**kt-Safety: Graph Release via $k$-Anonymity and $t$-Closeness**

Weilong Ren ©, Kambiz Ghazinour, and Xiang Lian ©

**Abstract**—In a wide spectrum of real-world applications, it is very important to analyze and mine graph data such as social networks, communication networks, citation networks, and so on. However, the release of such graph data often raises privacy issue, and the graph privacy preservation has recently drawn much attention from the database community. While prior works on graph privacy preservation mainly focused on protecting the privacy of either the graph structure only or vertex attributes only, in this paper, we propose a novel mechanism for graph privacy preservation by considering attacks from both graph structures and vertex attributes, which transforms the original graph to a so-called $kt$-safe graph, via $k$-anonymity and $t$-closeness. We prove that the generation of a $kt$-safe graph is NP-hard, therefore, we propose a feasible framework for effectively and efficiently anonymizing a graph with low anonymization cost. In particular, we design a cost-model-based graph partitioning approach to enable our proposed divide-and-conquer strategy for the graph anonymization, and propose effective optimization techniques such as pruning method and a tree synopsis to improve the anonymization efficiency over large-scale graphs. Extensive experiments have been conducted to verify the efficiency and effectiveness of our proposed $kt$-safe graph generation approach on both real and synthetic data sets.

**Index Terms**—$kt$-safety, $k$-anonymity, $t$-closeness

**1 INTRODUCTION**

The rapid development of social-network applications such as Twitter, Facebook, MySpace, and Friendster has brought a great opportunity for researchers and industries to better explore and understand social behaviors hidden in the world of human beings. To do so, some organizations or companies (e.g., Facebook) may need to release social-network data to the public, where the released data may contain users’ privacy information (e.g., salary and relationships with other users). To protect such privacy information from leakage, some critical identifiers (e.g., name and social security numbers) that can uniquely identify users are usually removed before publishing. However, simply removing these identifiers cannot prevent attackers from locating a user (node) in social networks, since a user can also be identified by one’s quasi-identifier (e.g., a combination of some non-sensitive attribute values) [24] and/or one’s neighborhood information in the graph [5], [38].

**Example 1 (Social-Network).** Fig. 1 is an example of a social-network graph before the anonymization and release. Each vertex in the graph corresponds to a user with two attributes, one non-sensitive attribute $A_1$ and one sensitive attribute $A_2$. Here, for sensitive attribute $A_2$ (e.g., salary), we assume that its value range is sensitive, if $A_2$ is less than 0.2 (i.e., low salary). Each edge represents the friendship between two users.

From Fig. 1, the release of such a graph may greatly threaten the privacy of users, for example, vertex $v_4$ with low salary. Thus, it is very important to study how to release the most graph information (i.e., attribute values and graph structure), without revealing the users’ privacy.

In order to publish such social-network data while protecting the users’ privacy, one possible direction is to separately release the anonymized graph structure and users’ attribute values. For example, [5], [38] considered the graph structure attack only, and protected the users’ identities via any structure attack. As an example in Fig. 1, by removing all attribute values at vertices, we can obtain a 2-automorphism anonymized graph [38], where each node cannot be distinguished from at least 1 ($= 2 - 1$) other symmetric vertex via structural information. A better graph releasing strategy is to release both the graph structure and attribute values together. Yuan et al. [33] proposed a graph anonymization mechanism, called $k$-degree-$l$-diversity, which guarantees that the candidate set of any vertex has at least $k$ candidate vertices (i.e., $k$-anonymity [24]) with the same degree and contains at least $l$ different values on sensitive attribute (i.e., $l$-diversity [21]).

**Example 2 (Privacy Disclosure).** In Fig. 1, vertices $v_1$ and $v_4$ satisfy the 2-degree-2-diversity requirement, where they both have 2 similar vertices (i.e., $v_1$ and $v_3$) and contain 2 different values (i.e., 0.1 and 0.7) on the sensitive attribute. However, this is actually not safe for vertices $v_1$ and $v_4$ in Fig. 1, if we consider their neighbors’ information. For instance, the neighbors of vertices $v_1$ and $v_4$ are $\{v_2, v_3, v_4\}$.
and \( \{v_1, v_2, v_6\} \), with non-sensitive attribute values \( \{0.4, 0.5, 0.5\} \) and \( \{0.5, 0.6, 0.4\} \), respectively. If an attacker knows neighbors of \( v_1 \) and \( v_4 \) (including their non-sensitive attributes), this attacker can uniquely identify \( v_1 \) and \( v_4 \), which reveals the privacy information of \( v_1 \) and \( v_4 \).

Therefore, most existing works (e.g., [5], [25], [33], [38]) usually protected the privacy of either the graph structure only or vertex attributes only. Their proposed techniques cannot be directly applied to the problem of privacy preservation of graph data, since graph data not only have sensitive attribute values, but also contain sensitive relationships (i.e., edges) among vertices in the graph structure. Meanwhile, although there are a few works (e.g., [23]) that consider both vertex attributes and the graph structure, they assumed that the attacker only knows direct neighbors of vertices (i.e., 1-hop neighbors), which cannot resist from attackers with \( n \)-hop (\( n > 1 \)) neighbor information.

