the number of edges whose two endpoints might be u_x and
383 u_y .

- 384 • As the targeted group, u_y , may belong to each λ_y classes ($C_y, C_{y1}, \dots,$
385 $C_{y\lambda_{y-1}}$),
- 386 • and the targeted individual, u_x , may belong to each λ_x classes ($C_x, C_{x1}, \dots,$
387 $C_{x\lambda_{x-1}}$),
- 388 • and the number of edges between any two classes, C_{xi} and C_{yi} , is $\leq k$
389 (according to the condition **SPa₃**).

390 Then: $n_{\text{bedge}} \leq k * \lambda_x * \lambda_y$.

391 So: $P(u_x, u_y, e) \leq \frac{1}{\lambda_x k} * \frac{1}{\lambda_y k} * \lambda_x \lambda_y k$. Therefore $P(u_x, u_y, e) \leq \frac{1}{k}$

392 So, the *safely partitioning condition* guarantees the edges privacy requirement
393 (**RQ₃**).

394 Figure 3 shows via examples, the violation of the requirements **RQ₁**, **RQ₂**
395 and **RQ₃** due to the dissatisfaction of the conditions **SPa₁**, **SPa₂** and **SPa₃**.

396

397 4.3. SNPP-1 Release (SNPP1R) algorithm

398 We propose a greedy algorithm (algorithm 1) that creates classes with at
399 least k nodes. The SNPP1R algorithm relies on the concept of greedy safe
400 k -grouping, used in [15][16][17], while minimizing the utility cost and consid-
401 ering the two types of nodes, user and group, and our new *safely partitioning*
402 *condition*. It creates one groups' class then one users' class alternatively un-
403 til all groups' nodes and users' nodes are grouped into classes. To partition
404 nodes, the SNPP1R algorithm calls the procedure Safely partitioning(X, Y, Z, k),
405 shown in algorithm 2, where X and Y are two partitions; Z is a set of nodes
406 and k is the privacy level.

407 The procedure Safely partitioning takes each time the first node, not yet
408 grouped, in Z , and it creates a new class C containing this node, line 1.
409 Then it repeatedly adds nodes to C under the *safely partitioning condition*
410 and with the minimum utility cost until the size of C reaches k , lines 2-12. If
411 the size of the created class, C , cannot reach k , the algorithm 2 removes this
412 class and groups its nodes in suitable classes, under the *safely partitioning*

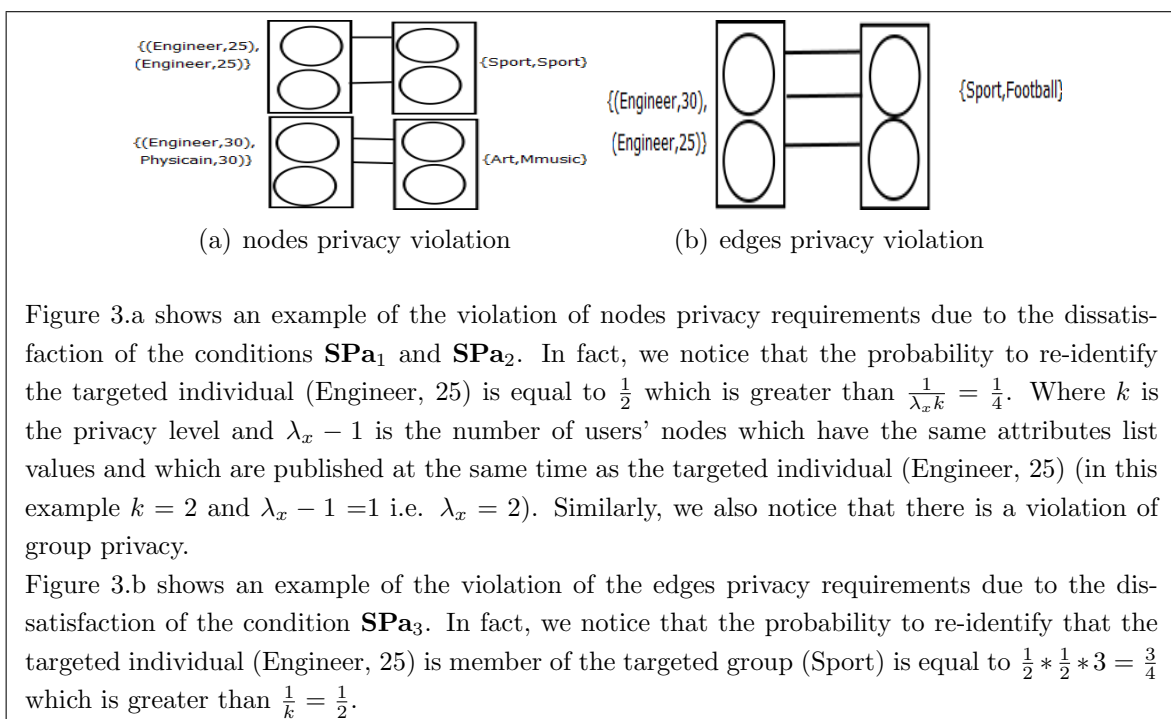


Figure 3: Explanations of the Property 1 conditions

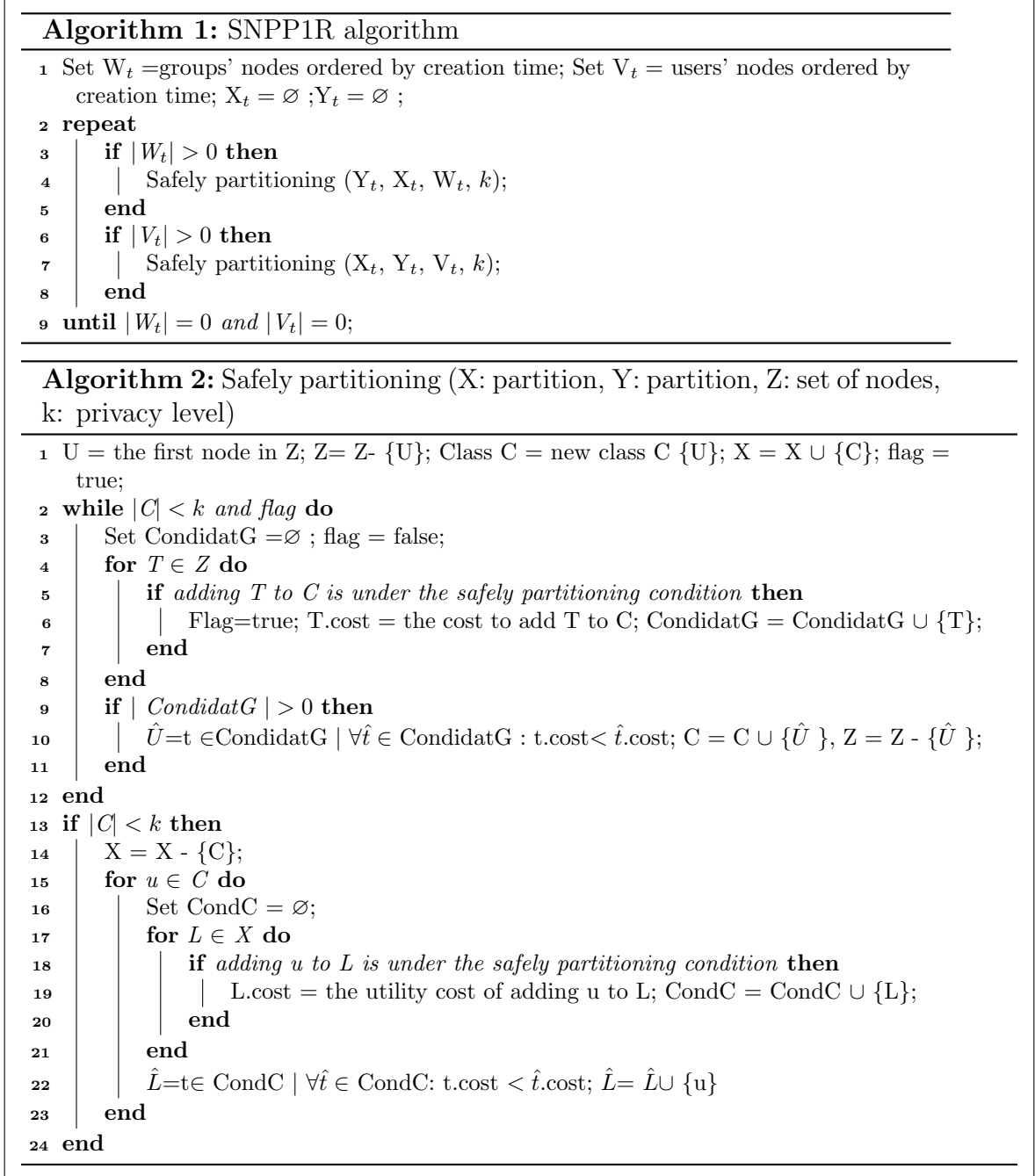


Figure 4: SNPP-1 Release algorithms

413 *condition* and with the minimum utility cost, lines 13-24.

414 The procedure Safely partitioning is a greedy algorithm that creates classes
415 with at least k nodes under the *safely partitioning condition* which depends
416 on the sparse property of social networks [17]. In fact, the social networks
417 graphs are relatively sparse [17], i.e. each user is typically member in only
418 a small fraction of all groups, and each group holds only a few users. As a
419 consequence, classes are easier to be found, based on the conditions in prop-
420 erty 1, i.e. **SPa₁**, **SPa₂**, **SPa₃** and **SPa₄**.

421

422 4.4. Complexity of the algorithm SNPP1R

423 The algorithm SNPP1R cost is in the worst case less than $O(N^2)$, where
424 N ($N = N_{user} + N_{group}$) is the number of nodes, N_{user} is the number of user
425 nodes, and N_{group} is the number of group nodes.

426 Let's consider adding a node to a class as an elementary operation.

427 To create a user class, the algorithm SNPP1R has to browse all the user
428 nodes not yet grouped to choose the node that satisfies the *safely partition-*
429 *ing condition* with the minimum utility cost and adds it to the class to be
430 created. So, to add the second user node to the first user class we have to do
431 $N_{user} - 2$ operations (i.e. verify all user nodes not yet grouped), then $N_{user} - 3$
432 for the third user node etc. To add the second user node to the second user
433 class, we need $N_{user} - (k + 2)$ operations etc (k is the class size). So, in total,
434 we have (N_{user}^2) operations. Similarly, we have (N_{group}^2) operations to create
435 groups' classes. So, the complexity of the algorithm is $O(N_{user}^2 + N_{group}^2)$,
436 which is less than $O(N^2)$.

437 Note that, the algorithm SNPP1R complexity does not depend on the at-
438 tributes lists values of nodes, as it does not generalize the attributes lists val-
439 ues to anonymize the graph, and it only performs comparison between these
440 attributes lists values to preserve the utility. The operation of "comparison
441 between attributes lists values" is dominated by the operation "adding node
442 to a class".

443 The complexity of the algorithm SNPP1R can be enhanced to reach $O(N)$,
444 if at no substantial utility cost (section 4.1). In that case, the algorithm
445 SNPP1R does not browse all the nodes not yet grouped, but instead, as soon
446 as a node is found satisfying the *safely partitioning condition*, it is added
447 directly to the class to be created.

