Protecting Privacy in Knowledge Graphs with Personalized Anonymization

Anh-Tu Hoang, Barbara Carminati *Senior Member,* and Elena Ferrari *IEEE Fellow*

Abstract—Knowledge graphs (KGs) are emerging data models allowing data providers to share data. This data sharing might bring new knowledge and collaborations, with evident benefits for providers. However, since KGs might contain sensitive information about users, it is of utmost importance to ensure KG anonymization before publishing. Recently, some proposals have addressed the problem of KGs' anonymization based on the k-anonymity principle. These techniques propose to anonymize the whole dataset with the same anonymization level. However, in a contest where data are collected from different users, it is crucial to consider also users' preferences on the anonymization level to adopt for their data. To cope with this requirement, this paper presents the Personalized k-Attribute Degree (p-k-ad) principle. It allows users to specify their anonymity levels (the k values) while preventing adversaries from re-identifying them with a confidence higher than $\frac{1}{k}$ with their specified k. Moreover, we design the Personalized Cluster-Based Knowledge Graph Anonymization Algorithm (PCKGA) to generate anonymized KGs satisfying p-k-ad. We conduct experiments on four real-life datasets and show that PCKGA greatly improves the quality of anonymized KGs comparing to previous algorithms.

Index Terms—Knowledge graphs, personalized privacy, k-anonymity.

1 Introduction

Knowledge graphs (KGs) are getting popular among data providers to share users' properties and relationships. After the announcement of Google Knowledge Graph in 2012, many giant tech companies (i.e., Microsoft[1], Amazon[2]) announced their applications for KGs. Data providers share their KGs with data recipients (e.g., researchers, data analysts) for a variety of applications. For instance, to accelerate fraud detection tasks, Amazon provides Fraud Knowledge Graph[3] modelling properties and relationships of fraudsters. Since KGs may contain users' sensitive data, they must be anonymized before sharing them.

The problem of graph anonymization has been well investigated in the past by using two main techniques: k-anonymity and differential privacy (DP) [1]. k-anonymity follows the non-interactive setting where data providers modify users' relationships in their graphs such that adversaries cannot re-identify a user in the modified graphs with a confidence higher than $\frac{1}{k}$ even by exploiting background knowledge on the target victim. For instance, k-degree, k-neighborhood [1] modify the original graph such that the degree or neighbor structure of any user in the anonymized graph is indistinguishable from that of $k-1$ other users. Unfortunately, it has been shown in [2] that the above approaches are not enough for KGs since adversaries can exploit both attribute values and relationships.

On the other hand, DP anonymizes graphs under two settings: interactive and non-interactive. The former [3] creates a system interactively answering statistics queries from graphs while preventing adversaries from inferring the existence of a user (i.e., a node) by looking at the extracted statistics. The latter publishes graphs' statistics once while ensuring the same privacy protection as the former. For example, [3] introduces an algorithm to generate statistics of graphs and their nodes' attributes (e.g., the number and size of communities, the distribution of attributes' values in the communities') while protecting users' identities. [4] uses DP to inject noise to data transferred between clients in order to protect users' privacy in federated learning settings. Although DP solves some of the k-anonymity's limitations by not relying on assumptions about what information adversaries can exploit, data providers must design a different DP analysis algorithm for each type of statistics that data recipients require. Having separate statistics on the same KG might represent a limitation for KG analyses, as several exploit the correlation between KGs' attributes and relationships (e.g., drug design, tax calculations, or financial reporting). As an example, in [3], the extracted statistics on graphs' attributes (e.g., the distribution of communities' attributes) are not correlated to those of users' relationships (e.g., the size of communities). Moreover, many KGs' analyses are still not supported by DP (e.g., tax calculations). Therefore, to enhance users' privacy protection in the current developments of KGs' analyses without sacrificing shared KGs' utility, we focus on k-anonymity approaches.

Recently, some proposals appeared (e.g., [2], [5]) to extend k-anonymity to KGs. [5] has introduced an anatomy approach to generate groups of at least k users and returns the frequency of each sensitive value in these groups. However, this approach restricts the utility of KGs, since data recipients cannot use standard analytical tools (e.g., deep learning) to analyze KGs. k-Attribute Degree (k-ad) [2] prevents users from being re-identified with a confidence higher than $\frac{1}{k}$ even though adversaries exploit both attribute values and relationship out-/in-degrees. To this end, k-ad

• *A.T. Hoang (contact@tuhoang.me), B. Carminati (barbara.carminati@uninsubria.it), and E. Ferrari (elena.ferrari@uninsubria.it) are with the Department of Theoretical and Applied Science (DiSTA), University of Insubria, Italy.*

1. https://www.microsoft.com/en-us/research/project/microsoft-academic-graph/
2. https://aws.amazon.com/neptune/knowledge-graphs-on-aws/
3. https://aws.amazon.com/neptune/fraud-graphs-on-aws/

requires to add/remove edges such that the information of any user in the anonymized KG is indistinguishable from that of $k-1$ other ones in the KG. However, several studies in academia [6] and industry,[4], for instance those, based on Westin's privacy types [7], suggest that online users can be classified into groups (typically 4) with different levels of privacy concerns. Therefore, we believe that users' privacy types should also be considered when data are anonymized, by allowing each user to set up their privacy levels.

Few k-anonymity and DP proposals for relational data [8], [9], [10] and undirected graphs [11], [12] allow users to specify personalized preferences for their data anonymization. Liu et al. [8] allow users to specify two thresholds. The former prevents anonymized data to be more detailed than the threshold. The latter is specified for each sensitive value to ensure that a user cannot be associated with any sensitive value with a confidence higher than his/her specified threshold for the value. In [11], users can specify their preferences in anonymizing: (1) only their attribute values, (2) their attribute values and relationships, (3) their values, relationships, and neighbors. [12] presents a similar approach but its supported preferences are: (high) anonymizing both users' sensitive values and neighbors, (medium) anonymizing only neighbors, and (low) no anonymization. [9], [10] are DP-based approaches allowing users to specify their own privacy parameters' values (i.e., ϵ) and data providers must consider all of the values when they generate statistics from the users' attributes in relational data. However, we are not aware of any proposals for KGs allowing users to set their personalized preferences (e.g., k). All of the state of the art privacy protection proposals for KGs [2], [5] apply the same k value to all users. A naive solution to make each user able to specify his/her k value is to select the maximum k among all specified values, but this might result in poor data quality.

Therefore, in this paper, we present the Personalized k-Attribute Degree (p-k-ad) principle, to protect users' identities even if multiple k's values are specified. p-k-ad requires that, for every user u in an anonymized KG, his/her attribute values and relationship out-/in-degrees are indistinguishable from those of at least k_u-1 other users in the KG, where k_u is the k value set by u. Thus, adversaries cannot re-identify a user u with a confidence higher than $\frac{1}{k_u}$.

To generate anonymized KGs satisfying the p-k-ad principle, following the approach presented in [2], we design the Personalized Cluster-Based Knowledge Graph Anonymization Algorithm (PCKGA), which generates anonymized KGs according to two steps: *Clusters Generation* and *Knowledge Graph Generalization*. The former allows data providers to specify their own clustering algorithm (e.g., k-medoids [13], HDBSCAN [14]) to generate clusters. The latter generates anonymized KGs such that the anonymized attribute values and relationships' out-/in-degrees of users in the same clusters are identical. We formally prove that if the clusters' sizes are greater than or equal to the maximum k values of their users, the generated anonymized KGs satisfy p-k-ad principle. Here, the anonymized attribute values of users in a cluster are the union of their attribute values. The out-/in-degrees of the anonymized relationships of the users are the maximum out-/in-degrees of the users' relationships. To optimize the quality of anonymized KGs, we must minimize (1) the difference between attribute values and relationships' out-/in-degrees of users in the same clusters, and (2) the disparity among their k values. (2) guarantees that users with smaller k values are not anonymized with those with excessively high k values, that leads to high information loss. Since these operations are done in the *Clusters Generation* step, in this paper, we focus on designing this step, and use the *Knowledge Graph Generalization* step that we proposed in [2].

Although allowing providers to choose their clustering algorithms increases flexibility, standard clustering algorithms do not fit the scenario of personalized anonymization since they only minimize the distance among users without considering their k values. This might bring to have clusters with very similar users but with too different k's values, thus resulting in high information loss. To address this challenge, we propose a new clustering algorithm, called the Varying-Anonymity Clustering Algorithm (VAC), that can be used in the first step of PCKGA instead of current clustering algorithms. VAC measures the *anonymization distance* between two users by using not only their distance w.r.t. their attributes and degrees, but also their k values. VAC generates clusters such that the anonymization distances among users in the same cluster are minimized. Moreover, it removes users whose k values or distances to other users are too high. To ensure that sizes of obtained clusters are greater than or equal to the maximum among k values of their users, we develop the Merge-Split Algorithm (MS). MS modifies clusters generated from the clustering algorithm by removing invalid clusters, whose size is less than the required k. Then, MS adds users belonging to the invalid clusters to nearest valid clusters such that and all clusters are still valid. We formally prove that no matter what clustering algorithm data providers exploit, by using MS, we always generate anonymized KGs satisfying the p-k-ad principle.

We conduct experiments by using four real-life datasets and compare anonymized KGs' quality generated by executing PCKGA with various settings: VAC, state-of-the-art clustering algorithms (i.e., k-medoids [13], HDBSCAN [14]) with appropriate parameters' values (i.e., $max/min/mean$ users' k values), and a simple anonymization approach using HDBSCAN (i.e., HDB*). HDB* gathers users whose k values are equal into groups and anonymizes users in the same group with their users' k value. The experimental results show that executing PCKGA with VAC results in anonymized KGs whose information loss is 210% lower than that of those generated by k-medoids and HDBSCAN and 26% lower than that of HDB*. Furthermore, our results indicate that PCKGA performance are good enough to be used in practice. Our experiments also show that PCKGA outperforms the previous anonymization algorithm for KGs [2] and relational data [15].