In this paper, we will explore how to release both the graph structure and attributes of vertices, while preventing the users’ identities and sensitive attribute values from the leakage. To achieve this goal, there are two requirements on the released graph. First, to avoid the graph structure attack, we need to take into account the structure of \( n \)-hop neighbors for any vertex in the graph. Moreover, we need to consider the intersection attack, raised by the possible combinations of \( n \)-hop neighbor graph structure and vertex attributes (from either the target vertex or its neighbors). One straightforward solution for the intersection attack is to separately publish the anonymized graph structure and vertex attribute values, in which vertex IDs are not consistent in both anonymized parts. However, this method will directly destroy the relationship between the graph structure and vertex attributes. Instead, in this paper, we aim to publish both the graph structure and attribute values of each vertex, under the premise of protecting the graph privacy via either \( n \)-hop neighborhood attack or attribute-related attack.

Specifically, in this paper, we will adopt \( k \)-anonymity [24] and \( t \)-closeness [18] (a stronger privacy mechanism than \( l \)-diversity [21]) to guarantee the privacy preservation for vertex attributes in graph. Furthermore, for graph structure, we will protect the vertex identity from \( n \)-hop neighborhood attack by inserting fake nodes, edges, and/or attributes.

**Example 3 (An Anonymized Graph).** Fig. 2 illustrates an anonymized graph of the social-network graph in Fig. 1 by adding two fake vertices, \( v_7 \) and \( v_8 \), with non-sensitive attribute values 0.6 and 0.5, as the new neighbors of \( v_1 \) and \( v_4 \), respectively. We can see that \( v_1 \) and \( v_4 \) cannot be identified by an attacker through their neighbors’ information, since either \( v_1 \) or \( v_4 \) has 4 neighbors with the same set of non-sensitive attributes.

To tackle the graph privacy preservation problem, in this paper, we will formalize a novel graph privacy preservation mechanism, namely \( kt \)-safe graph, which can efficiently anonymize the graph by letting their \( n \)-hop neighbors contain the same or similar information (e.g., same/similar non-sensitive values). We prove that the \( kt \)-safe graph generation is NP-hard and not tractable. Therefore, we propose a feasible framework for the \( kt \)-safe graph anonymization, and optimize the framework by designing a cost-model-based graph partitioning strategy, proposing an effective pruning method, and devising a tree synopsis for facilitating efficient anonymization process (especially for a large-scale graph).

Note that, unlike previous works that release graph statistics only [7], [9] or synthetically generated (anonymized) graphs [12], in this paper, we aim to publish a “real” anonymized graph. Moreover, even if we use the widely adopted differential privacy mechanism [9], [10] to release a real anonymized graph (rather than graph statistics [9]), it may introduce too many nodes/edges in the real attributed graph due to its strong privacy guarantee, which leads to poor utility of the anonymized graph. We would like to leave interesting topics of releasing real attributed anonymized graphs via the differential privacy as our future work.

**Contributions.** In this paper, we make the following contributions:

1. We formalize a novel and important problem, namely \( kt \)-safe graph anonymization, based on our newly proposed graph privacy-preservation mechanism, \( kt \)-safety, in Section 2.
2. We prove the NP-hardness of the \( kt \)-safe graph anonymization problem, and propose a feasible \( kt \)-safe graph generation framework in Section 3.
3. We propose the detailed algorithms, under our \( kt \)-safe graph generation framework in Section 4.
4. We design a cost-model-based graph partitioning strategy, a pruning method, and a tree synopsis to optimize the \( kt \)-safe graph anonymization process in Section 5.
5. We evaluate through extensive experiments the performance of our \( kt \)-safe graph generation approach on real/synthetic data in Section 6.

In addition, Section 7 reviews related works on privacy preservation over tabular and graph data. Section 8 concludes this paper.

## 2 PROBLEM DEFINITION

### 2.1 Models

In this subsection, we present the graph model considered in this paper, and the model of the attacker’s knowledge used for identifying some target vertices.
Graph Model. First, we give the definition of graph data (e.g., social networks) as follows.

Definition 1 (Graph, $G$). A graph $G$ is represented by a triple $(V(G), E(G), A(G))$. Here, $V(G)$ is a set of vertices (nodes) $v_i$ (for $1 \leq i \leq |V(G)|$), $E(G)$ is a set of unlabelled edges $e_j$ (for $1 \leq j \leq |E(G)|$), and $A(G) = (A_1, \ldots, A_d, A_d)$ is an attribute set at each vertex $v_i \in V(G)$, where the first $(d-1)$ attributes $A_i$ (for $1 \leq j \leq d-1$) are non-sensitive attributes and the $d$th attribute $A_d$ is a sensitive attribute.

In Definition 1, we use $v_i, A_j$ to denote the value of attribute $A_j$ at vertex $v_i$. When an attribute value $v_i, A_j$ is not available (or missing), we denote it as $v_i, A_j = "-"$. The first $(d-1)$ attributes, $(v_i, A_1, \ldots, A_{d-1})$, of each vertex $v_i$ can be used as a quasi-identifier, denoted as $v_i, QI$, which partially discloses the identity of vertex $v_i$. Moreover, the $d$th attribute, $v_i, A_d$, contains a sensitive value that the user (vertex) $v_i$ is not willing to disclose (e.g., user’s age, salary, home address, or mobile number).