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

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

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

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

464 4.5. Inadequacy to sequential releases

465 Using the SNPP1R algorithm to produce sequential anonymous versions,
 466 harvested at different times of the same social network, can lead to privacy
 467 violation. That is due to the generation of each anonymous version being
 468 made separately, thus leading to nodes changing classes from one release to
 469 another, and having their classes' attributes lists values modified. Hence,
 470 an attacker making comparison of classes' attributes lists values between
 471 releases, can violate the privacy.

472 As we order the nodes in the SNPP1R algorithm in the same way (by the
 473 creation time). So, there are three reasons that lead nodes changing their
 474 classes from one release to another:

- 475 1. The violation of the condition **SPa₃** (**Act5**): adding new edges between
 476 old users, who belong to the same class; and old groups, which belong
 477 to the same class, can lead to a number of edges between two classes
 478 greater than k . As that does not satisfy the condition **SPa₃**, then nodes
 479 must leave their classes until the condition **SPa₃** is satisfied again.
- 480 2. Deleting nodes (**Act4**): deleting nodes can produce classes with less
 481 than k nodes; hence other nodes must be added to these classes until
 482 their size is greater than or equal to k .
- 483 3. Adding new nodes (**Act3**): the old nodes can be grouped with the new
 484 ones if they better preserve the data utility.

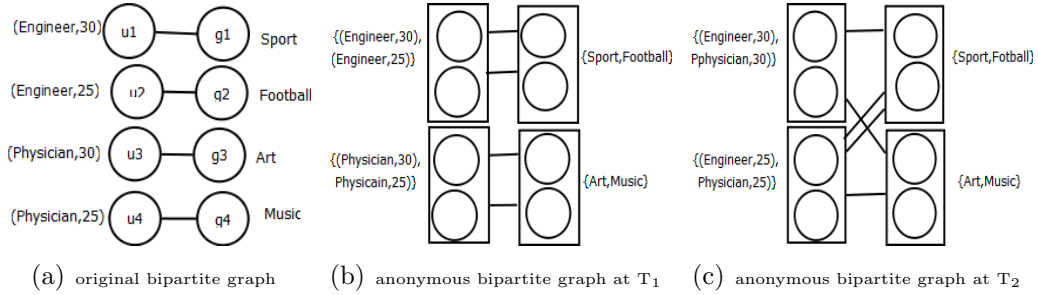


Figure 5: Example of the SNPP1R algorithm’s privacy violation in the case of sequential releases

485 **Example:** Figure 5 gives an example of privacy violation with the privacy
 486 level $k = 2$, the targeted individual = u_2 and her/his attributes list values
 487 = (Engineer, 25). At T_1 , there are four users and four groups, as depicted
 488 in Figure 5.a. Note that the nodes’ identifiers i.e. u_1, \dots, u_4 and g_1, \dots, g_4 ,
 489 are not published in the anonymous bipartite graphs, they are only used for
 490 explanation. The anonymous bipartite graph \bar{G}_{T_1} , produced by the SNPP1R
 491 algorithm, is presented in Figure 5.b. At T_2 , the targeted individual, u_2 ,
 492 joins the group g_1 and the anonymous bipartite graph \bar{G}_{T_2} , produced by the
 493 SNPP1R algorithm, is presented in Figure 5.c.

494 the attacker is assumed to know that the nodes do not change classes unless
 495 there is a violation of the condition **SPa₃**, or some nodes’s adding or deleting.
 496 Thus by comparing \bar{G}_{T_1} and \bar{G}_{T_2} , the attacker notices that the targeted
 497 individual (Engineer, 25), has changed her/his class and that is due to the
 498 violation of the condition **SPa₃**. Then the attacker can infer that the targeted
 499 individual, (Engineer, 25), is member of the groups with topics ”Sport” and
 500 ”Football”, which violates privacy.

501 5. SNPP-SEQUENTIAL RELEASES

502 This section proposes a privacy preserving solution for publishing sequen-
 503 tial releases of the same social network, where addition or deletion of nodes
 504 or edges might happen from one release to another.

505 As discussed in subsection 4.5, the privacy threats are coming from changes
 506 occurring among the classes’ attributes lists values, in particular, when one
 507 of the three following events occur:

- 508 (i) the violation of the condition **SPa₃** due to newly added edges between
 509 old users’ classes and old groups’ classes (**Act5**). (ii) newly deleted nodes

510 **(Act4)**. (iii) newly added nodes **(Act3)**.

511 To avoid these privacy threats, we need to keep the attributes lists values
512 of each class invariant over time, so the attributes lists values of each class
513 do not change from one release to another, and an attacker is unable to re-
514 identify the targeted individual (group) and her/his affiliation links. This
515 idea is inspired from the notion of m -invariance proposed by Xiao et al. [32]
516 for dynamic databases. The m -invariance ensures that the sensitive values
517 of each anonymized group (Quasi-Identifier) remain the same over publica-
518 tions.

519 Those three sensitive actions (i.e. the violation of the condition \mathbf{SPa}_3 , newly
520 deleted nodes and newly added nodes) are next provided with technical pri-
521 vacy preserving solutions.

522 5.1. Safe permutation

523 *Safely permuting condition*, hereafter defined as property 2, removes the
524 violation of the condition \mathbf{SPa}_3 issue identified in Section 4.5. A violation of
525 the condition \mathbf{SPa}_3 between two classes can be removed by decreasing the
526 number of edges between these two classes, and permuting nodes belonging
527 to one of these classes with other nodes of the other class, but under the
528 condition that the *safely permuting condition* is respected.

529 **Example:** Figure 6 gives an example of a permutation that ensures the
530 requirements \mathbf{RQ}_1 , \mathbf{RQ}_2 and \mathbf{RQ}_3 with the privacy level $k = 2$, the targeted
531 individual = u_2 and her/his attributes list values = (Engineer, 25). At T_1 ,
532 there are four users and four groups: As depicted in Figure 6.a, note that the
533 nodes identifier i.e. u_1, \dots, u_4 and g_1, \dots, g_4 , are not published in the anony-
534 mous bipartite graphs. They are only used for explanation. The anonymous
535 bipartite graph \bar{G}_{T_1} is presented in Figure 6.b. It contains two users' classes,
536 C_1 .attributes lists = {(Engineer, 30), (Engineer, 25)} and C_2 .attributes lists
537 = {(Engineer, 25), (Physician, 25)}, and two groups' classes, B_1 .attributes
538 lists = (Sport, Football), and B_2 .attributes lists = (Art, Music). At T_2 , the
539 user u_2 joins the group g_1 , hence, there is a violation of the condition \mathbf{SPa}_3
540 as the number of edges between C_1 and B_1 is $3 > k$. To remove this violation
541 of the condition \mathbf{SPa}_3 without violating the targeted individual privacy we
542 should:

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

- 545 • Permute these two nodes if that removes the violation of the condition
 546 **SPa₃** and does not create other ones.

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

- 551 • With probability equals to $\frac{1}{2}$ that the difference between \bar{G}_{T2} and \bar{G}_{T1}
 552 is due to the evolution of the social network, because users can join or
 553 leave groups over time. In this case, the requirements **RQ₁**, **RQ₂** and
 554 **RQ₃** are satisfied.
- 555 • With probability equals to $\frac{1}{2}$ that the difference between \bar{G}_{T2} and \bar{G}_{T1}
 556 is due to removing the violation of the condition **SPa₃** between C_1 and
 557 B_1 . In this case, the groups' privacy requirement is satisfied, we do
 558 not permute between groups. For the requirement **RQ₁**, the attacker
 559 identifies that the two users' nodes permuted are u_2 and u_3 , so the
 560 probability that the targeted individual is u_2 (respectively u_3) is equal
 561 to $\frac{1}{2} * \frac{1}{2} = \frac{1}{\lambda_{u_2} k}$,
- 562 1. where the first $\frac{1}{2}$ is the probability that this difference between
 563 \bar{G}_{T2} and \bar{G}_{T1} is due to the suppression of the violation of the
 564 condition **SPa₃**,
 - 565 2. the second $\frac{1}{2}$ is the probability that the targeted individual is u_2
 566 (respectively u_3),
 - 567 3. k is the privacy level, in this case $k=2$,
 - 568 4. and $\lambda_{u_2} - 1$ is the number of nodes which have the same attributes
 569 list values and which are published at the same time as the tar-
 570 geted individual u_2 , in this case $\lambda_{u_2} - 1 = 1$ (u_3).
 - 571 5. Hence the users' privacy requirement is satisfied too.

572 If the targeted individual is u_2 , the probability of affiliation link identi-
 573 fication is equal to $\frac{1}{2} * \frac{1}{2} * \frac{1}{2} * 2$, where the first $\frac{1}{2}$ is the probability that
 574 this difference between \bar{G}_{T2} and \bar{G}_{T1} is due to the suppression of the
 575 violation of the condition **SPa₃**; the second $\frac{1}{2}$ is the probability that the
 576 targeted individual is u_2 ; the third $\frac{1}{2}$ is the probability that the group
 577 selected is the targeted group and 2 is the number of edges between u_2
 578 and B_1 .

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

583 **Property 2. *safely permuting condition***

584 *A permutation between two nodes of the same type, N_1 and N_2 , that belong*
585 *to two different classes, C_x and C_{zi} , does not violate the requirements \mathbf{RQ}_1 ,*
586 *\mathbf{RQ}_2 and \mathbf{RQ}_3 and removes the violation of the condition \mathbf{SPa}_3 between C_x*
587 *and C_y if:*

- 588 1. \mathbf{SPE}_1 : $N_1 \in C_x, N_2 \in C_{zi}, |C_x.\text{attributes list values} \cap C_{zi}.\text{attributes list}$
589 $\text{values}| \geq \frac{k}{2}, k$ is the privacy level.
- 590 2. \mathbf{SPE}_2 : $N_1.\text{attributes list values} = N_2.\text{attributes list values}$.
- 591 3. \mathbf{SPE}_3 : The permutation between N_1 and N_2 decreases the number of
592 edges or removes the violation of the condition \mathbf{SPa}_3 between C_x and
593 C_y and does not create other violations of the condition \mathbf{SPa}_3 .

594 We permute nodes under these three conditions ($\mathbf{SPE}_1, \mathbf{SPE}_2, \mathbf{SPE}_3$), until
595 the violation of the condition \mathbf{SPa}_3 between C_x and C_y is removed.

596 We call these conditions: *safely permuting condition*. \square

597 We justify \mathbf{SPa}_1 and \mathbf{SPa}_2 (see Property 1): The condition \mathbf{SPE}_1 requires
598 that the two nodes which must be permuted, belong to two different classes,
599 and the condition \mathbf{SPE}_2 requires that these two nodes should have the same
600 attributes list values. So, to ensure that these two conditions (\mathbf{SPE}_1 and
601 \mathbf{SPE}_2) are satisfied, we should not put the nodes that have the same attributes
602 list values in the same class, when creating classes. That is why, conditions
603 \mathbf{SPa}_1 and \mathbf{SPa}_2 have to be satisfied when grouping nodes into classes.