The remainder of this paper is organized as follows. Section 2 explains the adversary knowledge and our protection model. Anonymization algorithms are presented in Section 3, while Section 4 explains their details. Privacy guarantees of our proposal are analyzed in Section 5. We illustrate the experimental results in Section 6 and conclude

4. https://www.cisco.com/c/dam/global/en_uk/products/collateral/security/cybersecurity-series-2019-cps.pdf

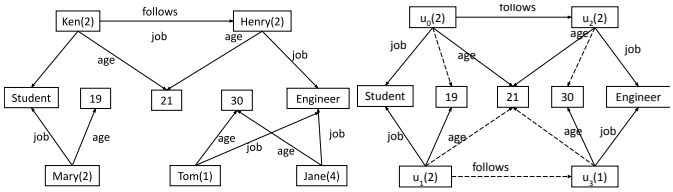
our paper in Section 7. Due to the lack of space, the summary of notations, the complexity analysis, proofs of theorems in Section 5, and some experimental results are included in the supplementary material section.

2 Personalized Anonymization of KGs

In this section, we introduce a new privacy principle to support KG anonymization with personalized k values. Before that, we briefly introduce KGs. We refer the reader to the supplementary material section for a summary of the notations used in this section and throughout paper.

2.1 Knowledge Graph

We model a KG as a graph $G(V, E, R)$, where V, E, R are the set of nodes, edges connecting these nodes, and relationship types of these edges, respectively. Since this work focuses on protecting users' privacy in KGs, we categorize the set of nodes V into the set of users V^U and the set of attribute values V^A, where $V^U \cup V^A = V$. Relationship types in R are categorized into two subsets: user-to-user relationship types R^{UU}, representing users' relationships (e.g., $follows$), and user-to-attribute relationship types R^{UA}, modelling users' attributes (e.g., age). Thus, $R = R^{UU} \cup R^{UA}$. We model each edge $e \in E$ as a tuple (u, r, v), where $u, v \in V$ and $r \in R$. We denote with $E^{UA} \subseteq E$ those $e = (u, r_a, v_a)$, such that $r_a \in R^{UA}$, and $E^{UU} \subseteq E$ those $e = (u, r_r, v_r)$, such that $r_r \in R^{UU}$. Fig. 1(a) illustrates an example of KG.



(a) The original KG G. (b) The p-k-ad anonymized KG $\overline{G}$.

Fig. 1. Knowledge graphs (dashed lines denote added fake edges). k values of *Ken* (u_0), *Mary* (u_1), *Henry* (u_2), *Tom* (u_3), and *Jane* (u_4) are 2, 2, 2, 1, and 4, respectively. $\overline{G}$ is generated from two clusters $\{u_0, u_1\}$ and $\{u_2, u_3\}$ while u_4 is removed due to its high k value.

2.2 Personalized k-Attribute Degree Principle

This section introduces the Personalized k-Attribute Degree principle. We start by characterizing the knowledge that an adversary can exploit to perform re-identification attacks.

Let $G(V, E, R)$ be a KG, and $\overline{G}(\overline{V}, \overline{E}, \overline{R})$ be its anonymized version created by modifying G. Given a target user $\overline{u} \in \overline{V}^U$, the adversary's goal is to re-identify $\overline{u}$ by using the background knowledge he/she has on $\overline{u}$ and the information he/she can extract from $\overline{G}$. We formally define the adversary knowledge as follows.

Definition 1 (Adversary Knowledge) Let $\overline{G}$ be an anonymized KG. The knowledge that an adversary can use to re-identify a user $\overline{u} \in \overline{V}^U$, denoted as $\mathcal{AK}(\overline{u})$, consists of:

- attribute values and relationship out-/in-degrees that the adversary knows about $\overline{u}$;
- attribute values and relationship out-/in-degrees of user $\overline{u}$ in $\overline{G}$, denoted as $\mathcal{I}(\overline{G}, \overline{u}) = \{\mathcal{I}_a(\overline{G}, \overline{u}), \mathcal{I}_o(\overline{G}, \overline{u}), \mathcal{I}_i(\overline{G}, \overline{u})\}$, where:
  - $\mathcal{I}_a(\overline{G}, \overline{u}) = \{(r_a, v_a) | (\overline{u}, r_a, v_a) \in \overline{E}^{UA}\}$ is the set of attributes' values contained in $\overline{G}$ associated with $\overline{u}$;
  - $\mathcal{I}_o(\overline{G}, \overline{u}) = \{(r_r, d_o(\overline{G}, r_r, \overline{u})) | r_r \in \overline{R}^{UU}\}$ and $\mathcal{I}_i(\overline{G}, \overline{u}) = \{(r_r, d_i(\overline{G}, r_r, \overline{u})) |\ r_r \in \overline{R}^{UU}\}$ are $\overline{u}$'s out- and in-degree in $\overline{G}$, for all relationship types, respectively; being $d_o(\overline{G}, r_r, \overline{u}) = |\{(\overline{u}, r_r, v_r) \in \overline{E}^{UU}\}|$ and $d_i(\overline{G}, r_r, \overline{u}) = |\{(v_r, r_r, \overline{u}) \in \overline{E}^{UU}\}|$ the out-/in-degree of relationship type r_r of $\overline{u}$.

Example 1. Let us consider the anonymized KG $\overline{G}$ shown in Fig. 1b. Since u_0 has one outgoing relationship of type $follows$ with u_2 and no incoming relationships of the same type, $\mathcal{I}_o(\overline{G}, u_0) = \{(follows,\ 1)\}$; $\mathcal{I}_i(\overline{G}, u_0) = \{(follows, 0)\}$. The knowledge about u_0 that an adversary can extract from $\overline{G}$ is $\mathcal{I}(\overline{G}, u_0) = \{\{(age, 19), (age, 21), (job, Student)\}, \{(follows, 1)\}, \{(follows, 0)\}\}$.

The Personalized k-Attribute Degree principle is defined as follows.

Definition 2 (Personalized k-Attribute Degree) Let $\overline{G}(\overline{V}, \overline{E}, \overline{R})$ be an anonymized KG. $\overline{G}$ satisfies the Personalized k-Attribute Degree (p-k-ad) if and only if for every user $\overline{u}$ in $\overline{V}^U$, there is a set $\overline{\mathcal{C}}(\overline{G}, \overline{u}) \subseteq \overline{V}^U$ such that $\overline{\mathcal{C}}(\overline{G}, \overline{u}) = \{v \in \overline{V}^U | \mathcal{I}(\overline{G}, \overline{u}) = \mathcal{I}(\overline{G}, v)\}$ and $|\overline{\mathcal{C}}(\overline{G}, \overline{u})| \geq k_{\overline{u}}$, where $k_{\overline{u}}$ is a positive integer specified by $\overline{u}$.

Fig. 1b illustrates the anonymized version of G (Fig. 1a) satisfying p-k-ad. In this work, similarly to the state-of-the-art personalized k-anonymity [8], [11], [12] and DP proposals [9], [10], we assume that users' personalized k values are public. However, as point out in [10] the privacy parameters could be used to infer personal information. For instance, a user might select high parameters' values based on his job/function (e.g., politician). So, attackers could infer user's job based on selected values (e.g., politician could be inferred by high values). To address this concern, we follow [10] to assume that there is no relationship between selected privacy values and any sensitive values embedded in anonymized KGs. Therefore, even though adversaries exploit the public k values of target users, they cannot increase the confidence of re-identifying users. Privacy protection guarantees of p-k-ad will be described in Section 5.

3 Personalized Cluster-Based Knowledge Graph Anonymization

This section introduces the overall idea for the generation of anonymized KGs satisfying p-k-ad (see Section 2.2). We propose the Personalized Cluster-Based Knowledge Graph Anonymization (PCKGA) algorithm (Algorithm 1) that exploits clustering to generate anonymized data. PCKGA takes as input a KG: G; a clustering algorithm: $\mathcal{A}$; the set of anonymity levels (k values) expressed by each user in G: $\mathcal{K}$; a threshold: τ; and it generates the anonymized version of G, i.e. $\overline{G}$, ensuring the p-k-ad principle and maximizing $\overline{G}$'s quality. In general, adopting clustering techniques for k-anonymization implies generating clusters containing at least k similar users, that is, whose profile data are not distant according to some distance metrics. Then, the k-anonymity principle is satisfied by anonymizing

users belonging to the same cluster with the same generalization (e.g., same attribute values and relationship out-/in-degrees). Therefore, PCKGA contains two main steps: *Clusters Generation* (lines 1-4) and *Knowledge Graph Generalization* (line 5). The first step generates clusters by calling the clustering algorithm $\mathcal{A}$ received as input with the anonymization distance matrix $\mathcal{D}^a$ and its parameters $\mathcal{P}$[5] (lines 1-3). Subsequently, these clusters are adjusted to ensure their validity, which requires that the number of users in each cluster is greater than or equal to the maximum k value of their users (line 4). Then, it calls the Knowledge Graph Generalization Algorithm (KGG) [2] to generalize attributes and relationships (line 5). Finally, it returns $\overline{G}$ (line 6).

Algorithm 1 PCKGA(G, $\mathcal{A}$, $\mathcal{K}$, τ)

Input: G: the original KG, $\mathcal{A}$: a clustering algorithm, $\mathcal{K}$: k values of users in G; τ: a threshold.
Output: The anonymized KG $\overline{G}$.
1: Let $\mathcal{D}^a$ be the anonymization distance matrix of users in G.
2: $\mathcal{P} \leftarrow$ Calculate the parameters' values for $\mathcal{A}$.
3: $\mathcal{C} \leftarrow \mathcal{A}(\mathcal{D}^a, \mathcal{P})$
4: $\overline{\mathcal{C}} \leftarrow \text{MS}(\mathcal{D}^a, \mathcal{C}, \mathcal{K}, \tau)$
5: $\overline{G} \leftarrow \text{KGG}(G, \overline{\mathcal{C}})$
6: **return** $\overline{G}$

Clusters generation: Literature offers several clustering algorithms (e.g., k-Medoids [13], HDBSCAN [14]), proposed for the k-anonymization of both relational and non-relational data that can be used by our PCKGA algorithm. Clusters generation rely on the definition of the anonymity level of a cluster, formally defined in what follows.

Definition 3 (Cluster anonymity level) Let c be a cluster of users, where each user $u \in c$ has specified an anonymity level, denoted as $k_u \in \mathbb{N}$. The anonymity level of cluster c, denoted as k^c, is defined as the maximum value among the anonymity levels specified by users in c, i.e., $k^c = \max_{u \in c} k_u$.