For simplicity, in this paper, we assume that there is only one sensitive attribute for each vertex. Nonetheless, our graph model can be easily extended to multiple sensitive attributes to be anonymized, by treating those sensitive attributes as a composite attribute.

Attacker Model. An attacker who wants to uniquely identify a vertex in the graph (e.g., some user in a social-network graph) may have a priori knowledge about features of this vertex, which is defined as follows.

Definition 2 (Attacker Model). We assume that an attacker knows the quasi-identifier, $v_i, QI$, of a target vertex $v_i$ and its $n$-hop neighbor information, denoted as $HN(v_i, n)$, where $HN(v_i, n)$ is a subgraph containing all neighbor vertices (with their non-sensitive attributes) that are less than or equal to $n$ hops away from $v_i$.

In Definition 2, an attacker can have two types of background information about a target vertex $v_i$, which are the target’s quasi-identifier $v_i, QI$ and its $n$-hop neighbor information $HN(v_i, n)$ (i.e., a subgraph structure with $n$-hop neighbors of $v_i$, containing non-sensitive attributes). In a special case where $n = 0$, the attacker only knows the quasi-identifier of the target vertex $v_i$.

2.2 $kt$-Safe Graph via Privacy Mechanisms

Preliminary. Prior privacy mechanisms such as $k$-anonymity [24] and $t$-closeness [18] are usually proposed for tabular data, which produce an anonymized version of the original tabular data. As a result, even if an attacker has the quasi-identifier of a target record, he/she can only obtain a set of candidate records, but still cannot uniquely identify the target from these records. Specifically, $k$-anonymity requires the size of this candidate set not be smaller than $k$, whereas $t$-closeness requires the distribution of sensitive attribute values within this set be similar to that of the entire released data (i.e., distances between the two distributions should not be larger than a threshold $t$).

Different from tabular data, the privacy protection over graph data is much more complex, since it involves the relationships (i.e., edges in the graph structure) among vertices in the graph (e.g., users in social networks).

The Protection Set of a Vertex. Specifically, for each vertex $v_i$ (associated with a quasi-identifier $v_i, QI$ and its $n$-hop neighbor information $HN(v_i, n)$) in the privacy-protected graph, we aim to guarantee that there exists a set, $v_i, PS$, of (at least $k$) candidate vertices $v_m$ with the same quasi-identifier $v_m, QI$ as $v_i, QI$ and $n$-hop neighbor information $HN(v_m, n)$ similar to $HN(v_i, n)$.

Below, we define the protection set, $v_i, PS$, of vertex $v_i$.

Definition 3 (Protection Set, $v_i, PS$). Given a parameter $t$, a graph edit distance threshold $c$, and a vertex $v_i$ with its quasi-identifier $v_i, QI$ and $n$-hop neighbor information $HN(v_i, n)$, a protection set, $v_i, PS$, of vertex $v_i$ is a set of vertices $v_m$ that satisfy the following three conditions:

1. $v_i, QI = v_m, QI$;
2. $ged(HN(v_i, n), HN(v_m, n)) \leq c$; and
3. $att\_dist(HN(v_i, L), HN(v_m, L), A_j) \leq t$ (for $1 \leq j \leq d - 1$ and $1 \leq L \leq n$);

where $ged(g_1, g_2)$ is the graph edit distance between subgraphs $g_1$ and $g_2$ by considering vertex/edge insertion operators, $HN(v_i, L)$ is a subgraph that contains all neighbor vertices that are less than or equal to $L$ hops away from $v_i$, and $att\_dist(g_1, g_2, A_j)$ is a distance function between attribute value distributions from subgraphs $g_1$ and $g_2$, for each non-sensitive attribute $A_j$ ($1 \leq j \leq d - 1$).

Intuitively, in Definition 3, the protection set $v_i, PS$ contains a set of vertices $v_m$ with (1) the same quasi-identifiers as $v_i, QI$, (2) graph structures of $n$-hop neighbors similar to that of $HN(v_i, n)$ (restricted via parameters $n$ and $c$), and (3) non-sensitive attributes $A_j$ in subgraphs $HN(v_m, L)$ similar to that of $HN(v_i, v_i)$ (restricted via parameter $t$). Note that, we do not want to identify vertex $v_i$ from the distributions of non-sensitive attributes of its $L$-hop neighbors $HN(v_j, L)$. Thus, in Definition 3, we apply $t$-closeness [18] to the distributions of non-sensitive attributes $A_j$, which can indirectly protect sensitive attribute values of graph nodes.

To preserve the privacy of graph data, in this paper, we consider graph edit operators such as node and edge insertions. Therefore, the graph edit distance, $ged(HN(v_i, n), HN(v_m, n))$, is given by considering vertex or edge insertion operators (i.e., changes of the graph structure).

To insert new attribute values, we will introduce new fake nodes with attributes, such that $att\_dist(HN(v_i, L), HN(v_m, L), A_j) \leq t$ holds for each attribute $A_j$ (for $1 \leq j \leq d - 1$) and with $(L, L)$-hop neighbors (for $1 \leq L \leq n$).