604 The proof of this property is shown in Appendix A.

605 *5.2. Safely permuting algorithm*

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

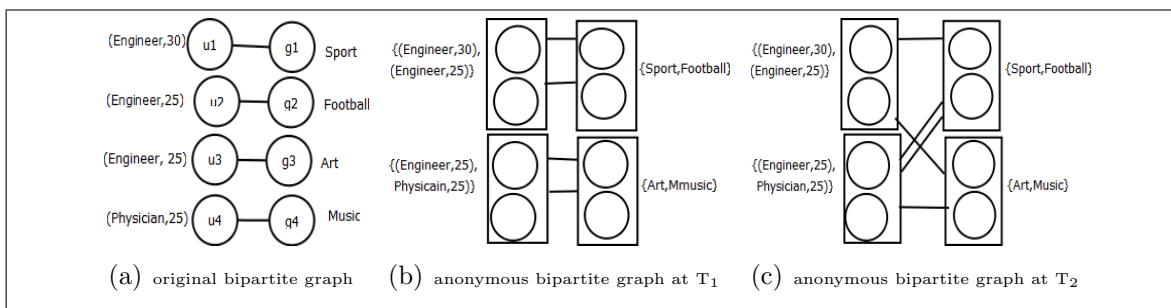


Figure 6: A permutation that preserves the privacy in the case of sequential releases

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

623 *Procedure of noise groups' class creation:* we create a noise group class, by
 624 using noise nodes, to remove the violation of the condition \mathbf{SPa}_3 between
 625 a group class C and a user class T . For the sake of simplicity, we create k
 626 noise nodes such as: $\frac{k}{2}$ noise nodes have the same attributes lists values as
 627 $\frac{k}{2}$ different nodes in C , and the attributes lists values of the other $\frac{k}{2}$ noise
 628 nodes are assigned randomly.

629 5.3. Dynamic actions: deleting and adding nodes

630 Deleting old nodes

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

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

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

```
1 while C violate the condition  $SPa_3$  do
2   T = the first node in  $\hat{X}$  ;
3   B = safe_class(C, T, C.PermutSet, T.PermutSet) ;
4   if B.type = C.type then
5     Let  $N_1 \in C$  and  $N_2 \in B$  |  $N_1$ .attributes list values =  $N_2$ .attributes list values;
6     C=C-{  $N_1$  } ; C=C  $\cup$  {  $N_2$  } ; B=B-{  $N_2$  } , B=B  $\cup$  {  $N_1$  };
7   else
8     Let  $N_1 \in T$  and  $N_2 \in B$  |  $N_1$ .attributes list values =  $N_2$ .attributes list values;
9     T=T-{  $N_1$  } ; T=T  $\cup$  {  $N_2$  } ; B=B-{  $N_2$  } , B=B  $\cup$  {  $N_1$  };
10  end
11   $\hat{X}$  =  $x \in X_t$  | the nbr of edges between x and C > k;
12 end
```

Algorithm 4: SNPP-Sequential Releases algorithm

```
1  $W_{ti}$  = 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  $T_{i-1}$ ;
2 Safely deleting (  $A_W, W_{ti-1}, W_{ti}$  ) ;
3 Safely deleting (  $A_V, V_{ti-1}, V_{ti}$  ) ;
4 for  $C \in Y_{ti}$  do
5   set  $\hat{X} = x \in X_{ti}$  | the nbr of edges between x and C > k;
6   Safely permuting (C,  $\hat{X}$ , k);
7 end
8 if  $|\hat{W}| < k$  then
9    $W_{ti} := W_{ti} - \hat{W}$ ;  $\hat{W} = \emptyset$ ;
10 end
11 if  $|\hat{V}| < k$  then
12    $V_{ti} := V_{ti} - \hat{V}$ ;  $\hat{V} = \emptyset$ ;
13 end
14 if  $|\hat{W}| \geq k$  or  $|\hat{V}| \geq k$  then
15   repeat
16     if  $|\hat{W}| > 0$  then
17       Safely partitioning (  $Y_{ti}, X_{ti}, \hat{W}, k$  );
18     end
19     if  $|\hat{V}| > 0$  then
20       Safely partitioning (  $X_{ti}, Y_{ti}, \hat{V}, k$  );
21     end
22   until  $|\hat{W}| = 0$  and  $|\hat{V}| = 0$ ;
23 end
```

Figure 7: SNPP-Sequential Release algorithms

639 We call this procedure the Safely deleting procedure.

640

641 **Adding new nodes**

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

644

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

648 We call this procedure the adding nodes procedure.

649

650 5.4. SNPP solution

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

655 **Property 3. SNPP solution:**

656 *Let:*

$$657 \text{SNPP}(\mathfrak{G}, G_t, k, \text{attacker}) \rightarrow \bar{G}_t$$

658 with $G_t = (V_t, W_t, E_t, L_{V_t}, L_{W_t})$, $\bar{G}_t = (X_t, Y_t, \bar{E}_t, L_{CX_t}, L_{CY_t})$ and
659 $\mathfrak{G} = \bar{G}_1, \dots, \bar{G}_{t-1}$. Assume that the attacker has access to \mathfrak{G} and \bar{G}_t .

660 \bar{G}_t satisfies the requirements **RQ**₁, **RQ**₂ and **RQ**₃ if:

- 661 1. the grouping of new nodes together in classes of size at least k , satisfies
662 the adding nodes procedure.
- 663 2. the old nodes deletion satisfies the Safely deleting procedure.
- 664 3. the removal of the violation of the condition **SPa**₃ satisfies property 2.

665 5.5. SNPP-Sequential Releases (SNPPnR) algorithm

666 We propose a greedy algorithm (algorithm 4) that permits publishing se-
667 quential releases of the same social network over time. This algorithm ensures
668 that the requirements **RQ**₁, **RQ**₂ and **RQ**₃ are preserved. In this algorithm,
669 we need only to keep the last original and published bipartite graph and

670 the set of nodes that their deletion is delayed in the previous releases when
671 preparing the current data to be published. First, this algorithm calls the
672 Safely deleting procedure. This procedure permits deleting nodes from the
673 published bipartite graph without causing privacy leak. For each old node
674 u that should be deleted ($u \in G_{t_{i-1}}$ and $u \notin G_{t_i}$), Safely deleting searches a
675 new node, T ($T \in G_{t_i}$ and $T \notin G_{t_{i-1}}$), that has the same attributes list values
676 and the same type as u . If such node is found, the Safely deleting algorithm
677 deletes the old node and replaces it with the new one. In the opposite case
678 (where the node T cannot be found), the Safely deleting algorithm delays
679 the deletion of the old node, lines 2-3. Then, the algorithm SNPPnR calls
680 repeatedly the procedure Safely permuting, see algorithm 3, to remove the
681 violation of the condition \mathbf{SPa}_3 , lines 4-7. Finally, the algorithm SNPPnR
682 groups the new nodes. If the number of new nodes is less than k it delays
683 their publication by suppressing them from the current set of nodes (W_{t_i}
684 $, V_{t_i}$) in order to be considered in the next releases, lines 8-13. Else it calls
685 the procedure Safely partitioning, see algorithm 2, to group these nodes in
686 classes of size at least k under the *safely partitioning condition*. In order
687 to create classes under the condition \mathbf{SPE}_1 i.e. classes that share at least $\frac{2}{k}$
688 attributes lists values, we order the nodes by their attributes list values fre-
689 quency. We then modify the utility cost in algorithm 2, by taking the node's
690 utility cost equals to the inverse of the frequency of this node's attributes
691 list value in the bipartite graph (i.e. $N.cost = \frac{1}{N.freq+1}$, where $N.freq$ is the
692 number of nodes that have the same attributes list values as N), lines 14-23.
693 Indeed, this allows creating classes that share a large number of attributes
694 lists values, as we take each time the node that has the greatest attributes
695 list values frequency, not yet grouped, and we create a new class C containing
696 this node. Then we add repeatedly nodes to C until its size reaches k , such
697 as the next node that will be added to C is the node that has the great-
698 est attributes list values frequency, among other ones, that satisfy the *safely*
699 *partitioning condition*.

700 5.6. Complexity of the algorithm SNPPnR

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

704 **The cost of grouping new nodes:**

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

707 section 4.4).

708 **The cost of deleting old nodes:**

709 Let's consider "deleting node as an elementary operation". The cost of delet-
710 ing old nodes is:

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

- 715 • In the worst case: there is no new node with the same attributes lists
716 values as the node to be deleted. So, for each user node to be deleted,
717 we should browse all new user nodes. So, we need $N_{user-old} * N_{user-new}$
718 operations to delete user nodes; Where $N_{user-old}$ is the number of old
719 user nodes to be deleted and $N_{user-new}$ is the number of new user nodes.
720 Similarly, we need $N_{group-old} * N_{group-new}$ operations for deleting group
721 nodes; Where $N_{group-old}$ is the number of old group nodes to be deleted
722 and $N_{group-new}$ is the number of new group nodes. So, in total, we
723 need $N_{user-old} * N_{user-new} + N_{group-old} * N_{group-new}$ operations to delete
724 group and user nodes. In this case, the cost of deleting nodes is less
725 than $O(N_{old} * N_{new})$; where $N_{old} = N_{user-old} + N_{group-old}$ is the number
726 of old nodes (user and group) to be deleted, and $N_{new} = N_{user-new} +$
727 $N_{group-new}$ is the number of new nodes.

728 **The cost of removing the violations of the *safely partitioning con-*
729 *dition*:**

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

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

- 736 • In the best case, the first group class tested can remove the violation
737 of the *safely partitioning condition*. So, in this case, the cost of remov-
738 ing the violations of the *safely partitioning condition* is $O(N_{violation})$.
739 Where $N_{violation}$ is the number of violation of the *safely partitioning*
740 *condition*.

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

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

760 6. EXPERIMENTS: DATA UTILITY AND PRIVACY MEASURE- 761 MENTS

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

765 6.1. Utility evaluation

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

- 769 • Attribute properties on one side only (Type 1): This kind of queries
770 takes into account only the attributes properties of one type of nodes.
771 E.g. find the average degree of lawyers' users.
- 772 • Attributes properties on both sides (Type 2): the second kind of queries
773 takes into account the attributes properties of the two types of nodes,

774 user and group. E.g. find the number of lawyers' users who are mem-
775 bers of groups whose topic is politics.

776 6.2. Querying anonymized data

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

790 6.3. Experimental model

791 To evaluate the utility of our SNPPnR algorithm, we adapt the solution
792 proposed in [13] to our SNPP problem and then we compare our SNPPnR
793 algorithm to the adapted solution. We also compare the SNPPnR algorithm
794 with the SNPP1R algorithm.