The PCKGA's cluster generation step creates only *valid clusters*, that is, clusters c whose number of users, i.e., $|c|$, is greater than or equal to its anonymity level. This is done by the Merge-Split Algorithm (MS) (Algorithm 3) that possibly refines the output of the provided clustering algorithm $\mathcal{A}$ so that all the generated clusters are valid.

Although PCKGA can work with any clustering algorithm, previously defined algorithms do not fit well our scenario of personalized anonymization, since they minimize the distance among users without considering the values of k they select. This could result in clusters with very similar users but with very different k's values, bringing therefore the risk of big clusters. Indeed, even if only a few users specify high k values, the cluster has to be enlarged so that its cardinality is greater than or equal to the maximum among all k's values specified by its users. To cope with this issue, we propose a new clustering algorithm for PCKGA, called the Varying Minimum-Size Constraint Clustering Algorithm (VAC) (Algorithm 2). VAC generates clusters useful for personalized k-anonymization.

Knowledge graph generalization: The previous step generates only clusters whose cardinality is always greater than or equal to the maximum value among the anonymity levels specified by their users. This satisfies only partially the requirements imposed by the p-k-ad principle introduced by Definition 2. In particular, it ensures that, for each user in $\overline{G}$, there exists a set of users $\overline{\mathcal{C}}(\overline{G}, u)$, i.e., a cluster, such that $|\overline{\mathcal{C}}(\overline{G}, u)| \geq k_u$. To fully satisfy p-$k$-ad principle, we have also to ensure that all users in $\overline{\mathcal{C}}(\overline{G}, u)$ have identical attributes' values and the same out-/in-degree for all relationship types.

For this purpose, we exploit the Knowledge Graph Generalization Algorithm (KGG) presented in [2], which has been designed to satisfy the k-Attribute Degree (k-ad) principle [2]. This principle requires that, each user within the anonymized graph must have a minimum of $k-1$ other users having the same attribute values and out-/in-degree for all relationship types. To this end, KGG first extracts the union of the attribute values of all users in the target cluster. Subsequently, it generalizes attribute values of users in the cluster by adding fake user-to-attribute edges to make the users' values identical to those in the union. To generalize the user-to-user relationships of users in a cluster, KGG finds the maximum out-/in-degree of all relationship types of the users belonging to the cluster. It then increases the users' out-/in-degrees to match the maximum degrees by adding user-to-user edges. If it is impossible to add edges, it reduces the maximum out-/in-degree of the relationship types by removing edges of the users whose degrees are equal to the maximum ones. It then continue adding user-to-user edges.

However, KGG works under the assumption that k is unique for the whole graph. Applying KGG in the context of PCKGA implies executing it on each cluster c generated by PCKGA's first step with k set to k^c.

Example 2. Let $c_0 = \{u_0, u_1\}$, $c_1 = \{u_2, u_3\}$ be the clusters generated by the Clusters Generation step from the original KG G (Fig. 1a). Since their anonymity levels (i.e., $k^{c_0} = \max\{2,2\} = 2$ and $k^{c_1} = \max\{2,1\} = 2$) are higher than or equal to their cardinality, they are valid. KGG adds 3 fake edges, namely, $(u_0, age, 19)$, $(u_1, age, 21)$, $(u_1, follows, u_3)$ to make attributes and relationships' out-/in-degrees of users in c_0 (i.e., u_0, u_1) identical. Similarly, it adds 2 fake edges, that is, $(u_2, age, 30)$, $(u_3, age, 21)$ to make those of c_1's users (i.e., u_2, u_3) identical. The resulting anonymized KG is showed in Fig. 1b.

In the following section, we will focus on the clusters generation step, which is the one impacted by supporting personalized k values. We refer interested readers to [2] for more details on KGG.

4 Clusters Generation

The Cluster Generation step generates a set of valid clusters. It first calculates the parameters' values $\mathcal{P}$ used to execute the clustering algorithm $\mathcal{A}$ (line 2, Algorithm 1) and then executes the clustering algorithm $\mathcal{A}$ with the calculated parameters $\mathcal{P}$ (line 3, Algorithm 1). For instance, in the case of VAC, $\mathcal{P}$ includes $\mathcal{K}$: k's values of users in G. We explain how we set parameters' values for other clustering algorithms (i.e., k-Medoids, HDBSCAN) in Section 6.2. This step creates a preliminary set of clusters $\mathcal{C}$. Then, we run the Merge-Split algorithm (MS) that merges and splits clusters in $\mathcal{C}$ to generate the set of valid clusters $\overline{\mathcal{C}}$ (line 4, Algorithm 1).

We recall that even though data providers can use any clustering algorithm as $\mathcal{A}$, we propose a new clustering

5. We explain how $\mathcal{P}$ is calculated according to each clustering algorithm in Section 6.2.

algorithm, namely VAC, which takes into account users' k values. The main advantage of VAC is that providers only need to specify an anonymization distance matrix and users' k values. In contrast, if k-medoids is used, it requires specifying the number of generated clusters, whereas HDBSCAN needs the minimum size for all generated clusters. Even though we present VAC as the clustering algorithm $\mathcal{A}$ that generates clusters for PCKGA, the providers may have different needs and want to develop their own clustering algorithms. In such cases, the providers can still use their own algorithms with PCKGA without any modification.

VAC aims to generate clusters while minimizing: (1) the maximum distance between users in the same cluster, and (2) the differences among these users' k values. The key idea is the definition of a novel distance metric that considers both (1) and (2). In the rest of the section, we first present the proposed distance metric, then VAC and MS.

4.1 Distance measure

The proposed distance measure aims to estimate the anonymization cost of a cluster by considering its users' k values. Given a user u and his/her k value, k_u, we first estimate the cost of finding $k_u - 1$ users that are most similar to u. Hereafter, we refer to these users as *u's nearest neighbors*, denoted as $N(u)$. To measure the similarity between two users and thus computing $N(u)$, we use the Attribute and Degree Information Loss (ADM) Distance [2], denoted as d_{adm}. This measure incorporates three components: the attributes' distance (d_{am}), the distance of the target users' out-degree (d^o_{dm}), and the distance of their in-degree (d^i_{dm}).

d_{am} estimates the information loss of generalizing attribute values of user u (denoted as $AL(u,v)$), and those of user v (denoted as $AL(v,u)$) to make attribute values' of u and v equal. Given an attribute r_a, the information loss of generalizing r_a's values for a user u such that the values are equal to those of a user v is the differences between u's generalized attribute values (denoted in what follows as $GV(G,u,v,r_a)$) and u's original values (i.e., $\mathcal{I}_a(G,r_a,u)$, $\mathcal{I}_a(G,r_a,v)$). If r_a is a categorical attribute, the differences are the number of values added by the generalization step (i.e., $|GV(G,u,v,r_a) \setminus \mathcal{I}_a(G,r_a,u)|$). If r_a is a numerical attribute, the differences are the changes of minimum and maximum values after generalizing r_a's values (i.e., $\min GV(G,u,v,r_a) - \min \mathcal{I}_a(G,r_a,u)$ and $\max GV(G,u,v,r_a) - \max \mathcal{I}_a(G,r_a,u)$). d_{am} then takes the average of the information loss of u and v. d_{am} is formally defined as follows:

Definition 4 (Attribute Information Loss Distance [2]) Let u, v be two users in a KG G. The Attribute Information Loss Distance (d_{am}) measuring the information loss of making u's and v's attribute values identical is computed as follows:

$$d_{am}(u,v) = \frac{AL(u,v) + AL(v,u)}{2}$$

$$AL(u,v) = \frac{1}{|R^{UA}|} \times \sum_{r_a}^{R^{UA}} \begin{cases} AL_c(u,v,r_a), \text{if } r_a \text{ is categorical} \\ AL_n(u,v,r_a), \text{if } r_a \text{ is numerical} \end{cases}$$

$$AL_c(u,v,r_a) = \frac{|GV(G,u,v,r_a) \setminus \mathcal{I}_a(G,r_a,u)|}{|dom(r_a) \setminus \mathcal{I}_a(G,r_a,u)| + 1}$$

$$AL_n(u,v,r_a) = \frac{|\mathcal{I}^{min}_a(u,v,r_a)| + |\mathcal{I}^{max}_a(u,v,r_a)|}{|\mathcal{I}^{min}_{a_{dom}}(u,r_a)| + |\mathcal{I}^{max}_{a_{dom}}(u,r_a)| + 1}$$

$$\mathcal{I}^M_a(u,v,r_a) = M(GV(G,u,v,r_a)) - M(\mathcal{I}_a(G,r_a,u))$$

$$\mathcal{I}^M_{a_{dom}}(u,r_a) = M(dom(r_a)) - M(\mathcal{I}_a(G,r_a,u))$$

where M is either the max or min function, $\mathcal{I}_a(G,r_a,u) = \{v_a | (u,r_a,v_a) \in E^{UA}\}$, $GV(G,u,v,r_a) = \mathcal{I}_a(G,r_a,u) \cup \mathcal{I}_a(G,r_a,v)$, and $dom(r_a) = \{v_a | (u,r_a,v_a) \in E^{UA}\}$.

In contrast, d^o_{dm} and d^i_{dm} measure the information loss of user u and user v when generalizing their relationships. The information loss of the target users on a given relationship r_r is computed by finding the differences between users' generalized out-/in-degree (denoted as $GD_o(G,u,v,r_r)$, $GD_i(G,u,v,r_r)$) and original ones (denoted as $d_o(G,r_r,u)$, $d_i(G,r_r,u)$), respectively. Since the definition of d^o_{dm} and d^i_{dm} are similar, in what follows, we only present the formal definition of d^o_{dm}.

Definition 5 (Out-Degree Information Loss Distance [2]) Let u, v be two users in a KG G. The Out-Degree Information Loss Distance (d^o_{dm}) of making u and v having the same out-degrees on all relationship types is computed as follows:

$$d^o_{dm}(u,v) = \frac{DL_o(u,v) + DL_o(v,u)}{2}$$

$$DL_o(u,v) = \frac{1}{|R^{UU}|} \times \sum_{r_r}^{R^{UU}} \frac{GD_o(G,u,v,r_r) - d_o(G,r_r,u)}{|V^U|}$$

where $GD_o(G,u,v,r_r) = \max\{d_o(G,r_r,u), d_o(G,r_r,v)\}$.