Following [18], we adopt Earth Mover’s Distance (EMD) as our distance function $att\_dist(g_1, g_2, A_j)$. To compute the Earth Mover’s Distance for category data [18], function $att\_dist(HN(v_i, L), HN(v_m, L), A_j)$ is given by half of the $L_1$-norm distance between two normalized (attribute frequency) distributions from subgraphs $HN(v_i, L)$ and $HN(v_m, L)$.

$$att\_dist(HN(v_i, L), HN(v_m, L), A_j) = \frac{1}{2} \sum_{a_j \in dom(A_j)} |pdf(a_j \in HN(v_i, L)) - pdf(a_j \in HN(v_m, L))|,$$

where $dom(A_j)$ is the value domain of attribute $A_j$, function $pdf(a_j \in HN(v, L)) = \frac{freq(a_j \in HN(v, L))}{freq(a_j \in HN(v, L))}$ is the frequency of attribute value $a_j$ in the graph $HN(v, L)$.
Given a graph $G$ from Eq. (1) and its anonymized graph $G'$, the anonymization cost, $Cost(G, G')$, from $G$ to $G'$ is given by

$$Cost(G, G') = |(V(G') \cup V(G)) - (V(G') \cap V(G))| + |(E(G') \cup E(G)) - (E(G') \cap E(G))|,$$  

where $V(G)$ and $E(G)$ are the sets of nodes and edges in graph $G$, respectively, and $|S|$ is the size of set $S$. $lacksquare$

In Definition 6, the anonymization cost, $Cost(G, G')$, is defined as the total number of graph edit operators from $G$ to $G'$ (as given by Eq. (2)).

In particular, the first term in Eq. (2) (i.e., $|(V(G') \cup V(G)) - (V(G') \cap V(G))|$) is the cost of vertex edits from $G$ to $G'$, while the second term in Eq. (2) (i.e., $|(E(G') \cup E(G)) - (E(G') \cap E(G))|$) corresponds to the cost of edge edits from $G$ to $G'$. The anonymization cost $Cost(G, G')$ sums up the two terms above.

Graph Release via $k$-Anonymity and $t$-Closeness. After defining the anonymization cost for graph privacy preservation, we provide the definition of our $kt$-safe graph anonymization problem below.

Definition 7 ($kt$-Safe Graph Anonymization Problem). Given a graph $G$, and thresholds $k$, $t$, $\alpha$, and $\epsilon$, the problem of $kt$-safe graph anonymization is to produce an anonymized $kt$-safe graph $G'$ from $G$, with minimal anonymization cost $Cost(G, G')$ (given in Eq. (2)). $lacksquare$

In Definition 7, given a graph $G$, the $kt$-safe graph anonymization problem aims to generate and release an anonymized $kt$-safe graph $G'$ from graph $G$ with the minimum anonymization cost in Definition 6.

Challenges. The major challenges to tackle the $kt$-safe graph anonymization problem are twofold. First, previous works assumed that an attacker knows the quasi-identifier and degree of a target only [33], or the graph structure of the target's $n$-hop neighbors only [5], [38]. However, this assumption may not always hold in practice, since the attacker may know both quasi-identifier and $n$-hop neighbors (for $n \geq 1$, as given in Definition 2). It is non-trivial how to protect the identity and sensitive attribute values of each vertex against attacker’s knowledge in Definition 2. Thus, we need to propose an effective solution to transforming each vertex in $G$ to a $kt$-safe vertex.

Second, it is very challenging to efficiently anonymizing vertices for the large-scale graph. The $kt$-safe graph anonymization problem is NP-hard (as will be proved in Section 3.1) and thus intractable. We need to design efficient heuristic approaches to improve the efficiency of the graph anonymization.

Table 1 depicts the commonly used symbols and their descriptions in this paper.
3 **kt-Safe Graph Generation**

3.1 **NP-Hardness of the kt-Safe Graph Anonymization Problem**

**Theorem 3.1 (NP-hardness).** The structural-attack-resistance-only version of kt-safe graph anonymization problem in Definition 7 is NP-hard.

**Proof.** Please refer to Appendix A.1 in our supplemental materials, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2022.3221333.

3.2 **kt-Safe Graph Generation Framework**

Since the kt-safe graph anonymization problem is NP-hard and not tractable, we will propose alternative heuristics-based algorithms. Specifically, Algorithm 1 illustrates a heuristic-based kt-safe graph generation framework, which produces a kt-safe graph \( G^* \) from a given graph \( G \), where we only consider graph adding operations (i.e., inserting vertices/edges) for simplicity. We leave as our future work for extending Algorithm 1 by embedding graph deletion operations (i.e., removing vertices/edges).

This framework consists of three phases: preprocessing, kt-safe graph generation, and kt-safe graph merging phases. In the preprocessing phase, we first partition graph \( G \) into \( s \) disjoint subgraphs, and store these partitioned subgraphs in a set \( G \) via function \( \text{partition}_\text{graph}(G, y, s) \) (line 1).