795 The solution proposed in [13] aims to preserve the privacy of the relationships
796 between users (user-user). First, it uses the prediction links algorithms to
797 predict the evolution of these relationships. Then, the solution proposed in
798 [13] partitions new nodes together, using this prediction, such as this parti-
799 tioning remains safe in the future releases. As our interest is for the affiliation
800 relationship between users and groups, we cannot use these algorithms to pre-
801 dict the affiliation relationship evolution. So, for the sake of simplicity, we
802 group the new nodes together without using the prediction algorithm. We
803 notice that this solution does not take into account the case where several
804 nodes share the same attributes list values. To ensure the requirements **RQ**₁,
805 **RQ**₂ and **RQ**₃, we use our *safely partitioning condition* to group new nodes.
806 We also notice that the case of deleting nodes is not considered, thus we use
807 our Safely deleting procedure to remove the nodes from the social network.
808 The algorithms SNPPnR, SNPP1R and the algorithm of the adapted solution

809 (A5), proposed in [13], were implemented in Python using NetworkX. Net-
810 workX is a Python language software package for the creation, manipulation,
811 and study of the structure, dynamics, and function of complex networks.
812 We used the random data generator to generate data sets, i.e. bipartite
813 graphs. In our experiments, we used the data that are generated randomly.
814 Consequently, our results are valid for any given social network that satisfies
815 the sparse property. The first original social network bipartite graph has 1000
816 nodes, and 3660 edges. To generate the next bipartite graphs, we take the
817 previous bipartite graph and we randomly remove and add nodes and edges,
818 so that the number of nodes and edges raises from one release to another by
819 rate between 10% and 20% (these rates are observed in the data set used in
820 [29]).
821 In the experiments, for the sake of simplicity, we consider two users' at-
822 tributes, i.e. user's attributes list = (age, job), and two groups' attributes,
823 i.e. group's attributes list = (topic, subtopic).
824 We use the algorithm SNPPnR, the algorithm SNPP1R, and the adapted
825 solution, A5, proposed in [13] to publish sequential releases based on the
826 first original bipartite graph.

827

828 6.4. The data set

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

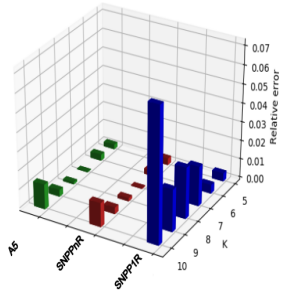
Table 2: Bipartite Graphs in different releases

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

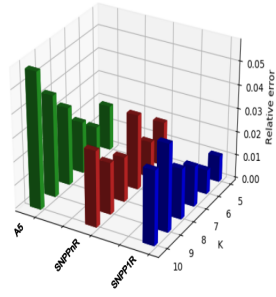
829

830 6.5. Utility accuracy

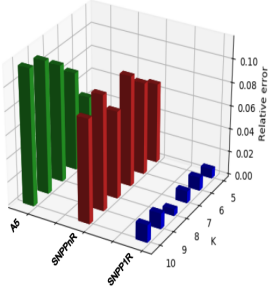
831 Figures 8 and 9 show the average relative errors of type1 and type2 queries
832 for releases R1, R3 and R5, respectively. Each point in the figures is the ex-
833 pected answer of type1 and type2 queries, on the corresponding anonymous
834 bipartite graph of 6 privacy levels using the algorithms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



(a) R1

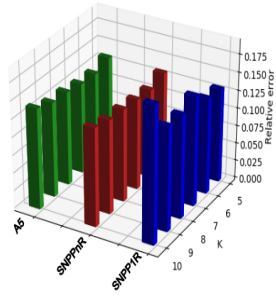


(b) R3

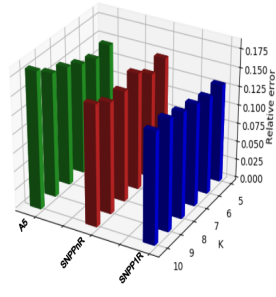


(c) R5

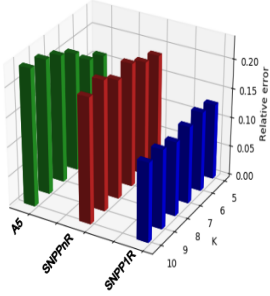
Figure 8: Relative error for type1. **a:** Average relative errors (type1) of the release R1, **b:** Average relative errors (type1) of the release R3, **c:** Average relative errors (type1) of the release R5.



(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.

836 we generate 20 consistent sampling bipartite graphs to get the "expected"
837 result. We choose to use 20 consistent sampling bipartite graphs because this
838 guarantees the reproducibility of the results . In fact, we calculated the mar-
839 gin of error,of the 20 consistent bipartite graphs, using a confidence interval
840 with 99% and we obtained a margin of error of 1.5% for type1 queries and
841 3.7% for type2 queries. The average relative query error, type2, obtained by
842 our solution, algorithm SNPPnR, ranging between 12% and 22% is in the
843 same range as existing solutions based on the safe k -grouping concept [15].
844 From the results, we can observe that there is no significant difference on the
845 average relative error between the anonymous bipartite graphs, produced by
846 the algorithm SNPPnR, and the corresponding anonymous bipartite graphs
847 produced, by algorithm SNPP1R, and the ones, produced by adapted so-
848 lution A5. One reason of the little variation between them is the random
849 characteristics when sampling consistent bipartite graphs. Another reason is
850 the effect of the noise nodes added by the algorithm SNPPnR. The algorithm
851 SNPPnR preserves the utility because it adds noise nodes as the number of
852 edges increases, therefore adding noise nodes may degrade the utility; while
853 using a bipartite graph, which contains a large number of edges, allows an-
854 swering queries with more accuracy. However we notice that (especially in
855 Figure 8.c) the algorithm SNPPnR and A5 give an average relative error
856 more important than the algorithm SNPP1R, one reason is that the utility
857 cost function, used in the algorithm SNPP1R to group the nodes, preserves
858 the utility better than the one used in the algorithm SNPPnR and A5. The
859 other reason is that the algorithm SNPP1R anonymizes each release inde-
860 pendently, unlike the algorithms A5 and SNPPnR that consider the previous
861 releases, when anonymizing the current one.
862 We also find that the average relative error increases with k , which satisfy
863 the intuition, since sampling consistent bipartite graphs from an anonymous
864 one with large k , leads to more significant error than the one with small k .
865 However, at several points, such as the point $k = 6$ in Figure 9.b, the average
866 relative error suddenly decreases a little. One reason is the algorithm random
867 characteristics, used for sampling consistent bipartite graphs, which might
868 give better result for a large k .
869 Figures 10.a and 10.b show the average relative error of type1 and type2
870 queries, respectively, over the different releases and by taking $k = 10$. The
871 results show that the average relative error for algorithms A5 and SNPPnR
872 increases with the number of releases. The reason is the algorithms A5 and
873 SNPPnR produce the current release based on the previous one. The other

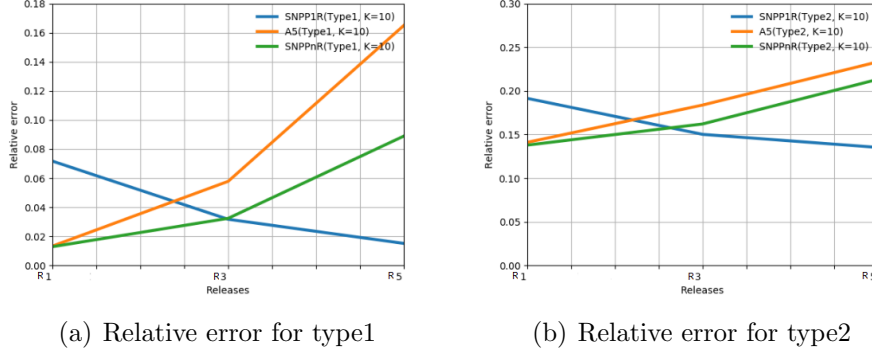


Figure 10: Relative error for different releases. **a**: Average relative error of type 1 queries over the different releases (R1, R3 and R5), **b**: Average relative error of type 2 queries over the different releases (R1, R3 and R5).

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

879 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:

$$NSafe = NSafePermut + NSafeNew + NSafeDelet \quad (3)$$

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

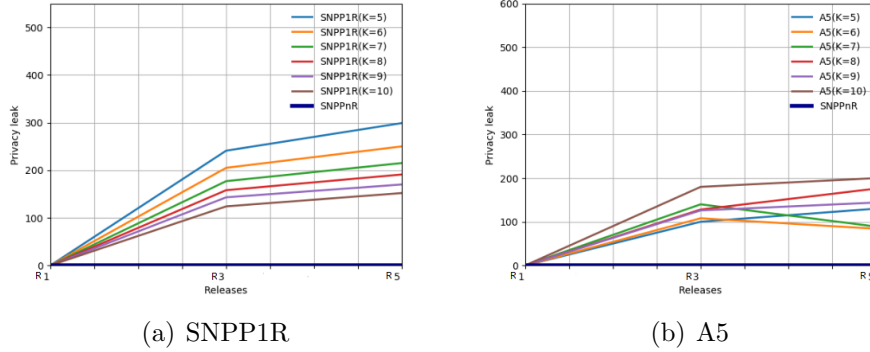


Figure 11: Privacy leak. **a**: Privacy leak engendered by the algorithm SNPP1R, **b**: Privacy leak engendered by the algorithm A5.

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:

$$Vleakage = Vclasses * 2k \quad (4)$$

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

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

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

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

898 A5 increases with k , because the privacy leakage measures the number of
 899 nodes (user and group) in each class their privacy is affected. As the size of
 900 classes increases with k , then the number of nodes their privacy is affected
 901 increases. However, at several points, such as the point $k = 9$, the privacy
 902 leakage decreases suddenly a little. The reason, is that the grouping of nodes
 903 in classes, in the previous releases, with large k resists better to the social
 904 networks evolution (the edges adding) than the nodes grouping with small
 905 k .

906 Notice that the algorithm SNPPnR depicts no privacy leakage due to com-
 907 parison of different releases, as it proved in section 5.

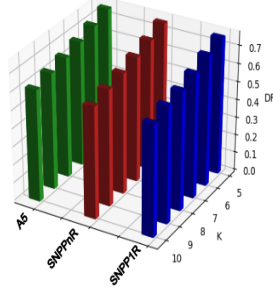
908 6.7. SNPPnR privacy evaluation measurement

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:

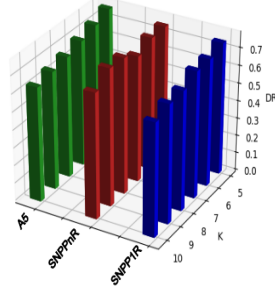
$$DR_X(Y) = 1 - \frac{H(X/Y)}{H(X)} \quad (5)$$

909 Where $H(X)$ is the entropy of X and $H(X/Y)$ is the conditional entropy.
 910 The $DR_X(Y)$ measures the capability of the attribute Y to identify the at-
 911 tribute X . It is scaled between 0 and 1. The greater the $DR_X(Y)$, the greater
 912 the identification, and lower the privacy. For example, if $DR_X(Y)$ equals to
 913 1, it means that the values of the attribute Y permits to completely identify
 914 the values of the attribute X ; hence the privacy of X is completely disclosed.
 915 We use the DR metric to measure the real users' privacy, groups' privacy
 916 and edges' privacy of the three solutions, SNPPnR, SNPP1R and A5. So, to
 917 measure the:

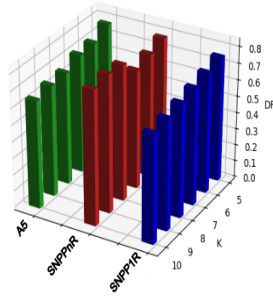
- 918 • users' privacy: we measure $DR_X(Y)$ where X is the user node identifier
 919 and Y is the attributes list of the user node. We measure the iden-
 920 tity user de-anonymization attack (as we measure the capability of the
 921 values of the attributes lists to identify the user node).
- 922 • groups' privacy: we measure $DR_X(Y)$ where X is the group node iden-
 923 tifier and Y is the attributes list of the group node. We measure the
 924 identity group de-anonymization attack (as we measure the capability
 925 of the values of the attributes lists to identify the group node).