By combining the above defined distances, d_{adm} estimates the information we lose on generalizing two users u and v's attributes and relationships. The higher the distance between two users is, the less similar they are. d_{adm} is formally defined as follows:

Definition 6 (Attribute and Degree Information Loss Distance [2]) Let u, v be two users in a KG G. The Attribute and Degree Information Loss Metric (ADM) of making u and v having the same values on all attributes, and the same out-/in-degree on all relationship types is computed as follows:

$$d_{adm}(u,v) = \frac{d_{am}(u,v) + 0.5 \times d^o_{dm}(u,v) + 0.5 \times d^i_{dm}(u,v)}{2}$$

where d_{am}, d^o_{dm}, and d^i_{dm} are the attribute, out-degree, and in-degree information loss distance between u and v, formally defined in Definitions 4-5.

Thus, the cost of finding $k_u - 1$ users similar to u is defined as $cost(u) = \max_{v \in N(u)} d_{adm}(u,v)$. We refer to this cost as *u's core distance*.

Example 3. Considering Fig. 1a, the nearest neighbors of each user are: $N(u_0) = \{u_1\}$, $N(u_1) = \{u_0\}$, $N(u_2) = \{u_3\}$, $N(u_3) = \varnothing$, $N(u_4) = \{u_1, u_2, u_3\}$. $N(u_3)$ is empty since u_3 does not need any neighbors to be in a valid cluster ($k_{u_3} = 1$). Therefore: $cost(u_0) = cost(u_1) = \max\{d_{adm}(u_0,u_1)\} = \max\{0.108\} = 0.108$; $cost(u_2) = \max\{d_{adm}(u_2,u_3)\} = \max\{0.108\} = 0.108$; $cost(u_3) = 0$; $cost(u_4) = \{d_{adm}(u_4,u_1), d_{adm}(u_4,u_2), d_{adm}(u_4,u_3)\} = \max\{0.208, 0.108, 0\} = 0.208$.

Given a user u, his/her core distance allows us to determine the minimum information loss that a cluster containing u will have to accommodate enough similar users

to satisfy k_u constraint. By using the user core distance, we can now define the anonymization distance between two users u and v as the cost to generate the smallest valid cluster around them. To build a valid cluster we have to insert a number of users equal to the maximum between k_u and k_v. Moreover, to minimize the cost, we can insert u's and v's nearest neighbors (i.e., $N(u), N(v)$). Thus, the cost estimation of the obtained cluster depends on the maximum of distances between its users. This can be seen as the maximum among: the core distance of u (i.e., the maximum distance between u and its $N(u)$), the core distance of v (i.e., the maximum distance between v and its $N(v)$), and their ADM distance (i.e., $d_{adm}(u, v)$). The total cost is given by the maximum distance of its users multiplied by the number of its users, this latter set as the maximum of k_u and k_v. This multiplication allows us to measure not only the anonymity level of the valid cluster containing both u and v but also the distances between the cluster's users. The higher either their distances or their anonymity levels is, the higher their anonymization distance is.

Definition 7 (Anonymization Distance) Let u, v be two users in a KG G, and k_u k_v be their corresponding k values. The anonymization distance between u, v is defined as:

$$d^a(u, v) = \max\{cost(u), cost(v), d_{adm}(u, v)\} \times \max\{k_u, k_v\}$$

Example 4. Given the KG in Fig. 1a, $d^a(u_0,\ u_1) = \max\{cost(u_0),\ cost(u_1),\ d_{adm}(u_0,\ u_1)\} \times \max\{k_{u_0},\ k_{u_1}\} = \max\{0.108,\ 0.108,\ 0.108\} \times \max\{2,\ 2\} = 0.216$, whereas $d^a(u_3,\ u_4) = \max\{cost(u_3),\ cost(u_4),\ d_{adm}(u_3,\ u_4)\} \times \max\{k_{u_3},\ k_{u_4}\} = \max\{0, 0.208, 0\} \times \max\{1, 4\} = 0.833$. Note that since u_4 has high anonymity level, even though the ADM distance of u_3 and u_4 is 0, their anonymization distance (i.e., 0.833) is higher than that of u_0 and u_1 (i.e., 0.216) whose ADM distance is 0.108.

We exploit the anonymization distance to compute the cost of u's anonymization. Intuitively, each user u must be added to a valid cluster that have at least $k_u - 1$ other users. To minimize the information loss of anonymizing user u, his/her cluster must include u and his/her $k_u - 1$-nearest neighbors. So, the minimum anonymization cost of user u can be considered as the maximum anonymization distances among users in the cluster. We define the cost as follows.

Definition 8 (User Anonymization Cost) Let u be a user in a KG G. The anonymization cost of u, denoted as $cost(u)$, is the maximum anonymization distance with his/her $k_u - 1$-nearest neighbors $N(u)$, that is, $cost(u) = \max_{v \in N(u)} d^a(u, v)$.

Example 5. Considering Example 4, the anonymization costs of the involved users are: $cost(u_0) = cost(u_1) = \max\ \{d^a(u_0,\ u_1)\} = \max\ \{0.216\} = 0.216$; $cost(u_2) = \max\ \{d^a(u_2,\ u_3)\} = \max\ \{0.216\} = 0.216$; $cost(u_3) = 0$; $cost(u_4) = \max\ \{d^a(u_4,\ u_1),\ d^a(u_4,\ u_2),\ d^a(u_4,\ u_3)\} = \max\{0.833,\ 0.833,\ 0.833\} = 0.833$.

4.2 The Varying-Anonymity Clustering Algorithm

The Varying-Anonymity Clustering Algorithm (VAC) (see Algorithm 2) receives as input the anonymization distances computed on each pair of users in G: $\mathcal{D}^a$; the set of users k values: $\mathcal{K}$; and returns a set of clusters built by leveraging on the user anonymization cost previously defined. Distances are modelled as a matrix, called *distance matrix*, $\mathcal{D}^a \in \mathbb{R}^{|V^U| \times |V^U|}$, whose element $\mathcal{D}^a[i, j]$ contains $d^a(u_i, u_j)$, $u_i, u_j \in V^U$. The usage of the distance matrix allows our algorithm to be compatible with state-of-the-art clustering libraries.[6] In case the original KG is too big and it is impractical to calculate the matrix, VAC's scalability can easily be improved by calculating distances at run-time and caching them in a database.

Algorithm 2 VAC($\mathcal{D}^a$, $\mathcal{K}$)

Input: $\mathcal{D}^a$: the anonymization distance matrix of users in the original KG G; $\mathcal{K}$: k values of users in G.
Output: The set of clusters $\mathcal{C}$.

1: Let $\mathcal{U}_{asc}$ be the sorted set containing users in V^U sorted in ascending order by their anonymization cost
2: $\mathcal{C} \leftarrow \varnothing$
3: **while** $|\mathcal{U}_{asc}| \geq k_{\mathcal{U}_{asc}[0]}$ **do**
4: $\quad u \leftarrow get\_and\_remove\_at\_index(\mathcal{U}_{asc}, 0)$
5: $\quad c \leftarrow find\_best\_cluster(u, \mathcal{D}^a, \mathcal{U}_{asc}, \mathcal{K})$
6: $\quad add(\mathcal{C}, c)$
7: **end while**
8: **return** $\mathcal{C}$

Function 1 $find\_best\_cluster\ (u, \mathcal{D}^a, \mathcal{U}_{asc}, \mathcal{K})$

1: Let U_c be the sorted array of users in U_{asc} who are sorted in ascending order by the distance between them and u.
2: $c \leftarrow [u]$
3: $k^c \leftarrow k_u$
4: $i \leftarrow 0$
5: **while** $k^c - |c| > 0 \wedge i < |U_c|$ **do**
6: $\quad add(c, U_c[i])$
7: $\quad k^c \leftarrow \max\{k^c, k_{U_c[i]}\}$
8: $\quad remove(\mathcal{U}_{asc}, U_c[i])$
9: $\quad i \leftarrow i + 1$
10: **end while**
11: **return** c

To minimize the distances between users, VAC utilizes the approach of prioritizing the creation of clusters for users with lower anonymization costs over those with higher costs. More precisely, the algorithm first sorts all users in V^U in ascending order by their anonymization cost and stores the sorted users in a sorted set $\mathcal{U}_{asc}$ (line 1). A sorted set ensures the uniqueness of users and the efficiency of users' addition/removal as well as of the retrieval of the smallest-anonymization-cost user (i.e., $\mathcal{U}_{asc}[0]$). The set of clusters $\mathcal{C}$ is initialized as the empty set (line 2). While $\mathcal{U}_{asc}$'s size is higher than or equal to the k value of the first user in $\mathcal{U}_{asc}$, the algorithm starts generating clusters (lines 3-7). It removes the first element in $\mathcal{U}_{asc}$ and stores the element in u (line 4). u is passed to function $find\_best\_cluster()$ to create its best valid cluster c, and insert c into $\mathcal{C}$ (line 5-6). Once the cluster has been created, the function removes from $\mathcal{U}_{asc}$ all users in c, so that in the next while iteration a new user is considered. After the while cycle ends, if $\mathcal{U}_{asc}$ still contains some users, they will not be included in any cluster because the cluster containing them is invalid (i.e., $|\mathcal{U}_{asc}| < k_{\mathcal{U}_{asc}[0]}$). In Section 6, we will show how these removed users impact the obtained information loss. Finally, the algorithm returns the set of generated clusters $\mathcal{C}$ (line 8).

Function *find_best_cluster()*. Given a user: u, the anonymization distance matrix: $\mathcal{D}^a$, the set of users: $\mathcal{U}_{asc}$, and k values of users in G: $\mathcal{K}$, this function finds a cluster for u such that the maximum anonymization distance between

6. Scikit-Learn: https://scikit-learn.org/

pairs of users in the found cluster is minimized. First, it sorts users in $\mathcal{U}_{asc}$ in ascending order by their anonymization distance to u and stores them in the array U_c (line 1). Then, it initializes a cluster c with only user u, whereas the cluster anonymity level k^c is set to u's anonymity level (lines 2-3). It initializes i as 0 (line 4). While c is not valid (i.e., $k^c - |c| > 0$) and U_c still has users (i.e., $i < |U_c|$), it starts finding u's nearest users to add into c (lines 5-10). In each iteration, the function adds the i-th user in U_c (i.e., $U_c[i]$) to c (line 6). Since U_c is sorted in ascending order by the anonymization distance to u, the added user is the one closest to u. Then, it updates k^c (line 7), removes the user from $\mathcal{U}_{asc}$ (line 8), and increase i by 1 (line 9). Finally, it returns cluster c (line 11).