In the kt-safe graph generation phase, we make each partitioned subgraph \( g \in G \) a kt-safe subgraph. For each partitioned subgraph \( g \in G \), we sort vertices in a queue \( Q_g \) by the counts of their \((\leq n)\)-hop neighbors in ascending order (line 4), based on which we anonymize vertices. Intuitively, the vertices with less neighbors should be considered first, since graph edit operators (i.e., insertion operators over vertex/edge/attribute) over these vertices may have less effect on the neighbor information of other unprocessed vertices. Next, for each vertex \( v_i \in Q_g \) (assuming that \( v_i \) is the target vertex), we invoke function \( \text{initial}_\text{candidate}(v_i, g) \), and retrieve an initial candidate set \( v_i,CS \) for \( v_i \in g \) (lines 5-6), which can be used for quickly obtaining the protection set \( v_i,PS \) of \( v_i \).

Then, for each vertex \( v_i \in g \) in the queue \( Q_g \), we make \( v_i \) a kt-safe vertex via function \( \text{kt}_\text{safe}_\text{vertex}(v_i, v_i,CS, g) \) (line 7). Note that, in the anonymization process of \( v_i \), if graph edit operators (i.e., adding edges/attributes) over a vertex \( v_i \in g \) affects the privacy of previously anonymized vertices (i.e., identity or sensitive attribute values), then we will create a duplicate \( v'_i \) of \( v_i \) and perform graph edits on duplicate vertex \( v'_i \) (instead of \( v_i \)). This way, we can obtain a kt-safe graph \( g' \) from subgraph \( g \in G \), and add \( g' \) to the set \( G' \) (lines 8-9).

In the kt-safe graph merging phase, we merge vertices in kt-safe subgraphs \( g \in G' \) and their duplicates in other kt-safe subgraphs in \( G' \) via function \( \text{merge}_\text{subgraphs}(g') \) (final 10). Finally, we return the merged graph \( G' \) as the kt-safe graph (line 11).

3.3 **kt-Safe Graph Anonymization Algorithms**

Please refer to Appendix A.3 in our supplemental materials, available online, for a running example with the 3 phases of anonymizing the graph in Fig. 1 via our kt-safe graph generation approach.

4.1 **Preprocessing Phase**

**Graph Partitioning.** We use the divide-and-conquer strategy to divide \( G \) into subgraphs with similar sizes, which reduces the time cost to anonymize each vertex to be kt-safe by making each subgraph a kt-safe graph.

Given a graph \( G' \), the maximum size, \( y \), of subgraphs, and the number, \( s \), of partitions, we have an algorithm to produce a
set, $G$, of partitioned subgraphs. Please refer to Algorithm 5 in our supplemental materials for details, available online.

Note that, after we divide $G$ into subgraphs $g_i$, we will expand each subgraph $g_i$ by including duplicated $n$-hop neighbors of border vertices $v \in g_i$. We will conduct the anonymization process on each expanded subgraph, and then merge the anonymized subgraphs.

**Algorithm 2. kt-safety_vertex($v_i$, $v_f$, $CS$, $G$)**

**Input:** a graph $G$, a vertex $v_i$, a candidate set $v_i, CS$, and thresholds $k_t, e, \alpha$

**Output:** a $kt$-safe vertex $v_i$

1. for each $v_m \in v_i, CS$ do
2. 
3. for $L = 1$ to $n$ do
4. for $j = 1$ to $d - 1$ do
5. if $\text{att_dist}(HN(v_i, L), HN(v_m, L), A_j) > t$ then
6. $\text{att_safe} = \text{false}$
7. break;
8. if $\text{att_safe} = \text{false}$ then
9. break;
10. if $\text{att_safe} = \text{true}$ then
11. $v_i, PS \leftarrow v_i, PS \cup \{v_m\}$

// make $v_i$ satisfy $[v_i, PS] \geq k$
12. if $[v_i, PS] \in [0, k]$ then
13. $N_{\text{needed}} \leftarrow k - |v_i, PS|$
14. for each $v_m \in (CS - v_i, PS)$ do
15. if $N_{\text{needed}} = 0$ then
16. break;
17. if $v_m$ can be a member of $v_i, PS$ via adding fake nodes and edges then
18. $v_i, PS \leftarrow v_i, PS \cup \{v_m\}$
19. for each vertex $v_f \in Q_G$ prior to $v_i$ in $Q_G$ do
20. if $\text{graph edit operators over vertex } v_f \text{ affect the privacy of } v_i$ then
21. make a duplicate $v_f'$ of $v_f$
22. proceed graph edit operators over $v_f'$
23. $N_{\text{needed}} \leftarrow N_{\text{needed}} - 1$
24. if $N_{\text{needed}} > 0$ then
25. add $N_{\text{needed}}$ fake duplicate vertices $v_f'$ of existing vertices $v_m \in v_i, PS$ with non-sensitive value on sensitive attribute $A_i$, via fake vertex/edge/attribute insertion
26. $v_i, PS \leftarrow v_i, PS \cup \{v_f'\}$
// make $v_i$ satisfy $P_{\text{sens}}(v_i, PS, A_i) \leq \alpha$
27. if $P_{\text{sens}}(v_i, PS, A_i) > \alpha$ then
28. add $x = \left(\frac{\alpha - \text{att_safe}}{\alpha - \text{att_safe}} - [v_i, PS]\right)$ fake duplicate vertices $v_f'$ of existing vertices $v_m \in v_i, PS$ with non-sensitive value on sensitive attribute $A_i$, via fake vertex/edge/attribute insertion
29. $v_i, PS \leftarrow v_i, PS \cup \{v_f'\}$

**Discussions on Parameters $\gamma$ and $s$.** Please refer to Appendix A.5 in our supplemental materials, available online, for the discussions on parameters $\gamma$ and $s$.