(a) users' privacy for R1



(b) users' privacy for R3

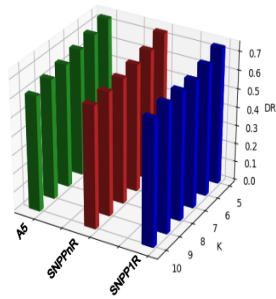


(c) users' privacy for R5

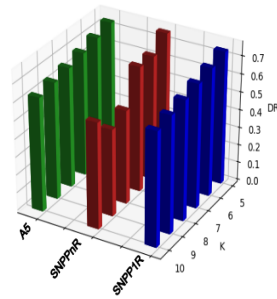
Figure 12: DR metric for users' privacy. **a**: DR metric of the release R1, **b**: DR metric of the release R3, **c**: DR metric of the release R5.

- edges' privacy: we measure $DR_X(Y)$ 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).

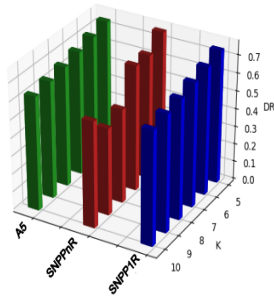
Figures 12, 13 and 14 show the users' privacy, the groups' privacy and the edges' privacy of the bipartite graphs R1, R3 and R5, produced by the algorithm SNPP1R, SNPPnR and A5 respectively. the DR metric of our solution, SNPPnR, which ranges between 41% and 86% is in the same range as the 15-anonymity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



(a) groups' privacy for R1

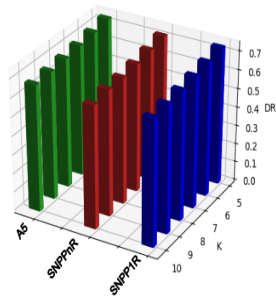


(b) groups' privacy for R3

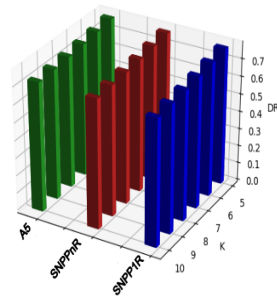


(c) groups' privacy for R5

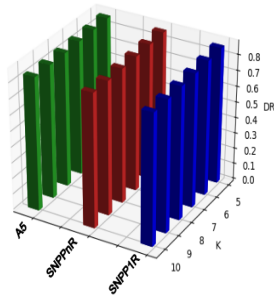
Figure 13: DR metric for groups' privacy. **a**: DR metric of the release R1, **b**: DR metric of the release R3, **c**: DR metric of the release R5.



(a) edges' privacy for R1



(b) edges' privacy for R3



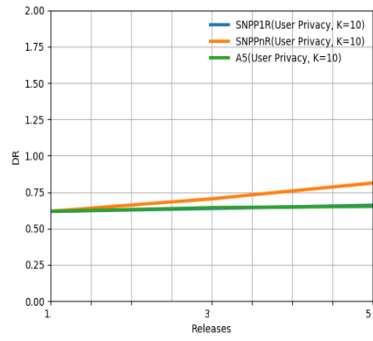
(c) edges' privacy for R5

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.

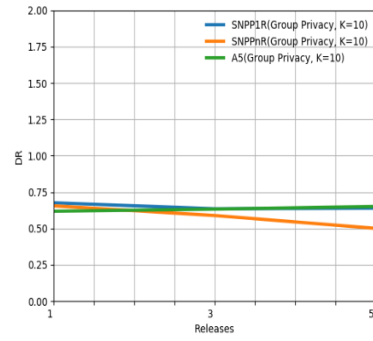
936 tice that the DR metric decreases with the increasing of k , which satisfies
 937 the intuition, as large k leads to more privacy. However, in Figures 12 and
 938 13, we notice that the DR metric for users' and groups' privacy knows little
 939 increase with k (e.g. Figure 13.b, $k = 10$). The reason is the grouping of
 940 nodes in classes with large k preserves better the utility which leads to less
 941 privacy. The other reason is with large k , the algorithm SNPPnR creates
 942 less noise groups which lead to less privacy.

943 The results (Figures 12, 14, and 13.a) show also that there is no significant
 944 difference between the privacy level, offered by our solution, SNPPnR, and
 945 the others (SNPP1R and A5). The reason, is that we have used the same
 946 anonymization technique in the three algorithms. However in Figures 13.b
 947 and 13.c, that represent the groups' privacy in the release R3 and R5, re-
 948 spectively, we notice that our solution the algorithm, SNPPnR, outperforms
 949 the algorithms SNPP1R and A5. The reason is that the algorithm SNPPnR
 950 creates a noise groups to ensure the privacy in sequential releases (see the
 951 algorithm 3). As the groups' privacy is $\leq \frac{1}{\lambda_y k}$ (see **RQ₂**) and as the created
 952 noise groups have the same attributes lists values like the real groups. Then,
 953 the average values of λ_y , in the anonymous bipartite graphs, created with
 954 the algorithm SNNPnR is larger than those created by the algorithm A5 and
 955 SNPP1R. So, we get a better groups' privacy.

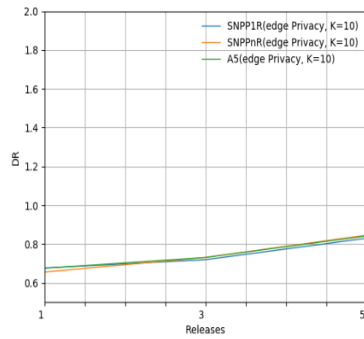
956 Figures 15.a, 15.b and 15.c show the users' privacy, the groups' privacy and
 957 the edges' privacy, over the different releases and by taking $k = 10$. Figures
 958 15.a and 15.b show also that the DR metric (for users' privacy and groups'
 959 privacy), of the algorithms A5 and SNPP1R, does not vary a lot with the
 960 number of releases (R1, R3 and R5). One reason is that with the algorithm
 961 A5 and SNPP1R, we do not modify the number of nodes (groups and users).
 962 Figure 15.a shows also that the DR metric (for users' privacy) of the al-
 963 gorithm SNPPnR increases a little across releases. The reason is that the
 964 algorithm SNPPnR, when creating classes, groups similar user nodes in the
 965 same class which leads to less privacy. The Figure 15.b shows that the DR
 966 metric (for groups privacy)of the algorithm SNPPnR decreases with releases
 967 (which lead to more privacy). One reason is the effect of the noise groups,
 968 created by the algorithm SNPPnR to remove the violation of the condition
 969 **SPa₃**. Figure 15.c shows that the DR metric for edges' privacy knows little
 970 increase with the number of releases. The reason is the evolution of the social
 971 networks, i.e. the bipartite graph in R5 contains more edges and nodes than
 972 the ones in R3 and R1. This leads to less privacy.



(a) users' privacy for k=10

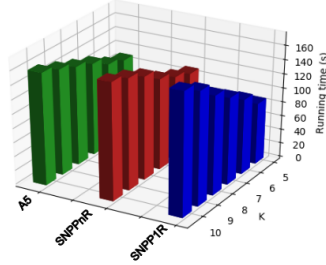


(b) groups' privacy for k=10

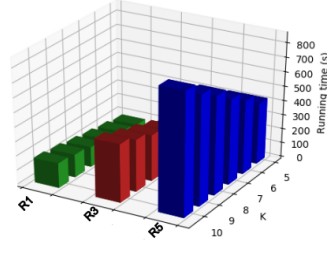


(c) edges' privacy for k=10

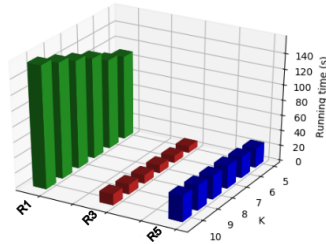
Figure 15: DR metric over the different releases. **a**: DR metric for users' privacy over the different releases (R1, R3 and R5), **b**: DR metric for groups' privacy over the different releases (R1, R3 and R5), **c**: DR metric for edges' privacy over the different releases (R1, R3 and R5).



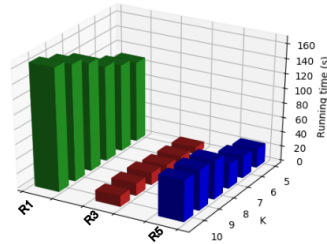
(a) R1



(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.

973 *6.8. Run time*

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

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

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

987 Figures 16.b, 16.c and 16.d show the run time to produce the anonymous
988 bipartite graphs R1, R3 and R5, using the algorithms SNPP1R, A5 and
989 SNPPnR, respectively. From the results, we can notice that the running
990 time of the algorithm SNPP1R increases with the releases which satisfy the
991 intuition as the size of bipartite graphs increases with releases. The results
992 show also that the running time needed by the algorithm SNPPnR and the
993 algorithm A5 to produce the bipartite graphs R3 and R5 is shorter than the
994 running time needed by the algorithm SNPP1R. For example for the point
995 (R3, $k = 5$) the algorithm SNPPnR needs only 9.4 seconds and the algorithm
996 A5 needs 8.3 seconds while the algorithm SNPP1R needs 204 seconds. The
997 reason is that the algorithm SNPPnR and the algorithm A5 group only the
998 new nodes while the algorithm SNPP1R groups all the nodes. We notice also
999 that the algorithm A5 needs less time than the algorithm SNPPnR. The jus-
1000 tification is that the algorithm A5 does not make any processing to remove
1001 the violation of the condition **SPa**₃.

1002 6.9. λ_x and λ_y impact on utility

1003 Recall that $\lambda_x - 1$ represents the number of users' nodes that have the
1004 same attributes list values and which are published at the same time as node
1005 u_x and $\lambda_y - 1$ is the number of groups' nodes that share the same attributes
1006 list values and published at the same time as node u_y .

1007 To study the impact of the values of parameters λ_x and λ_y on the utility, we
1008 generate two bipartite graphs, G1 and G2, with different values of λ_x and λ_y .
1009 We use the algorithm SNPPnR to produce the anonymous bipartite graphs.
1010 Table 3 gives the number of nodes, edges and the average values of λ_x and
1011 λ_y for each bipartite graph.

1012 Figure 17 shows the average relative errors of type1 and type2 queries for
1013 bipartite graphs G1, and G2. Each point in the figure is the expected answer
1014 of type1 and type2 queries on the corresponding anonymous bipartite graph
1015 of 6 privacy levels, using the algorithm SNPPnR. The results show that the
1016 relative error in the bipartite graph G1 is larger than the one in the bipartite
1017 graph G2. The reason is that in the bipartite graph G2, the average values
1018 of λ_x and λ_y are larger than the ones in the bipartite graph G1, so, as we
1019 have large average values of λ_x and λ_y , in G2, we get better utility because
1020 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

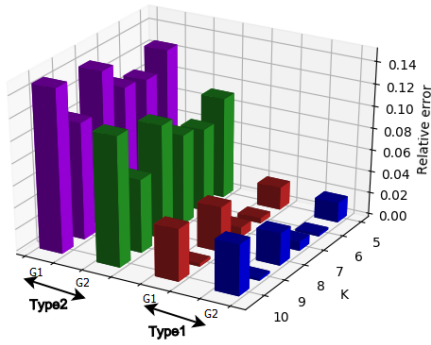


Figure 17: λ_x and λ_y impact on utility.