Example 6. VAC initializes $\mathcal{U}_{asc} = [u_3, u_0, u_1, u_2, u_4]$ according to their costs, calculated in Example 5. First, it finds u_3's cluster as $\{u_3\}$, since u_3's anonymity level is 1. Then, VAC considers u_0. The cluster for u_0 is $\{u_0, u_1\}$, since u_1 is u_0's closest user. VAC removes these users from $\mathcal{U}_{asc}$, and creates the last cluster $\{u_2, u_4\}$ for the remaining users. Then, all these clusters are returned. Note that, the last cluster is invalid and it will be modified by the MS algorithm, explained in the next section.

The complexity of VAC is $\mathcal{O}(n^2 \log n)$, where n is the number of users in V^U.[7]

4.3 The Merge-Split Algorithm

The set of clusters $\mathcal{C}$ generated by the provided clustering algorithm $\mathcal{A}$ is possibly modified such that only valid clusters are passed to the generalization phase. This is done by the Merge-Split Algorithm (MS) (see Algorithm 3). This algorithm takes as input the anonymization distance matrix: $\mathcal{D}^a$; a set of clusters: $\mathcal{C}$; k values of users in G: $\mathcal{K}$; and a threshold: τ. It returns the set of clusters $\overline{\mathcal{C}}$ such that, for each $c \in \overline{\mathcal{C}}$, the number of its users is: $|c| \geq k^c$ (i.e., c is valid); $|c| \leq 2 \times k^c$. The second condition ensures that the MS algorithm does not generate too large clusters. This assurance allows MS to reduce the greatness of the generalized attributes and the out-/in-degrees of relationships. Thus, the number of fake edges needed for generalization also decreases.

Algorithm 3 first selects from $\mathcal{C}$ only the valid clusters (i.e., C_{valid}), while users in the other clusters are collected into U (lines 1-9). Then, it calls Procedure *assign_valid_clusters()* to assign these users to valid clusters in C_{valid} (line 10). As detailed later, each user $u \in U$ is inserted into a valid cluster c where the maximum anonymization distance between u and other users in c is less than or equal to the maximum distance calculated by threshold τ. Then, Algorithm 3 refines each cluster c in C_{valid} by splitting it if it is too large, that is, if c's size is higher than or equal to twice its anonymity (lines 12-21). If this is the case, Algorithm 3 calls Function *split_big_cluster()* to split c into a set of smaller clusters whose sizes are greater than or equal to k^c and less than $|c|$ (line 16). The obtained set of clusters C_c is then added to C_{valid} (line 17). Otherwise, Algorithm 3 adds c to $\overline{\mathcal{C}}$ (line 19). Finally, it returns $\overline{\mathcal{C}}$ (line 22).

Procedure *assign_valid_clusters()*. This procedure first calculates the maximum anonymization distance τ_d, based on the maximum and minimum anonymization distance of users and threshold τ (lines 1-2). Then, for each user u in U, it searches for a cluster $c \in C_{valid}$ where the maximum anonymization distance between u and other users in c is less than or equal to τ_d and c is still valid after adding u. If it exists, u is assigned to c, otherwise it will be removed. In particular, for every user u in U, it initializes min_d to $+\infty$ (line 4) and c_{min} to the empty set (line 5). Then, for each c in C_{valid}, it calculates the maximum distance between the users in c and u, i.e., d (lines 7-10). If d is less than min_d, the number of users in c after adding u is greater than or equal to the maximum anonymity of c and u, and d is less than or equal to τ_d, it updates min_d (line 13) and c_{min} (line 14). If there is a cluster c_{min} in C_{valid} satisfying the above conditions, it adds u to c_{min} (line 18). Then, it continues finding a cluster for the next user in U (line 3).

Example 7. Let $\{u_3\}$, $\{u_0, u_1\}$, $\{u_2, u_4\}$ be the clusters generated in Example 6 and suppose τ has been set to 0.5. Since $d^a_{max} = 0.933$, $d^a_{min} = 0.216$, MS sets $\tau_d = \tau \times (d^a_{max} - d^a_{min}) + d^a_{min} = 0.5 \times (0.933 - 0.216) + 0.216 = 0.575$. It initializes $C_{valid} = \{\{u_3\}, \{u_0, u_1\}\}$ and $U = \{u_2, u_4\}$, since cluster $\{u_2, u_4\}$ is invalid. Next, it starts finding valid clusters for users in U. It merges u_2 to cluster $\{u_3\}$ as the

Algorithm 3 MS($\mathcal{D}^a, \mathcal{C}, \mathcal{K}, \tau$)

Input: $\mathcal{D}^a$: the anonymization distance matrix of users in G; $\mathcal{C}$: the set of clusters returned from the cluster generation phase; $\mathcal{K}$: k values of users in G; and τ: a threshold.
Output: A set of valid clusters $\overline{\mathcal{C}}$.

1: $C_{valid} \leftarrow \varnothing$
2: $U \leftarrow \varnothing$
3: **for** $c \in \mathcal{C}$ **do**
4: **if** $|c| \geq k^c$ **then**
5: $add(C_{valid}, c)$
6: **else**
7: $add\_array(U, c)$
8: **end if**
9: **end for**
10: $assign\_valid\_clusters(C_{valid}, U, \mathcal{D}^a, \mathcal{K}, \tau)$
11: $\overline{\mathcal{C}} \leftarrow \varnothing$
12: **while** $C_{valid} \neq \varnothing$ **do**
13: Let c be a cluster in C_{valid}.
14: $remove(C_{valid}, c)$
15: **if** $|c| \geq k^c \times 2$ **then**
16: $C_c \leftarrow split\_big\_cluster(\mathcal{D}^a, c, \mathcal{K})$
17: $remove\_set(C_{valid}, C_c)$
18: **else**
19: $add(\overline{\mathcal{C}}, c)$
20: **end if**
21: **end while**
22: **return** $\overline{\mathcal{C}}$

Procedure 1 $assign\_valid\_clusters$ (C_{valid}, U, $\mathcal{D}^a$, $\mathcal{K}$, τ)

1: Let d^a_{max}, d^a_{min} be the maximum and minimum distance between all users.
2: $\tau_d \leftarrow \tau \times (d^a_{max} - d^a_{min}) + d^a_{min}$
3: **for** $u \in U$ **do**
4: $min_d \leftarrow +\infty$
5: $c_{min} \leftarrow \varnothing$
6: **for** $c \in C_{valid}$ **do**
7: $d \leftarrow -\infty$
8: **for** $v \in c$ **do**
9: $d \leftarrow \max\{d, d^a(u, v)\}$
10: **end for**
11: $k^c \leftarrow \max_{v \in c \cup u} k_v$
12: **if** $d < min_d \wedge |c| + 1 \geq \max\{k^c, k_u\} \wedge d \leq \tau_d$ **then**
13: $min_d \leftarrow d$
14: $c_{min} \leftarrow c$
15: **end if**
16: **end for**
17: **if** $c_{min} \neq \varnothing$ **then**
18: $add(c_{min}, u)$
19: **end if**
20: **end for**

7. The detailed analysis of the complexity can be found in the supplementary material section.

anonymization distance between u_2 and cluster $\{u_3\}$ (i.e., $d^a(u_2, u_3) = 0.216$) is less than τ_d (i.e., 0.575) and the resulting cluster is still valid. However, it cannot find any cluster for u_4 as adding it to any cluster will make the cluster invalid. MS returns clusters: $\{u_0, u_1\}$ and $\{u_2, u_3\}$.

Function 2 *split_big_cluster* ($\mathcal{D}^a$, c, $\mathcal{K}$)

1: $n_c \leftarrow \lfloor |c|/k^c \rfloor$
2: Let $\mathcal{D}^a_c$ be distance matrix of c's users.
3: $M_c \leftarrow$ *BanditPAM*$(\mathcal{D}^a_c, n_c)$
4: $c_r \leftarrow c \setminus M_c$
5: $C_c \leftarrow \varnothing$
6: **for** $i \in \{0 \ldots n_c - 1\}$ **do**
7: **if** $i < n_c - 1$ **then**
8: $c_c \leftarrow$ *find_nearest_elements*$(\mathcal{D}^a_c, c_r, M_c[i], k^c - 1)$
9: **else**
10: $c_c \leftarrow c_r$
11: **end if**
12: $remove\_set(c_r, c_c)$
13: $add\_set(C_c, c_c \cup \{M_c[i]\})$
14: **end for**
15: **return** C_c

Function *split_big_clusters()*. This function receives the anonymization distance matrix: $\mathcal{D}^a$, a cluster: c, and users' k values: $\mathcal{K}$. First, it calculates the number of clusters to be generated: n_c (line 1) and finds the distance matrix of users in c: $\mathcal{D}^a_c$ (line 2). Then, it calls Algorithm *BanditPAM* [16], an efficient variant of k-Medoids, to find the set of users who are the center of clusters to be generated: M_c (line 3). Next, it finds the set of users who are not in M_c: c_r (line 4) and initializes the set of resulting clusters C_c as empty set (line 5). It starts creating n_c clusters of users in c and adding the clusters to C_c (line 6-14). For every i between 0 to $n_c - 1$, if i is less than $n_c - 1$, it calls Function *find_nearest_elements()* to find $k^c - 1$ users in c_r who are the closest ones to the i-th medoid user ($M_c[i]$): c_c (line 8).[8] In case i is greater than or equal to $n_c - 1$, it assigns c_c of c_r (line 10). Then, it removes users in c_c from c_r (line 12) and adds the cluster containing the medoid user and c_c's users to C_c (line 13). Finally, the function returns C_c (line 15).

The complexity of MS is $\mathcal{O}(m \times n^2 \log n)$, where m is the number of clusters in $\mathcal{C}$, and n is the number of users in V^U. We include the detailed time complexity analysis in the supplementary material section.

5 Privacy Analysis

In this section, we analyze how the p-k-ad principle and the PCKGA algorithm protect user privacy. The following theorem states that, if an anonymized KG $\overline{G}$ satisfies p-k-ad, any user u in $\overline{G}$ cannot be re-identified with a confidence higher than $\frac{1}{k_u}$, where k_u is specified by u.