### 4.2 $kt$-Safe Graph Generation Phase

#### 4.2.1 Candidate Set Detection

For any vertex $v_i$, in order to obtain its protection set $v_i, PS$ (as given in Definition 3), we will first retrieve its initial candidate set, $v_i, CS$, of vertices $v_m$ that satisfy the first two conditions (w.r.t. quasi-identifier and graph structure) in Definition 3, which we can further anonymize to obtain $v_i, PS$ (satisfying the third condition). Please refer to Algorithm 6 in our supplemental materials for details, available online.

Note that, if many vertices $v_m \in V(G)$ have the same quasi-identifier (i.e., $v_m, Q_I$) as $v_i$, we will obtain many candidates in $v_i, CS$, which leads to a lower anonymization cost to anonymize $v_i$ as a $kt$-safe vertex later in Section 4.2.2.

#### 4.2.2 $kt$-Safe Vertex Generation

Algorithm 2 illustrates the details of the $kt$-safe graph generation phase, which obtains a protection set $v_i, PS$ of each vertex $v_i$, based on the candidate set $v_i, CS$, and protects the identity and sensitive attribute values of $v_i$.

*Initialization of the Protection Set $v_i, PS$.* In Algorithm 2, we first compute an initial protection set $v_i, PS$ of vertex $v_i$ (based on the candidate set $v_i, CS$) that satisfies the three conditions in Definition 3 (lines 1-11). Specifically, candidate vertices $v_m$ in $v_i, CS$ satisfy the first two conditions of Definition 3. Thus, we only need to further check whether vertices $v_m$ also satisfy the third condition (i.e., similar quasi-identifiers in $L$-hop neighbors of vertex $v_i$).

Specifically, for each vertex $v_m \in v_i, CS$, we check the similarity of value distributions on attribute $A_j$ (for $1 \leq j \leq d - 1$) between $L$-hop neighbors of $v_i$ and $v_m$ (for $1 \leq L \leq n$; lines 1-9). If it holds that $\text{att_dist}(HN(v_i, L), HN(v_m, L), A_j) \leq t$ for all non-sensitive attributes $A_j$ ($1 \leq j \leq d - 1$) and all $L$-hop neighbors (i.e., $\text{flag att_safe} = \text{true}$ holds), then we will add $v_m$ to the protection set $v_i, PS$ (lines 10-11).

*Identity Protection of $v_i$.* After obtaining an initial protection set $v_i, PS$, we next aim to protect the identity of vertex $v_i$, by letting $[v_i, PS] \geq k$ hold (i.e., the first condition of the $kt$-safe vertex in Definition 4).

Specifically, if the initial protection set $v_i, PS$ satisfies the condition $|v_i, PS| \geq k$, then we do nothing; otherwise, we will let the remaining candidates in set $(v_i, CS - v_i, PS)$ become protection vertices for $v_i$ (lines 12-26). That is, when $|v_i, PS|$ is less than $k$, we will obtain $N_{\text{needed}} = k - |v_i, PS|$ more vertices as protection vertices, either from $(v_i, CS - v_i, PS)$ (lines 14-23) and/or by adding fake duplicate nodes (lines 24-26).

For the remaining vertices $v_m \in (v_i, CS - v_i, PS)$ that do not satisfy the third condition (w.r.t. attribute distributions) in Definition 3, we will attempt to add new nodes (and edges as well) to their $L$-hop neighbors to make them the members of $v_i, PS$ (i.e., $v_m \in v_i, PS$). Specifically, we will do a rehearsal to add some fake nodes and edges to make $v_m$ satisfy the third condition of Definition 3, where the addition operations are finally executed only if they will not make $v_m$ violate the first two conditions of Definition 3. If we can transform $v_m$ to a member of $v_i, PS$, then we will add it to $v_i, PS$ and update those vertices $v_i$ affected by node/edge insertions for $v_m$, via vertex/edge duplication (lines 17-23). In particular, the anonymization operations of $v_m \in (v_i, CS - v_i, PS)$ may affect the privacy (i.e., identity and sensitive attribute values) of those $kt$-safe vertices $v_i$ prior to $v_i$ in the queue $Q_G$. Thus, in the case that graph edit operators on a vertex $v_f$ affect $v_i$, we will duplicate $v_f$ and obtain $v_f'$ to proceed the operators (lines 19-22). That is, the anonymization operations of a vertex do not affect the privacy of previously anonymized vertices. When
we include a new vertex $v_m \in v_i, PS$, we will decrease the vari-
able $N_{\text{needed}}$ by 1 (line 23). The iteration process terminates,
when $N_{\text{needed}} = 0$ holds (lines 15-16) or there are no more ver-
tices in $(v_i, CS - v_i, PS)$ (line 14).