1021 attributes list values.

1022

1023 6.10. Nodes/edges addition/deletion impact on utility

1024 Recall that the algorithm SNPPnR delays the publication of new nodes if
1025 the number of these new nodes is less than the privacy level, k . The algorithm
1026 SNPPnR delays the suppression of nodes if no new ones can replace them
1027 (new nodes that have the same attributes list values as those to be deleted).
1028 Also, the algorithm adds noise nodes to avoid the violation of the *safely*
1029 *partitioning condition* (due to the edges addition between old classes).

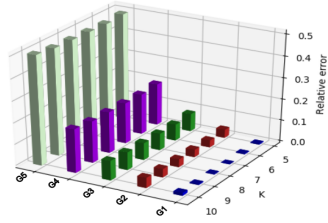
1030 As the number of new nodes in large social networks is usually greater than
1031 k , this section studies the impact on the utility of the delayed deleted nodes
1032 and that of the noise nodes due to the edges addition between old classes.

1033 To study the impact of delayed deleted nodes, we generate a graph G1, and
1034 we produce the graphs G2, G3, G4, and G5 by adding/deleting nodes/edges
1035 from G1. 5% nodes are deleted from G1 to G2, 10% nodes from G1 to
1036 G3, and 20% nodes from G1 to G4, and 50% nodes from G1 to G5. the
1037 characteristics of graphs G1, G2, G3, G4, and G5 are shown in Table 4.

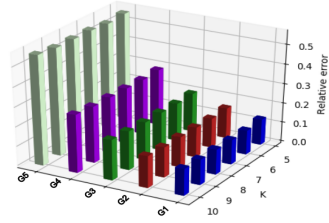
1038 Figure 18 shows the average relative errors of type1 and type2 queries for
1039 bipartite graphs G1, G2, G3, G4 and G5. Each point in the figure is the
1040 expected answer of type1 and type2 queries on the corresponding anonymous
1041 bipartite graph of 6 privacy levels using the algorithm SNPPnR. The results
1042 show that the relative error type1 and type2 increases with the number of
1043 deleted nodes (5% in G2, 10% in G3, 20% in G4, 50% in G5) which satisfy the
1044 intuition. The nodes that should be deleted remain in the next graphs (G2,
1045 G3, G4, G5), until the arrival of new nodes that can replace them. These
1046 delayed deleted nodes are considered as noise data, thus leading to less utility.
1047 Figure 18 shows that the relative error (type1 and type2) in graph G4 (where
1048 there are 20% deleted nodes) ranges between 19% and 29%, which is in the
1049 same range as existing solutions based on the safe k -grouping concept [15].

1050 As a result, our solution can preserve the utility even though the number of
1051 deleted nodes from one release to the next one is higher (20% deleted nodes).

1052 To study the impact of noise nodes, due to adding new edges between old
1053 classes), we generate a graph G1, and we produce the graphs G6, G7, G8, and
1054 G9 by adding/deleting nodes/edges from G1 as follows: 5% edges are added
1055 between old classes in G1 to produce G6, 10% edges between old classes in
1056 G1 to produce G7, 20% edges between old classes in G1 to produce G8, and
1057 50% edges between old classes in G1 to produce G9. The characteristics of

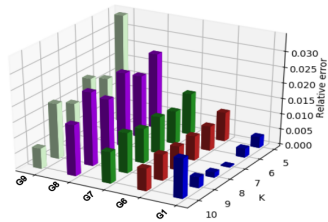


(a) Relative error for type 1

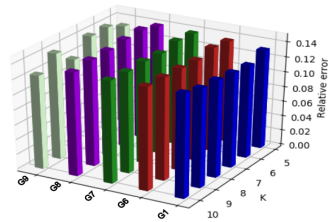


(b) Relative error for type 2

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

1058 graphs G1, G6, G7, G8, and G9 are shown in Table 5.
 1059 Figure 19 shows the average relative errors of type1 and type2 queries for
 1060 bipartite graphs G1, G6, G7, G8 and G9. Each point in the figure is the
 1061 expected answer of type1 and type2 queries on the corresponding anonymous
 1062 bipartite graph of 6 privacy levels, using the algorithm SNPPnR. The results
 1063 show that the relative error type1 and type2 (especially type 2) does not
 1064 vary between graphs (i.e. with the edges addition). The reason is that the
 1065 algorithm SNPPnR adds noise nodes proportionally to the number of added
 1066 edges (to suppress the violation of the *safely partitioning condition*). As a
 1067 consequence, although the addition of noise nodes may decrease the utility,
 1068 the use of graphs which contain a big number of edges leads to better utility
 1069 (i.e. the big number of edges hides the impact of noise nodes). As such,
 1070 our solution can preserve the utility even though the number of added edges
 1071 from one release to the next one is high.
 1072 As a consequence, the SNPPnR algorithm preserves the utility in the case of
 1073 high dynamic social networks.

1074 *6.11. Big social networks*

1075 Recall that the privacy preserving property of our solution, SNPPnR, is
 1076 proved in section 5.
 1077 This subsection studies the utility, offered by our solution in the case of large
 1078 social networks.
 1079 We used the random data generator to generate data sets. The first original
 1080 social network bipartite graph has 101770 nodes, and 309980 edges. To
 1081 generate the next bipartite graphs for the next releases, we take the previous
 1082 bipartite graph and we randomly remove and add nodes and edges, so that
 1083 the number of nodes and edges raises from one release to another by rate

1084 between 10% and 20% [29] (see Table 6).

1085 As these graphs are large, to generate the anonymous bipartite graphs, we
1086 use a modified version of the algorithm SNPPnR by removing the utility cost
1087 function from the algorithm SNPPnR for the following reasons:

- 1088 • Tests on social networks of reasonable sizes show that taking into ac-
1089 count the utility cost function does not bring much for the utility ac-
1090 curacy (see Appendix B).
- 1091 • The running time of the algorithm SNPPnR without the utility cost
1092 function is more efficient (see section 5.6). For example, the algorithm
1093 SNPPnR without the utility cost function requires only 16 seconds to
1094 anonymize a graph with 5000 nodes and 16900 edges while the algo-
1095 rithm SNPPnR with the utility cost function requires 278 minutes.
1096 Also, the algorithm SNPPnR without the utility cost function requires
1097 only 131 minutes to anonymize a graph with 101770 nodes and 309980
1098 edges (see Appendix C).

1099 As such, for testing large social networks, we decide to remove the utility
1100 cost function from the algorithm SNPPnR. We also use modified versions of
1101 the algorithms A5 and SNPP1R (Without utility cost function) to evaluate
1102 our solution (SNPPnR without utility cost function).

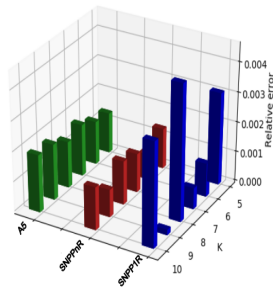
Table 6: BIPARTITE GRAPHS CHARACTERISTICS

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

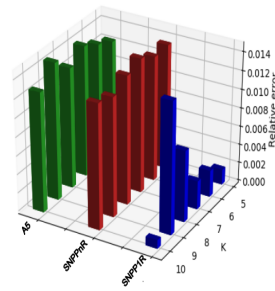
1103

1104 Figures 20 and 21 show the average relative errors of type1 and type2 queries
1105 for releases R1, R2 and R3, respectively. Each point in the figures is the ex-
1106 pected answer of type1 and type2 queries, on the corresponding anonymous
1107 bipartite graph of 6 privacy levels using the modified versions of the algo-
1108 rithms SNPP1R, A5 and SNPPnR (algorithms without the utility cost).

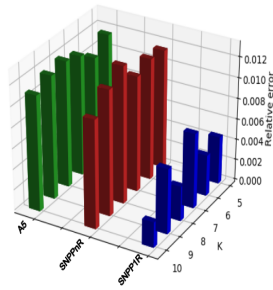
1109 We can notice that the obtained results share the same logic with those ob-
1110 tained in small social networks using the algorithms (SNPPnR, SNPP1, A5)



(a) R1

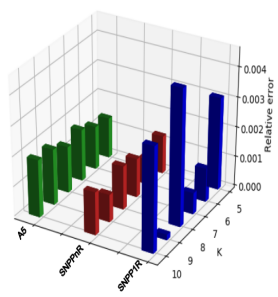


(b) R2

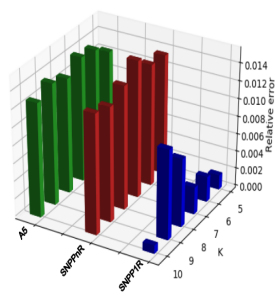


(c) R3

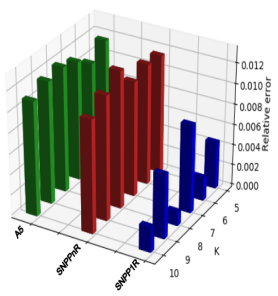
Figure 20: Average relative error for type 1. **a:** for release R1, **b:** for release R2, **c:** for release R3.



(a) R1



(b) R2



(c) R3

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

1111 with utility cost see section 6.5. Indeed, the results (especially in figures 20.b,
1112 20.c, 21.b and 21.c) show that the modified version of the algorithm SNPP1R
1113 outperforms the modified versions of the algorithms A5 and SNPPnR. The
1114 reason is that the modified version of the algorithm SNPP1R anonymizes
1115 each release independently, unlike the modified versions of the algorithms
1116 A5 and SNPPnR that consider the previous releases, when anonymizing the
1117 current one. From the results, we notice also that the modified version of
1118 the algorithm A5 outperforms a little bit the modified version of algorithm
1119 SNPPnR. The reason is the effect of the noise nodes created by the modi-
1120 fied algorithm SNPPnR to ensure the privacy. We notice that the effect of
1121 noise nodes is more important in big social networks than in small social
1122 networks, which can be explained by the high evolution (nodes/edges addi-
1123 tion/deletion) in big social networks.
1124 The results show that the relative error for type1 and type2, of the modified
1125 version of the algorithm SNPPnR, ranges between 0.05% and 2%, against
1126 the range 0.5% and 22% for small social networks using the SNPPnR with
1127 utility cost (see section 6.5). We can conclude that the modified version of
1128 the algorithm SNPPnR preserves well the utility in the case of large social
1129 networks despite the removal of the utility cost function which was originally
1130 introduced for preserving the utility within classes. The reason is that, in
1131 large graphs, the massive number of edges leads to better utility, the other
1132 reason is that the frequency of attributes lists values is higher in large social
1133 networks, i.e. the high probability to find several nodes with same attributes
1134 list values leads to fewer errors when sampling consistent bipartite graphs,
1135 thus resulting in high utility. As a consequence, our solution provides high
1136 utility accuracy in the case of big social networks, especially when the data
1137 receivers are interested only in a small number of attributes list.