Theorem 1. Let $\overline{G}$ be an anonymized KG. If $\overline{G}$ satisfies p-k-ad, for every user $u \in \overline{V}^U$, an adversary cannot exploit $\mathcal{AK}(u)$ to re-identify u with a confidence higher than $\frac{1}{k_u}$.

The privacy protection of PCKGA relies on MS, which generates valid clusters, and KGG [2], that generalizes users' attributes and relationships. As the following theorem proves, MS always generates a set of valid clusters.

8. Interested readers can check Function *find_nearest_elements()*'s implementation in popular source code sharing platforms (e.g., GeeksForGeeks).

TABLE 1
Properties of datasets used for experiments.

Dataset	$\lvert\mathbf{V^U}\rvert$	$\lvert\mathbf{V^A}\rvert$	$\lvert\mathbf{R^{UA}}\rvert$	$\lvert\mathbf{R^{UU}}\rvert$	$\lvert\mathbf{E^{UA}}\rvert$	$\lvert\mathbf{E^{UU}}\rvert$
Freebase [17]	5,000	4,016	10	3	41,067	2,713
Yago [18]	8,917	4,386	21	4	94,472	1,894
Coil [19]	5,822	41	86	0	500,692	0
Credit [20]	1,000	1,046	21	0	21,000	0

Theorem 2. Let G be a KG and $\overline{\mathcal{C}}$ be a set of clusters generated by Algorithm 3 when its input is G. Every c in $\overline{\mathcal{C}}$ is valid.

By applying KGG [2] on the set of valid clusters generated from MS, PCKGA always generates anonymized KGs satisfying p-k-ad.

Theorem 3. Let G be a KG, $\overline{\mathcal{C}}$ be the set of clusters generated by Algorithm 3 over G, and $\overline{G}$ be the anonymized version of G created by KGG algorithm, executed with G and $\overline{\mathcal{C}}$. $\overline{G}$ satisfies p-k-ad.

The supplementary material includes theorems' proofs.

6 Evaluation

We conduct experiments on four real-life datasets to evaluate the proposed algorithms (i.e., VAC and MS). The first experiment aims to show the effectiveness of VAC over state-of-the-art clustering algorithms (i.e., k-Medoids [13] and HDBSCAN [14]). The second experiment is designed to evaluate the impact of MS in improving the quality of anonymized KGs. The third experiment shows the performance of VAC and MS. In the final experiment, we conduct a comparison between the anonymized KGs produced by our algorithms and those generated by k-ad's algorithm [2]. Since KGs can illustrate relational data, we also compare our algorithm against a cluster-based anonymization algorithm designed for this type of data [15].

6.1 Datasets

We use four popular real-life datasets, namely *Freebase* [17], *Yago* [18], *Credit* [20], and *Coil* [19]. *Freebase* and *Yago* are selected since they are the most widely used in state-of-the-art KGs' deep learning publications. *Freebase* and *Yago* store attributes' values (e.g., $nationality$, $location$) and relationships (e.g., $spouse$, $parent$) of famous people (e.g., the film director Anthony Asquith) derived from Wikipedia, WordNet, and other data sources. Even if they present semantically similar attributes/relationships, they have different types of attributes/relationships and different sizes. *Credit* stores properties of real bad credits in Germany and *Coil* contains information on customers of an insurance company. For each dataset, we manually categorize its nodes into the set of users (i.e., V^U) and set of attributes' values (i.e., V^A). Since *Freebase* has $5{,}000$ users and $4{,}016$ values, it has $9{,}016$ nodes. *Yago* has 13,303 nodes including $8{,}917$ users and $4{,}386$ values. *Coil* has $5{,}863$ nodes containing $5{,}822$ users and 41 values while that of *Credit* is $2{,}046$ including $1{,}000$ users and $1{,}046$ values.

These datasets represent a valid benchmark to test our algorithms as they contain different numbers of users, attributes and relationships (see Table 1 for datasets' properties). However, they do not contain users' anonymity levels, which we generated synthetically following two main

strategies. As first strategy, we adopt the approach used to generate anonymity levels to evaluate state-of-the-art personalized k-anonymity methods for location data [21], [22], [23]. Here, anonymity levels are generated using Zipf distribution, with α parameter set to 2. As the second strategy, we consider that individuals do not set their anonymity levels randomly. To model users' behavior, we assume that users who share more information are more willing to protect their data and thus require a higher anonymity level. This assumption comes from a recent survey run by CISCO[9] that pointed out that users who actively set their privacy settings are those that exploit more online services and thus those that expose more information. According to this second strategy, the anonymity level of a user u is determined by the number of u's edges. The greater the number of edges is, the greater the associated k value is.

In both the strategies, we assume that anonymity levels are taken from fixed intervals. State-of-the-art personalized anonymization techniques for location data specified these levels to be from 2 to 5 [21], from 3 to 15 [22], and from 10 to 50 [23]. In this paper, we therefore use two intervals containing these levels. The first contains levels between 2 and 5 with an increment by 1 (denoted hereafter as $2, 5, 1$); the second contains levels between 5 and 50 with an increment by 5 (denoted as $5, 50, 5$ in what follows). Thus, the maximum anonymity levels of users are 5 and 50, respectively for the two considered intervals.

We generate users' anonymity levels for every dataset following the above-described strategies, $zipf$ and te, and the two intervals $2, 5, 1$ and $5, 50, 5$. The obtained datasets are referred hereafter as: $zipf\#2, 5, 1$, $zipf\#5, 50, 5$, $te\#2, 5, 1$, and $te\#5, 50, 5$. Since the second interval has higher k values, $zipf\#5, 50, 5$ and $te\#5, 50, 5$ settings give stronger privacy protection to users than the others.

6.2 Clustering settings

The first experiment aims to show the effectiveness of VAC over two state-of-the-art clustering algorithms: k-Medoids [13] and HDBSCAN [14]. However, these two algorithms do not support personalized k values. k-Medoids receives as input only the number of clusters to be generated, say κ, whereas HDBSCAN receives the minimum size of clusters to be generated, say k_{unique}. This last parameter represents the k-anonymity level that has to be applied to the whole dataset, i.e., all the clusters. In order to exploit them in PCKGA, as an alternative to VAC, we have to specify κ in k-Medoids and k_{unique} for HDBSCAN, included in the parameters' values $\mathcal{P}$ (line 2, Algorithm 1).

In particular, to define k_{unique}, we adopt three strategies, according to which k_{unique} is set as the max, min, average of all anonymity levels of users in the considered datasets. As such, we run HDBSCAN (k-Medoids, resp.) with k fixed as max, denoted as $hdbscan\#max$ ($km\#max$, resp.); min, e.g., $hdbscan\#min$ ($km\#min$, resp), and average, e.g., $hdbscan\#mean$ ($km\#mean$, resp.). κ is defined as the number of users in the dataset divided by the minimum number of users in each cluster, that is, the adopted k_{unique}. Moreover, since HDBSCAN ensures the minimum size of its generated clusters, we develop a basic HDBSCAN's extension, namely HDB*, to generate clusters with multiple k_{unique} and compare it with VAC. In particular, we first gather users into clusters such that users in the same cluster have the same k values. Then, we execute HDBSCAN on each of these clusters with k_{unique} equal to its users' k value. Therefore, each user is only anonymized with users sharing the same k value.

Due to the lack of space, this section only presents the results for k-Medoids, HDB*, and VAC, whereas the Supplementary Material reports HDBSCAN's settings' results.

6.3 Metrics

To evaluate the quality of generated KGs, we use two metrics: the average information loss (AIL), and the ratio of removed users (RRU). AIL of a user is estimated as the differences between his/her attributes' values and out-/in-degrees in anonymized KG $\overline{G}$ with his/her original ones. If the user is removed, his/her information loss is 1.

$$AIL(G, \overline{G}) = \frac{1}{|V^U|} \sum_{u \in V^U} \frac{AL'(G, \overline{G}, u) + DL'(G, \overline{G}, u)}{2}$$

where AL' and DL' are u's information loss on his/her attributes and relationships. We define AL' as follows:

$$AL'(G, \overline{G}, u) = \frac{1}{|R^{UA}|} \sum_{r_a \in R^{UA}} \frac{|\mathcal{I}_a(\overline{G}, r_a, u) \cap \mathcal{I}_a(G, r_a, u)|}{|dom(r_a) \cap \mathcal{I}_a(G, r_a, u)|}$$

where $\mathcal{I}_a(G, r_a, u) \subseteq \mathcal{I}_a(G, u)$ ($\mathcal{I}_a(\overline{G}, r_a, u) \subseteq \mathcal{I}_a(\overline{G}, u)$, resp.) represents values of u's attribute r_a in original KG G (anonymized KG $\overline{G}$, resp.); and $dom(r_a) = \{v_a | (u, r_a, v_a) \in E^{UA}\}$. DL' is defined as follows:

$$DL'(G, \overline{G}, u) = \frac{1}{|R^{UU}|} \sum_{r_r \in R^{UU}} \frac{|d_o^{r_r}(G, \overline{G}, u)| + |d_i^{r_r}(G, \overline{G}, u)|}{2 \times |V^U|}$$

$$d_o^{r_r}(G, \overline{G}, u, r_r) = d_o(\overline{G}, r_r, u) - d_o(G, r_r, u)$$

$$d_i^{r_r}(G, \overline{G}, u, r_r) = d_i(\overline{G}, r_r, u) - d_i(G, r_r, u)$$

where $d_o(G, r_r, u)$ and $d_i(G, r_r, u)$ ($d_o(\overline{G}, r_r, u)$ and $d_i(\overline{G}, r_r, u)$, resp.) are the out- and in-degree of relationship r_r of u in original KG G (anonymized KG $\overline{G}$, resp.).

Since removed users have a high impact on AIL, we use RRU to analyze these impacts in detail. RRU measures the percentage of users in the original KG G that are not included in its anonymized version $\overline{G}$.

In case the dataset exploits user anonymity levels generated by using $zipf$, we generate the levels three times and run our experiments separately. The metrics' results are given as average of the three executions.