After checking all vertices in $v_i, CS$, if we still need more ver-
tices in the protection set $v_i, PS$ (i.e., $N_{\text{needed}} > 0$), then we
have to add $N_{\text{needed}}$ fake duplicate nodes $v'_i$ of $v_i$, with
non-sensitive values on attribute $A_2$ and with the same asso-
ciated edges as $v_i$ (lines 24-25). Then, we include duplicate
nodes $v'_i$ in the protection set $v_i, PS$ (line 26).

Algorithm 3. merge_subgraphs($G'$)

**Input:** a set, $G'$, of anonymized subgraphs $g'$

**Output:** a refined $k$-safe graph $G'$

1. for any two expanded subgraphs, $g_1$ and $g_2$, from $G'$ do
2. for each common vertex $v$ or its duplicate $v'$ between $g_1$ and $g_2$ do
3. if $HN(v, n)$ and $HN(v', n)$ in $g_1$ and $g_2$ are not changed w.
t. that in initial graph $G$ then
4. merge $v$ and $v'$ into one vertex
5. else
6. keep both $v$ and $v'$ in $g_1$ and $g_2$, respectively
7. obtain a merged graph $g$ from $g_1$ and $g_2$
8. $G' \leftarrow G' \cup \{g\} - \{g_1, g_2\}$
9. return $G' \in G'$

**Sensitive Attribute Value Protection of $v_i$.** Next, we will let vert-
ex $v_i$ satisfy the second condition of the $k$-safe vertex in Defini-
tion 4 (i.e., $P_{\text{sens}}(v_i, PS, A_2) \leq \alpha$). Specifically, if it holds that
$P_{\text{sens}}(v_i, PS, A_2) > \alpha$, we will add to graph $G$ $x$ fake dupli-
cates of existing vertices $v_m \in v_i, PS$, but with different non-sensitive
values on attribute $A_2$, where $x$ is given by $\left\lceil \frac{N_{\text{needed}} - |v_i, PS|}{\alpha} \right\rceil$.

Algorithm 4. Graph Partitioning Set $G$ Selection

**Input:** a graph $G$, thresholds $\gamma$ and $\delta$, and iterations $\text{ite}$

**Output:** a good graph partitioning set $G$

1. for $i = 1$ to $\text{ite}$ do
2. if $i = 1$ then
3. $cts \leftarrow$ randomly and recursively select $|g|$ centers (verti-
ces) from $G$
4. $G \leftarrow$ obtain $|g|$ subgraphs centered at $cts$ via clustering
5. cost $\leftarrow$ estimate anonymization cost of set $G$ via Eq. (3)
6. else
7. $CTs \leftarrow$ randomly replacing a vertex in $cts$
8. $G_{\text{new}} \leftarrow$ obtain $|g|$ subgraphs centered at $CTs$ via clustering
9. $COST \leftarrow$ estimate the anonymization cost of set $G_{\text{new}}$
10. if $COST < \text{cost}$ do
11. $cts \leftarrow CTs$
12. cost $\leftarrow COST$
13. $G \leftarrow G_{\text{new}}$
14. return $G$

4.3 $k$-Safe Graph Merging Phase

In the $k$-safe graph merging phase, Algorithm 3 merges the anon-
ymized $k$-safe subgraphs from $G'$ back to one single $k$-
safe graph $G'$. In particular, for any two (expanded) $k$-
safe subgraphs, $g_1$ and $g_2$, from $G'$, we need to merge the
original vertices $v$ and their duplicates $v'$ (lines 1-6). That is,
if $HN(v, n)$ and $HN(v', n)$ are not modified (compared with
the original graph $G$), then we can merge $v$ and $v'$ into the
same vertex; otherwise, we will keep both versions (lines 3-
6). This way, we can obtain a merged graph $g$ from $g_1$ and $g_2$,
and update the set $G'$ (lines 7-8). Finally, there is only one
graph $G'$ left in $G'$, which will be returned as the refined
$k$-safe graph (line 9).

4.4 Complexity Analysis

In this subsection, we first provide the time complexity of
$k$-safe graph anonymization algorithms. The graph partition-
ing (Algorithm 5 in our supplemental materials, available
online), needs $O(|V(G)| \cdot log(|V(G)|))$ time complexity, which
includes the cost to load vertices $O(|V(G)|)$ and that to assign
vertices to $s$ subgraphs $O(|V(G)| \cdot log(|V(G)|))$. The candidate
set detection (Algorithm 6 in our supplemental materials, avail-
able online), needs $O(|V(G)| \cdot C_{\text{hop}})$ time complexity in the
worst case (i.e., all vertices share the same quasi-identifiers),
where $C_{\text{hop}}$ is the average time cost to calculate the graph
edit distance between $HN(v_i, n)$ and $HN(v_m, n)$. The $k$-safe
vertex anonymization (Algorithm 2) takes $O(|v_i, CS| \cdot |V(G)| \cdot
(|V(G)| + E(G)) + \frac{|v_i, CS|}{\alpha} \cdot (|V(G)| + E(G)))$ time complexity
in the worst case, which contains 1) the cost to initialize the pro-
tection set $v_i, PS$ of $v_i, (v_i, CS \cdot n \cdot (d - 1))$ when the neigh-
bor $HN(v_m, n)$ of all candidate vertices $v_m \in v_i, CS$ of $v_i$ have
the similar value distribution on all non-sensitive attributes
(1-11), 2) the cost to make each vertex $v_m \in v_i, CS - v_i, PS$, a member of $v_i, PS$ $O(|v_i, CS| \cdot |V(G)| \cdot (|V(G)| + E(G)))$ where the anonymization of $v_m \in v_i, CS$ affects the pri-

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
supplemental materials, available online, for the anonymization cost evaluation.