1138 7. CONCLUSION

1139 In this paper, we propose a privacy preserving technique for labeled bi-
1140 partite social networks. This technique allows publishing sequential releases
1141 of the same social network while ensuring that the published data meet the
1142 privacy requirements and remain useful for data mining tasks. This technique
1143 is based on anonymization method that groups nodes in classes and masks
1144 the true mapping between nodes and attributes values. We prove the effec-
1145 tiveness of the proposed technique through several experiments. The latter
1146 demonstrated the privacy preserving of sequential releases while maintaining

1147 high utility of the published bipartite graphs and a reasonable overhead in
 1148 terms, of running time. We propose this solution to encourage social net-
 1149 work data publishing and helping new services useful for the whole society
 1150 to emerge, especially in health or education sectors.

1151 **Appendix A. safely permuting condition proof**

1152 We prove that the permutation between two nodes of the same type that
 1153 belong to two different classes, under the *safely permuting condition*, satisfies
 1154 the **RQ₁**, **RQ₂** and **RQ₃**.

1155 **Proof that the safely permuting condition satisfies the require-**
 1156 **ments RQ₁ and RQ₂:**

1157 We prove that the probability that an attacker can re-identify the node, u_x ,
 1158 in the published bipartite graph that represents the targeted individual (the
 1159 targeted group, respectively) is less than or equal to $\frac{1}{\lambda_x k}$ ($\frac{1}{\lambda_y k}$ respectively) .

1160 Let suppose that:

1161 At T_i , we have a published bipartite graph $\bar{G}_{T_i} = (X_{T_i}, Y_{T_i}, \bar{E}_{T_i}, L_{CX_{T_i}},$
 1162 $L_{CY_{T_i}})$ such as:

- 1163 • A node u_x is in class $C_x^{T_i} \in X_{T_i}$.
- 1164 • $C_y^{T_i} \in Y_{T_i}$ is class of groups.

1165 At T_{i+n} , we have a published bipartite graph $\bar{G}_{T_{i+n}}$ such as:

- 1166 • $C_{z1}, C_{z2}, \dots, C_{zn} \in X_{T_{i+n}}$ are classes that satisfy, with $C_x^{T_{i+n}}$, the
 1167 condition **SPe₁**.
- 1168 • There are $(\lambda_x - 1)$ nodes published, at the same time as u_x , having the
 1169 same attributes list values like u_x , and belonging to $(\lambda_x - 1)$ different
 1170 classes, $(C_{x1}, \dots, C_{x\lambda_x - 1}) \in X_{T_{i+n}}$, as required by the condition **SPa₂**.
- 1171 • There is a violation of the condition **SPa₃** between $C_x^{T_{i+n}}$ and $C_y^{T_{i+n}}$,
 1172 so to remove this violation, we must permute, under the property 2,
 1173 between nodes belonging to $C_x^{T_{i+n}}$ and C_{zi} .

1174 We find two cases:

- 1175 • The attacker assumes with probability $\frac{1}{2}$ that, the difference between
 1176 $\bar{G}_{T_{i+n}}$ and \bar{G}_{T_i} is due to the social network evolution, then in this case
 1177 the probability that an attacker knows that the node u_x is the one

1178 representing the targeted individual is: $P(u_x) = \frac{1}{2} * \frac{1}{\lambda_x} * \frac{1}{|C_x^{Ti+n}|}$ as the
 1179 size of each

1180 $C_{xi} \geq k$ then $P(u_x) \leq \frac{1}{2} * \frac{1}{\lambda_x k} < \frac{1}{\lambda_x k}$

1181 • The attacker assumes, with probability $\frac{1}{2}$, that the difference between
 1182 \bar{G}_{Ti+n} and \bar{G}_{Ti} is due to the suppression of the violation of the con-
 1183 dition **SPa₃** between C_x^{Ti+n} and C_y^{Ti+n} . So, she/he tries to find two
 1184 nodes, u_1 and u_2 , belonging to C_x^{Ti+n} and C_{zi} , respectively, such as
 1185 permuting these two nodes, u_1 and u_2 , violates the condition **SPa₃** be-
 1186 tween C_x^{Ti+n} and C_y^{Ti+n} (or increases the number of edges between
 1187 C_x^{Ti+n} and C_y^{Ti+n}): so

1188 – If there is a node, u_k : $u_k \in C_{zi}$, such as u_k .attributes list values = u_x .attributes
 1189 list values, and as:
 1190 $|C_x^{Ti+n}$.attributes lists values \cap C_{zi} .attributes lists values $|\geq \frac{k}{2}$, then the
 1191 probability that the permuted node u_1 is the targeted individual node, u_x , is
 1192 less than or equal to $\frac{2}{k}$, so the probability that an attacker knows that the
 1193 node u_x is the node representing the targeted individual is:

1194
$$P(u_x) \leq \frac{1}{2} * \frac{1}{\lambda_x} * \frac{2}{k} \leq \frac{1}{\lambda_x k}$$

1195 – If there is not a node, u_k : $u_k \in C_{zi}$ such as u_k .attributes list values =
 1196 u_x .attributes list values, then the probability that an attacker knows that
 1197 the node u_x is the targeted individual node is:

1198
$$P(u_x) = \frac{1}{2} * \left(\frac{1}{\lambda_x} * \frac{1}{|C_x^{Ti+n}| - 1} \right).$$

1199 as the size of each $C_{xi} \geq k$ then

1200
$$P(u_x) \leq \frac{1}{\lambda_x} * \frac{1}{2(k-1)} < \frac{1}{\lambda_x k} \quad \forall k \geq 2 \text{ then } P(u_x) < \frac{1}{\lambda_x k}$$

1201 So the *safely permuting condition* guarantees that the requirement **RQ₁**(users'
 1202 privacy) is satisfied while allowing nodes permutation.

1203 Similarly, we prove that the requirement **RQ₂** (groups' privacy) is guaran-
 1204 teed by the *safely permuting condition*.

1205

1206 **Proof that the safely permuting condition satisfies the requirement**
 1207 **RQ₃:**

1208 We prove that the probability that an attacker re-identifies that a link exists
 1209 between a node u_x , representing the targeted individual, and a node u_y , rep-
 1210 resenting the targeted group, is less than or equal to $\frac{1}{k}$. Let suppose that:

1211 At T_i , we have a published bipartite graph $\bar{G}_{Ti} = (X_{Ti}, Y_{Ti}, \bar{E}_{Ti}, L_{CXTi},$
 1212 $L_{CYTi})$ such as:

1213 • A node u_x is in class $C_x^{Ti} \in X_{Ti}$.

1214 • A node u_y is in class $C_y^{Ti} \in Y_{Ti}$.

1215 At T_{i+n} , we have a published bipartite graph $\bar{G}_{T_{i+n}}$ such as:

- 1216 • $C_{z1}, C_{z2}, \dots, C_{zn} \in X_{T_{i+n}}$ are classes that satisfy, with $C_x^{T_{i+n}}$, the
1217 condition **SPe₁**.
- 1218 • There are (λ_x-1) nodes published, at the same time as u_x , having the
1219 same attributes list values like u_x , and belonging to (λ_x-1) different
1220 classes $(C_{x1}, \dots, C_{x\lambda_x-1}) \in X_{T_{i+n}}$, as required by the condition **SPa₂**.
- 1221 • There are (λ_y-1) other nodes published at the same time as u_y , having
1222 the same attributes list values like u_y , belonging to (λ_y-1) different
1223 classes $(C_{y1}, \dots, C_{y\lambda_y-1}) \in Y_{T_{i+n}}$ as required by the condition **SPa₁**.
- 1224 • all nodes in $C_x^{T_{i+n}}, C_{x1}, \dots, C_{x\lambda_x-1}$ have links with nodes in $C_y^{T_{i+n}},$
1225 $C_{y1}, \dots, C_{y\lambda_y-1}$.
- 1226 • There is a violation of the condition **SPa₃** between $C_x^{T_{i+n}}$ and $C_y^{T_{i+n}}$,
1227 so to remove this violation, we must permute, under the property 2,
1228 between nodes belonging to $C_x^{T_{i+n}}$ and C_{zi} .

1229 We find two cases:

- 1230 • The attacker assumes with probability $\frac{1}{2}$ that, the difference between
1231 $\bar{G}_{T_{i+n}}$ and \bar{G}_{T_i} is due to the social network evolution, then in this case
1232 the probability that an attacker knows that a link, e , exists between
1233 node u_x and u_y is:
1234 $P(u_x, u_y, e) \leq \frac{1}{2} * \frac{1}{k} \leq \frac{1}{k}$: because the anonymization of the bipartite
1235 graph $\bar{G}_{T_{i+n}}$ is under the *safely partitioning condition*. So, the property
1236 1 is verified i.e. the requirement **RQ₃** is satisfied (the probability of
1237 edge's identification is less than or equal to $\frac{1}{k}$).
- 1238 • The attacker assumes, with probability $\frac{1}{2}$, that the difference between
1239 $\bar{G}_{T_{i+n}}$ and \bar{G}_{T_i} is due to the suppression of the violation of the condition
1240 **SPa₃** between $C_x^{T_{i+n}}$ and $C_y^{T_{i+n}}$, so she/he tries to find two nodes,
1241 u_1 and u_2 , belonging to $C_x^{T_{i+n}}$ and C_{zi} , respectively, such as permut-
1242 ing these two nodes, u_1 and u_2 , violates the condition **SPa₃** between
1243 $C_x^{T_{i+n}}$ and $C_y^{T_{i+n}}$ (or increases the number of edges between $C_x^{T_{i+n}}$
1244 and $C_y^{T_{i+n}}$). In this case, the probability that an attacker knows that
1245 a link, e , exists between nodes u_x and u_y is:
1246
$$P(u_x, u_y, e) = \frac{1}{2} * \left(\frac{1}{|C_x^{T_{i+n}}| + |C_{x1}| + \dots + |C_{x\lambda_x-1}|} * \frac{1}{|C_y^{T_{i+n}}| + |C_{y1}| + \dots + |C_{y\lambda_y-1}|} \right) * n_{edge}$$

1247 According to the condition \mathbf{SPa}_3 , the number of edges between any two
 1248 classes, C_{xi} and C_{yj} , must be $\leq k$ and as we permute two nodes between
 1249 C_x^{Ti+n} and C_{zi} , then in the worst case the replaced node in C_x^{Ti+n} has
 1250 k edges with C_y^{Ti+n} . Hence the number of edges between C_x^{Ti+n} and
 1251 $C_y^{Ti+n} \leq 2k$, so:

$$1252 \quad n_{\text{bedge}} \leq k(\lambda_x * \lambda_y + 1)$$

$$1253 \quad \text{Then: } P(u_x, u_y, e) \leq \frac{1}{2} * \frac{1}{\lambda_x k} * \frac{1}{\lambda_y k} * k(\lambda_x * \lambda_y + 1).$$

$$1254 \quad \text{So: } P(u_x, u_y, e) \leq \frac{1}{2} * \left(\frac{1}{k} + \frac{1}{\lambda_x \lambda_y k} \right), \text{ then } P(u_x, u_y, e) \leq \frac{1}{k}$$

1255 So the *safely permuting condition* guarantees the requirement \mathbf{RQ}_3 (edges'
 1256 privacy) while allowing nodes permutation.