6.4 VAC Algorithm

This experiment compares the quality of anonymized KGs generated by running PCKGA with VAC, k-Medoids [13], and our extension of HDBSCAN (HDB*). To evaluate the impact of the clustering algorithms, we only execute VAC (k-Medoids, HDB*, resp.) without the Merge-Split Algorithm. Among the obtained clusters, we only keep those that are valid to generate anonymized KGs. Table 2 illustrates the average information loss of users and the ratio of removed

9. https://www.cisco.com/c/dam/global/en_uk/products/collateral/security/cybersecurity-series-2019-cps.pdf

TABLE 2
Anonymized KGs' quality generated with: k-Medoids, HDB*, and VAC.

Data	Settings	k-Medoids min	k-Medoids mean	k-Medoids max	HDB*	VAC
Freebase	$te\#2,5,1$	0.404	0.407	0.134	0.221	**0.0013**
	$te\#5,50,5$	0.566	0.349	0.193	0.290	**0.0137**
	$zipf\#2,5,1,2$	0.479	0.475	0.175	0.221	**0.0018**
	$zipf\#5,50,5,2$	0.689	0.698	0.267	0.222	**0.0168**
Yago	$te\#2,5,1$	0.474	0.476	0.188	0.260	**0.0005**
	$te\#5,50,5$	0.242	0.249	0.117	0.336	**0.0022**
	$zipf\#2,5,1$	0.496	0.496	0.215	0.189	**0.0008**
	$zipf\#5,50,5$	0.392	0.397	0.158	0.212	**0.0070**

(a) Average Information Loss

Data	Settings	k-Medoids min	k-Medoids mean	k-Medoids max	HDB*	VAC
Freebase	$te\#2,5,1$	0.400	0.403	0.119	**0.000**	0.0006
	$te\#5,50,5$	0.551	0.311	0.097	0.008	0.0040
	$zipf\#2,5,1$	0.475	0.478	0.159	**0.000**	0.0005
	$zipf\#5,50,5$	0.667	0.661	0.193	0.008	0.0040
Yago	$te\#2,5,1$	0.466	0.467	0.160	**0.000**	0.0001
	$te\#5,50,5$	0.215	0.217	0.024	**0.000**	0.0008
	$zipf\#2,5,1$	0.487	0.487	0.187	**0.000**	0.0001
	$zipf\#5,50,5$	0.362	0.369	0.067	**0.000**	0.0025

(b) Ratio of Removed Users

users in anonymized KGs, for both the considered datasets. VAC helps PCKGA to always generate the highest quality KGs compared to k-Medoids and HDB*, even though VAC does not require any additional input parameter. With the setting $te\#5,50,5$ for *Yago*, k-Medoids generates the highest quality KGs whose information loss is 0.117 while the information loss of those generated by VAC is 0.0022. The main reason is that VAC removes outliers. In this setting, VAC only removes a few number of users: 0.08%, while the lowest ratios of removed users among all executions of k-Medoids, and HDB* are 0% and 2.4%, respectively.

Table 2 also shows how the adopted set of anonymity levels impact. In general, the higher values of k (i.e., stronger privacy protection) result in lower anonymized KGs' quality. The information loss of KGs generated for settings $te\#5,50,5$ and $zipf\#5,50,5$, whose maximum k value is 50, is higher than that of those generated for settings $te\#2,5,1$ and $zip\#2,5,1$, whose maximum k value is 5, in most clustering settings (i.e., VAC, $km\#min$, $hdb*$). It is not the case for the other clustering settings (i.e., $km\#min$, $km\#mean$), because k-Medoids generates many invalid clusters whose users are removed from anonymized KGs. With clustering setting $km\#mean$, the ratio of removed users in *Yago* is 46.7% (21.7%, resp.) for setting $te\#2,5,1$ ($te\#5,50,5$, resp.). These huge amount of removed users make anonymized KGs to loose much information. Moreover, it is relevant to note that anonymized KGs generated using levels generated according to te have higher quality than those generated with $zipf$. We recall that according to te, users who have a similar number of edges have similar anonymity levels. Then, since users in the same cluster might have a similar number of edges, they also have similar anonymity levels. This prevents users with small anonymity levels to be anonymized with high anonymity levels. As a result, they do not lose too much information.

As VAC considers different k values, it generates high-quality anonymized KGs in both the real-life scenario (i.e., te) and the random simulation scenario (i.e., $zipf$) for the highest range of k values (i.e., $5,50,5$). The advantages of VAC over HDB* indicate that simple customization of current clustering algorithms does not fit the scenario of personalized anonymization.

6.5 MS Algorithm

This experiment aims to evaluate the effectiveness of MS. We consider the $zipf\#5,50,5$ setting, as using this setting, k-Medoids, HDB*, and VAC generate the high-information-loss KGs. We recall that MS removes invalid clusters and adds their users to valid ones. This is done only if the anonymization distances between these users and valid clusters are less than or equal to the maximum distance measured by τ. We also run MS by varying τ, to measure the impact of τ on the quality of generated anonymized KGs. Table 3 illustrates MS' effectiveness on *Freebase* and *Yago*.

By increasing τ, MS decreases the number of removed users, which leads to the decrements of users' information loss, since AIL of a removed user is 1. MS decreases the information loss of users in KGs generated by k-Medoids executed with all the settings. As an example, in case of $km\#min$ in dataset *Yago*, increasing τ from 0 to 1 decreases the information loss of the generated KGs from 0.368 to 0.016, whereas the ratio of removed users is decreased from 36.2% to 0%. MS improves the quality KGs generated by executing with HDB* to be closer to that of those generated by VAC. On *Freebase*, increasing τ from 0 to 1 decreases the information loss of HDB* (VAC, resp.) from 0.017 to 0.011 (from 0.0168 to 0.0087, resp.). Consequently, the information loss of users in original KGs is decreasing when increasing τ. Even though VAC does not generate big clusters, whose cardinality is higher than or equal to twice their anonymity, MS can still add some removed users to valid clusters and increase the quality of the generated clusters. As a result, the information loss of KGs using VAC is also decreased from 0.0070 to 0.0048 when increasing τ from 0 to 1 on *Yago*. Moreover, the maximum ratio of removed users of clusters generated by MS is at most those generated by removing invalid clusters. Therefore, MS is effective enough to improve the quality of anonymized KGs if data providers decide to choose clustering algorithms which do not support personalized anonymization (e.g., k-Medoids, HDB*).

6.6 Overall performance

We measure the performance of VAC by monitoring the execution time of VAC when running it with varying k values' generation strategies. MS' performance is computed by tracking the execution time of MS when running it with clusters generated by VAC with the strategy $zipf\#5,50,5$. Both algorithms have been implemented in Python 3 and run on a Debian 4 server, with 128 GB of RAM and its CPU is Intel Xeon with 128 cores. Table 4 illustrates the performance on the two datasets. The performance of VAC depends on the size of the datasets and the users' k values. The execution times of VAC running on *Yago* with $zipf\#5,50,5$ and $te\#5,50,5$ settings (523.93 and 469.86 seconds, respectively) are higher than those of VAC running on *Freebase* (150.14 and 183.21 seconds, respectively). The execution

TABLE 3
Anonymized KGs' quality generated by MS executed with clusters generated by k-Medoids, HDB*, and VAC.

Data	τ	k-Medoids min	k-Medoids mean	k-Medoids max	HDB*	VAC
Freebase	0.00	0.680	0.691	0.210	0.017	**0.0168**
	0.25	0.019	0.033	0.030	0.011	**0.0087**
	0.50	0.019	0.029	0.028	0.011	**0.0087**
	0.75	0.019	0.030	0.028	0.011	**0.0087**
	1.00	0.019	0.030	0.027	0.011	**0.0087**
Yago	0.00	0.368	0.375	0.079	0.004	**0.0070**
	0.25	0.016	0.016	0.015	0.004	**0.0051**
	0.50	0.016	0.016	0.015	0.004	**0.0049**
	0.75	0.016	0.016	0.015	0.004	**0.0048**
	1.00	0.016	0.016	0.015	0.004	**0.0048**

(a) Average Information Loss

Data	τ	k-Medoids min	k-Medoids mean	k-Medoids max	HDB*	VAC
Freebase	0.00	0.667	0.661	0.193	0.008	**0.0040**
	0.25	0.003	0.003	0.003	0.000	**0.0000**
	0.50	**0.000**	**0.000**	**0.000**	**0.000**	**0.0000**
	0.75	**0.000**	**0.000**	**0.000**	**0.000**	**0.0000**
	1.00	**0.000**	**0.000**	**0.000**	**0.000**	**0.0000**
Yago	0.00	0.362	0.369	0.067	**0.000**	**0.0025**
	0.25	0.001	0.001	0.001	**0.000**	0.0003
	0.50	**0.000**	**0.000**	**0.000**	**0.000**	0.0001
	0.75	**0.000**	**0.000**	**0.000**	**0.000**	**0.0000**
	1.00	**0.000**	**0.000**	**0.000**	**0.000**	**0.0000**

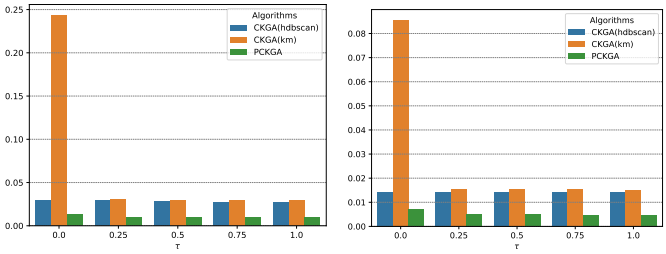
(b) Ratio of Removed Users

time of VAC running with $zipf\#5,50,5$ (150.14 seconds) is higher than that of VAC running with $zipf\#2,5,1$ (110.10 seconds). However, the number of users in the datasets has a higher impact on VAC's performance than the users' k values. Even though VAC has a higher execution time than k-Medoids, HDBSCAN, HDB* (i.e., on *Yago*, with $zipf\#5,50,5$, they have 32, 36, and 54 seconds, resp., where VAC is about 524 seconds), the information loss with k-Medoids and HDBSCAN (0.267, 0.334, and 0.222, resp., with $zipf\#5,50,5$ setting) is much higher than the information loss of VAC (0.0168). On the other hand, MS performance mostly relies on datasets' number of users instead of its parameter τ. On *Freebase* dataset, increasing τ from 0 to 0.25 decreases the execution time from 32.19 to 31.75 seconds. However, when it reaches 0.5, the execution time increases to 32.81 seconds. The standard deviations of execution times on varying values of τ are 0.71 seconds on *Freebase*, and 1.92 seconds on *Yago* that are small comparing to their execution times. Nevertheless, changing from *Freebase* to *Yago* increases the execution time. The average execution time on *Freebase* (i.e., 32.01 seconds) is smaller than the one on *Yago* (i.e., 99.52 seconds).

The complexity of VAC and MS are $\mathcal{O}(n^2 \log n)$ and $\mathcal{O}(m \times n^2 \log n)$, respectively (see Appendix 2). In practice, data providers do not need to anonymize their KGs in real-time. Instead, they will anonymize their KGs once and publish the anonymized versions. Therefore, MS and VAC are feasible with KGs' real-life applications.

6.7 Comparison with the Simple Knowledge Graph Personalized Anonymization

This experiment aims to compare the quality of anonymized KGs generated by PCKGA (i.e., using VAC/MS and KGG) and state-of-the-art anonymization algorithms in common usages. First, we compare PCKGA with the cluster-based anonymization algorithm, i.e., the Cluster-Based Knowledge Graph Anonymization Algorithm (CKGA) [2] in anonymizing KGs. Secondly, we compare it with Primule [15], the cluster-based anonymization algorithm for relational data which is used to anonymize users' profile. Unlike PCKGA, CKGA and Primule impose a unique anonymity level for all users. We use the most challenging setting (i.e., $zipf\#5,50,5$) to generate k values, and we choose the highest anonymity level present in the dataset as the k value for the execution of CKGA and Primule while PCKGA considers all k values.



(a) Freebase (b) Yago

Fig. 2. Average information loss in KGs from CKGA and PCKGA.

Since PCKGA and CKGA [2] supports similar parameters (i.e., clustering algorithm $\mathcal{A}$ and τ), we evaluate the impact of the parameters on the information loss of anonymized KGs generated from *Freebase* and *Yago*. Fig. 2 and Table 5 illustrate the quality of anonymized KGs generated by our algorithm and CKGA. Across all values of τ, PCKGA generates higher quality KGs. As τ increases from 0 to 1, the average information loss of anonymized KGs generated by PCKGA remains lower compared to CKGA. For example, Fig. 2a shows that the anonymized KGs of *Freebase* generated by CKGA exhibit a minimum and maximum average information loss of 0.027 and 0.244, respectively. In contrast, PCKGA achieves lower values: 0.009 and 0.013, respectively. Table 5 further demonstrates that the minimum and maximum ratios of removed users in *Freebase*'s KGs generated by CKGA are 0 and 0.223 respectively, while PCKGA achieves significantly lower values of 0 and 0.008. Similar trends are observed in the experimental results obtained from *Yago*. The reason is that CKGA does not consider the different users' anonymity levels and applies the maximum one to all users. Therefore, PCKGA can generate better quality anonymized KGs in the scenario of personalized anonymization.

Primule [15] is designed to anonymize relational data, we use two real-life relational datasets (Credit [20] and Coil [19]) to compare our work with Primule. Since Primule does

TABLE 4
Execution time of VAC and MS (seconds).

Settings	Freebase	Yago
zipf#2,5,1	110.10	363.41
zipf#5,50,5	150.14	523.93
te#2,5,1	106.63	330.05
te#5,50,5	183.21	469.86

(a) VAC

τ	Freebase	Yago
0.00	32.19	97.24
0.25	31.75	99.73
0.50	32.81	98.28
0.75	30.89	99.37
1.00	32.42	102.46

(b) MS

TABLE 5
Ratio of Removed Users in KGs from PCKGA and CKGA executed with HDBSCAN(hdb.) and k-Medoids(km.).

Data	Algorithms	τ 0.00	0.25	0.50	0.75	1.00
Freebase	CKGA(hdb.)	**0.003**	0.003	0.001	**0.000**	**0.000**
	CKGA(km.)	0.223	0.003	**0.000**	**0.000**	**0.000**
	PCKGA	0.008	**0.000**	**0.000**	**0.000**	**0.000**
Yago	CKGA(hdb.)	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
	CKGA(km.)	0.074	0.001	**0.000**	**0.000**	**0.000**
	PCKGA	0.002	**0.000**	**0.000**	**0.000**	**0.000**

not remove any outliers from the datasets; to ensure that PCKGA also does not remove outliers, we executed it with $\tau = 1$. PCKGA's anonymized KGs show 40.41% and 31% lower information loss (0.137 on *Credit* and 0.067 on *Coil*) compared to Primule's (0.339 on *Credit* and 0.213 on *Coil*) in both datasets. This difference is attributed to the fact that Primule creates large clusters that exceed the required k value by at least two times and applies the maximum k value to all users, while PCKGA considers all values.

7 Conclusion

In this paper, we proposed the Personalized k-Attribute Degree principle to allow users to specify their own protection level (k) and PCKGA to generate anonymized KGs satisfying the proposed principle. We conducted experiments by using two real-life datasets to show that running PCKGA with our clustering algorithm (VAC) generates high-quality anonymized KGs compared to those generated by running with state-of-the-art clustering algorithms. PCKGA has good performance to be used in practice, and outperforms the previous anonymization algorithm for KGs [2] and relational data [15]. Our work can be extended in various directions. We plan to design a risk assessment module to recommend anonymity levels to users, so that it is easier for non-expert users to specify their anonymity levels. Additionally, we plan to investigate novel generalization techniques that utilize fuzzy logic representation to represent users' relationships. These techniques allow for the representation of user relationships within the same clusters using the probability of their presence, rather than the traditional method of adding and removing relationships. Another extension is to protect users' privacy from inference attacks when attackers exploit the public protection levels. The protection can rely on the a risk assessment algorithm that predicts the probability that the protection level of a user is associated with the sensitivity level of his/her sensitive values. If the risk is too high, users' data are removed from anonymized KGs.

References

[1] S. Ji *et al.*, "Graph data anonymization, de-anonymization attacks, and de-anonymizability quantification: A survey," *IEEE CST*, 2017.
[2] A.-T. Hoang *et al.*, "Cluster-based anonymization of knowledge graphs," in *ACNS*, 2020.
[3] X. Chen *et al.*, "Publishing community-preserving attributed social graphs with a differential privacy guarantee," *PETS*, 2020.
[4] X. You *et al.*, "Reschedule gradients: Temporal non-iid resilient federated learning," *IEEE IoT-J*, pp. 747–762, 2023.
[5] M. Thouvenot *et al.*, "Knowledge graph anonymization using semantic anatomization," in *IEEE Big Data*, 2020, pp. 4065–4074.
[6] K. Sheehan, "Toward a typology of internet users and online privacy concerns," *Inf. Soc.*, vol. 18, no. 1, pp. 21–32, 2002.
[7] A. F. Westin, "Privacy and freedom," *Washington and Lee Law Review*, vol. 25, no. 1, p. 166, 1968.
[8] X. Liu *et al.*, "Personalized extended (α, k)-anonymity model for privacy-preserving data publishing," *Comput. Pract. Exp.*, 2017.
[9] B. Niu *et al.*, "Adapdp: Adaptive personalized differential privacy," in *IEEE INFOCOM*, 2021, pp. 1–10.
[10] Z. Jorgensen *et al.*, "Conservative or liberal? personalized differential privacy," in *IEEE ICDE*, 2015, pp. 1023–1034.
[11] M. Yuan *et al.*, "Personalized privacy protection in social networks," *Proceedings of the VLDB Endowment*, pp. 141–150, 2010.
[12] J. Jiao *et al.*, "A personalized privacy preserving method for publishing social network data," in *TAMC*, 2014, pp. 141–157.
[13] A. Fahad *et al.*, "A survey of clustering algorithms for big data: Taxonomy and empirical analysis," *IEEE TETC*, pp. 267–279, 2014.
[14] R. J. Campello *et al.*, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM TKDD*, 2015.
[15] F. Pratesi *et al.*, "Primule: Privacy risk mitigation for user profiles," *Data & Knowledge Engineering*, p. 101786, 2020.
[16] M. Tiwari *et al.*, "Banditpam: Almost linear time k-medoids clustering via multi-armed bandits," in *NeurIPS*, 2020.
[17] K. Bollacker *et al.*, "Freebase: a collaboratively created graph database for structuring human knowledge," in *SIGMOD*, 2008.
[18] A. García-Durán *et al.*, "Learning sequence encoders for temporal knowledge graph completion," in *EMNLP*, 2018, pp. 4816–4821.
[19] P. Van Der Putten *et al.*, "Coil challenge 2000: The insurance company case," Leiden LIACS Technical Report, Tech. Rep., 2000.
[20] U. Groemping, "South german credit data: Correcting a widely used data set," *Rep. Math., Phys. Chem.*, 2019.
[21] Z. Luo *et al.*, "A personalized k-anonymity with fake position generation for location privacy protection," in *ICoC*, 2015.
[22] J. Wang *et al.*, "Achieving personalized k-anonymity-based content privacy for autonomous vehicles in cps," *IEEE TII*, 2020.
[23] S. Zhang *et al.*, "A caching and spatial k-anonymity driven privacy enhancement scheme in continuous location-based services," *Future Generation Computer Systems*, vol. 94, pp. 40–50, 2019.

Anh-Tu Hoang is a Postdoctoral Researcher at the University of Insubria, Italy, where he received his Ph.D. in Computer Science. His research is related to privacy, security, decentralized systems, and machine learning. He has worked on privacy-preserving techniques for knowledge graphs and blockchain platforms.

Barbara Carminati is a professor of Computer science at the University of Insubria. She received her Ph.D. in Computer Science from the University of Milano, Italy. Her primary areas of interest in research are security and privacy for cutting-edge applications including cloud computing, semantic web, data outsourcing, XML data sources, Web services, and data streams, and online social networks. She has written more than 130 publications with peer reviews on these subjects. She is an IEEE and ACM senior member.

Elena Ferrari is a professor of Computer science at the University of Insubria, Italy, where she leads the STRICT Socialab. Her research activities are mainly related to data cybersecurity, data privacy and trust. Ferrari has a Ph.D. in Computer Science from the University of Milano, Italy. For her research, she received several pretigious awards, including the 2024 IEEE Innovation in Societal Infrastructure Award, the 2009 IEEE Computer Society Technical Achievement Award, and the ACM SIGSAC Outstanding Contributions Award. She is an IEEE and ACM fellow.