1257 Appendix B. The utility cost function impact on the utility

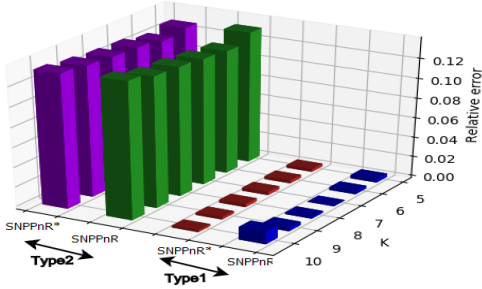


Figure B.21: The utility cost function impact on the utility accuracy

1258 This section studies the impact of the utility cost function on the utility
 1259 accuracy. So, we generate randomly a bipartite graph (G1) that contains
 1260 1000 nodes and 3360 edges and we use the algorithm SNPPnR and the al-
 1261 gorithm SNPPnR without utility cost function (SNPPnR*) to produce the
 1262 anonymous bipartite graph.

1263 Figure 22.B shows the average relative errors of type1 and type2 queries for
 1264 bipartite graph G1. Each point in the figure is the expected answer of type1
 1265 and type2 queries on the corresponding anonymous bipartite graph of 6 pri-
 1266 vacy levels, using the algorithm SNPPnR and the algorithm SNPPnR*. The
 1267 results show that the utility cost function does not impact much on the utility
 1268 as the relative error for type1 and type2 obtained by the algorithm SNPPnR

1269 and SNPPnR*, is mainly similar (only a little variation). The reason is that
 1270 the utility accuracy depends much more on the nature of the data (attributes
 1271 lists values). Indeed, i) if in the graph, the attributes values frequency is high,
 1272 then, even without considering the utility cost function, the probability of
 1273 putting nodes that share a big number of attributes values in the same class
 1274 is high. So, the algorithm SNPPnR* reaches a high utility accuracy. ii) if,
 1275 in the graph, the attributes values frequency is low, then, even though the
 1276 utility cost function is used, the utility accuracy will not enhance a lot, as
 1277 finally, the algorithm SNPPnR can put small number of nodes that share
 1278 attributes values together (as the attributes values frequency is low).

1279 **Appendix C. Running time of the algorithm SNPPnR without**
 1280 **utility cost function**

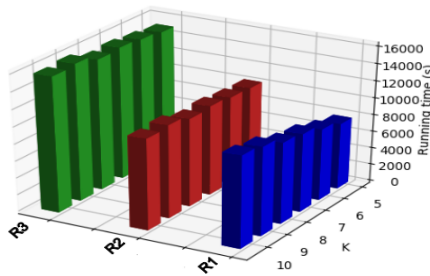


Figure C.22: the running time of the algorithm SNPPnR without utility cost function

1281 This section studies the running time of the algorithm SNPPnR without
 1282 utility cost function. So, we calculate the running time needed by the al-
 1283 gorithm SNPPnR without utility cost function to produce the anonymous
 1284 bipartite graphs of the original ones, presented in the section 6.11 (see Table
 1285 6).

1286 Figure 23.C shows that the algorithm SNPPnR without utility cost func-
 1287 tion produces an anonymous bipartite graph that contains 101770 nodes and
 1288 309980 edges in only 131 minutes. Figure 22.C shows also that the running
 1289 time increases with releases (R1, R2 and R3). The reason is that the bipar-
 1290 tite graphs in R2 and R3 contain a big number of new nodes that should
 1291 be grouped in new classes. The other reason is the running time needed

1292 to remove the old nodes (nodes that should be deleted) and the running
1293 time to suppress the violation of the *safely partitioning condition* (i.e. in the
1294 large social networks, we need to test a big number of classes to suppress the
1295 violation of the *safely partitioning condition*).

1296 **References**

- 1297 [1] J.Abawajy, M. Izuan, H. Ninggal, and T. Herawan. "Privacy Preserving
1298 Social Network Data Publication", in IEEE Communications Surveys and
1299 Tutorials, 2016.
- 1300 [2] R. Tackx , J.l. Guillaume, and F. Tarissan. "Revealing intricate proper-
1301 ties of communities in the bipartite structure of online social networks",
1302 in IEEE Ninth International Conference on Research Challenges in Infor-
1303 mation Science (RCIS), 2015.
- 1304 [3] M.Yuan, L.Chen, P.S.Yu, and T.Yu. "Protecting Sensitive Labels in So-
1305 cial Network Data Anonymization", in IEEE Transactions on Knowledge
1306 and Data Engineering, Volume 25, Issue 3 , pp. 633-647, 2013.
- 1307 [4] B. Zhou and J. Pei. "Preserving privacy in social networks against neigh-
1308 borhood attacks", in IEEE 24th International Conference on Data Engi-
1309 neering (ICDE), pp. 506-515, 2008.
- 1310 [5] Y. Wang, L. Xie, B. Zheng, and K. Lee. "High utility k-anonymization
1311 for social network publishing". in Knowledge and Information Systems,
1312 vol. 41, no. 3, pp. 697-725, 2014.
- 1313 [6] R. Okada, C. Watanabe, and H. Kitagawa. "A k-anonymization algo-
1314 rithm on social network data that reduces distances between nodes",
1315 in Reliable Distributed Systems Workshops (SRDSW), 2014 IEEE 33rd
1316 International Symposium on . IEEE, pp. 7681, 2014.
- 1317 [7] J.Cheng, A.W.Fu, and J.Liu, "K-isomorphism: privacy preserving net-
1318 work publication against structural attacks," in Pro-ceedings of the ACM
1319 International Conference on Management of Data (SIGMOD), pp. 459
1320 470, 2010.
- 1321 [8] Q. Liu, G. Wang; F. Li, S. Yang, and J. Wu, " Preserving Privacy with
1322 Probabilistic Indistinguishability in Weighted Social Networks", in IEEE

- 1323 Transactions on Parallel and Distributed Systems, Volume: 28, Issue: 5,
1324 pp. 1417 - 1429, 2017.
- 1325 [9] M. Hay, G. Miklau, D.Jensen, P. Weis, and S. Srivastava, "Anonymizing
1326 social networks," in Computer Science Department Faculty Publication
1327 Series. 180, pp. 1-17, 2007.
- 1328 [10] X. Ying and X. Wu. "Randomizing Social Networks: a Spectrum Pre-
1329 serving Approach", in SIAM International Conference on Data Mining
1330 (SDM), pp. 739-750, 2008.
- 1331 [11] P. Liu, L. Wang, and X. Li. "Randomized Perturbation for Privacy
1332 Preserving Social Network Data Publishing" , in the IEEE International
1333 Conference on Big Knowledge (ICBK), pp 208-2013, 2017.
- 1334 [12] M. Hay, G. Miklau, D. Jensen, D. F. Towsley, and C. Li. "Resisting
1335 structural reidentification in anonymized social networks", in Very Large
1336 Database Journal, 2010.
- 1337 [13] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. "Pre-
1338 diction Promotes Privacy In Dynamic Social Networks", in Workshop on
1339 Online Social Networks (WOSN), Boston, 2010.
- 1340 [14] T. Tassa and D. Cohen. "Anonymization of centralized and distributed
1341 social networks by sequential clustering," in IEEE Transactions on
1342 Knowledge and Data Engineering, 2012.
- 1343 [15] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava. "Class-
1344 based graph anonymization for social network data", in VLDB , 2009.
- 1345 [16] M. Yuan, L.Chen, and P.S. Yu. "Personalized Privacy Protection in
1346 Social Networks", in the VLDB Endowment, Vol. 4, 2011.
- 1347 [17] G.Cormode , D.Srivastava ,T.Yu , and Q. Zhang, "Anonymizing bipar-
1348 tite graph data using safe groupings", in the VLDB Journal , 2010.
- 1349 [18] H. Jiang, "A novel clustering-based anonymization approach for graph to
1350 achieve privacy preservation in social network", in International Confer-
1351 ence on Advances in Mechanical Engineering and Industrial Informatics.
1352 Atlantis Press, 2015.

- 1353 [19] F. Bonchi, A. Gionis, and T. Tassa. "Identity Obfuscation in Graphs
1354 Through the Information Theoretic Lens", in IEEE 24th International
1355 Conference on Data Engineering (ICDE), 2011.
- 1356 [20] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa. "Injecting uncertainty in
1357 graphs for identity obfuscation," in Proceedings of VLDB Endowment,
1358 Volume 5, No 11, 1376-1387, 2012.
- 1359 [21] H.H. Nguyen, A.Imine, and M. Rusinowitch. "Anonymizing Social
1360 Graphs via Uncertainty Semantics", in Proceedings of the 10th ACM
1361 Symposium on Information, Computer and Communications Security
1362 (ASIA CCS), 2015.
- 1363 [22] C. Borgs, J. T. Chayes and A. Smith. "Private Graphon Estimation
1364 for Sparse Graphs", in NIPS'15 Proceedings of the 28th International
1365 Conference on Neural Information Processing Systems, April 2015.
- 1366 [23] J. Blocki, A. Blum, A. Datta, and O. Sheffet. "Differentially Private
1367 Data Analysis of Social Networks via Restricted Sensitivity", in Innova-
1368 tions in (Theoretical) Computer Science (ICS), 2013.
- 1369 [24] Q.Wang, Y. Zhang, X.Lu , Z. Wang, Z. Qin, and K.Ren. "Real-time and
1370 Spatio-temporal Crowd-sourced Social Network Data Publishing with
1371 Differential Privacy", in IEEE Transactions on Dependable and Secure
1372 Computing, 2018.
- 1373 [25] C. M. Fung, K. Wang, A. Wai-Chee Fu, and P.S. Yu. "Introduction to
1374 Privacy-Preserving Data Publishing Concepts and Techniques", in Chap-
1375 man and Hall/CRC Data Mining and Knowledge Discovery Series, 2010.
- 1376 [26] E.Zheleva and L.Getoor. "Privacy in social networks: A survey", in
1377 Springer Science+Business Media, LLC 2011.
- 1378 [27] LP.Sondeck, M.Laurent, and V. Frey. "Discrimination rate: an
1379 attribute-centric metric to measure privacy", In: Annals of Telecommu-
1380 nications journal DOI:10.1007/s12243-017-0581-8, 2017.
- 1381 [28] C.Dwork. "Differential privacy", in Automata, languages and program-
1382 ming, pp. 1-12. Springer Berlin Heidelberg, 2006.

- 1383 [29] T.Wu, S.Yu, W.Liao, and C.Chang "Temporal Bipartite Projection and
1384 Link Prediction for Online Social Networks", in IEEE International Con-
1385 ference on Big Data, 2014.
- 1386 [30] F.Yu, M.Chen, B.Yu, W.Li, L.Ma, and H.Gao. "Privacy preservation
1387 based on clustering perturbation algorithm for social network", in Journal
1388 Multimedia Tools and Applications. 2018.
- 1389 [31] J.Yan, L.Zhang, Y.Tian, G.Wen, and J.Hu. "An Uncertain Graph Ap-
1390 proach for Preserving Privacy in Social Networks Based on Important
1391 Nodes", in International Conference on Networking and Network Appli-
1392 cations (NaNA). 2018.
- 1393 [32] X.Xiao, and Y.Tao. "m-invariance: Towards privacy preserving republi-
1394 cation of dynamic datasets", in Proceedings of the ACM SIGMOD Con-
1395 ference. ACM, New York. 2007.