5 OPTIMIZATIONS FOR THE kt-SAFE GRAPH ANONYMIZATION

5.1 Cost Model for Graph Partitioning

In the preprocessing phase in Section 4.1, we need to partition graph \( G \) into disjoint subgraphs of similar sizes. Since the anonymization cost highly depends on the graph partitioning strategy, in this subsection, we propose a cost model to estimate the anonymization cost of a graph partitioning strategy \( G' \), and guide the graph partitioning process.

The basic idea of our cost model is as follows. We retrieve a (small) sample graph \( S \) from \( G \), and anonymize \( S \) as a \( kt \)-safe graph \( S' \) via our proposed approach (Algorithm 1). The total cost of anonymizing \( S \) comes from two parts, anonymization and merging. For anonymization cost, it is the sum of anonymization cost \( u.\text{Cost}_\text{mk} \) for making each vertex \( u \in S \) to be \( kt \)-safe. For merging cost, it equals to the sum of cost \( u'.\text{Cost}_\text{merge} \) for merging each border vertex \( u' \in S \) and \( S' \). Then, given a graph partitioning strategy, we estimate the anonymization cost \( \text{Cost}(G, G') \) of \( G \) in Eq. (2), by summing up the approximate cost of each vertex \( v \in G \) to be \( kt \)-safe and that of each border vertex \( v' \in G \) to be merged based on the sample \( S \). Specifically, we estimate the anonymization cost, \( \text{Cost}(G, G') \), below.

\[
\text{Cost}(G, G') = \text{Cost}_{\text{anonymization}} + \text{Cost}_{\text{merging}} = \sum_{v \in G} \{u.\text{Cost}_\text{mk}[\arg\min_u \sum_{u \in S} \partial(u, v, n)]\} + \sum_{v \in G} \{u'.\text{Cost}_\text{merge}[\arg\min_{u'} \sum_{u' \in S} \partial(u', v', n)]\},
\]

where \( G \) is a sample graph from \( G \) containing vertices \( u, u'. \text{Cost}_\text{mk} \) is the anonymization cost of sample \( u \) to be \( kt \)-safe, \( u'.\text{Cost}_\text{merge} \) is the cost of border vertices \( u' \) and their duplicates to be merged, and function \( \partial(u, v, n) = |V(\text{HN}(u, n))| - |V(\text{HN}(v, n))| \) calculates the difference of vertex numbers from subgraphs \( \text{HN}(u, n) \) and \( \text{HN}(v, n) \).

For each subgraph \( g_i \subseteq G \), based on the processing order \( Q_i \), we estimate the \( kt \)-safety cost of each vertex \( v \in g_i \) as the cost \( u.\text{Cost}_\text{mk} \) of sample \( u \in S \) to be \( kt \)-safe, where \( u \) is the most similar sample to \( v \) among all samples in \( S \) w.r.t. their \( n \)-hop neighbor size (i.e., \( u = \arg\min_u \sum_{v \in S} \partial(u, v, n) \)). Similarly, for each border vertex \( u' \) in any two subgraphs \( g_i \) and \( g_k \), we estimate its merging cost via the cost \( u'.\text{Cost}_\text{merge} \) of the most similar border sample \( u' \) to \( u' \) in \( S \) to be merged.

Therefore, given a graph partitioning set \( G \), we can use Eq. (3) to estimate the anonymization cost (i.e., Eq. (2)). Intuitively, a good partitioning set will have a low estimated anonymization cost via Eq. (3). We can also adopt existing graph partitioning approaches (e.g., [28]) to accelerate the graph anonymization, as long as the estimated anonymization cost via Eq. (3) remains low.

Cost-Model-Based Graph Partitioning. Given a graph \( G \), thresholds \( \gamma \) and \( s \), and an iteration \( \text{ite} \), we have Algorithm 4 to select a good graph partitioning set \( G \), based on the cost model in Eq. (3) (lines 1-14). Specifically, in the first iteration (i.e., \( \text{ite} = 1 \)), we randomly and recursively select \( |G| \) (determined by parameters \( \gamma \) and \( s \)) centers, \( cts \), from \( G \), and we obtain \( |G| \) subgraphs via clustering, in which cluster nodes in \( G \) have the shortest path to a center in \( cts \) (lines 2-4). Note that, if the size of a subgraph reaches \( \gamma \), we will not assign more vertices to this subgraph. Then, we use Eq. (3) to estimate the anonymization cost for anonymizing and merging subgraphs in \( G \) (line 5). In the remaining iterations (i.e., \( \text{ite} \neq 1 \)), we obtain a new set, \( CTS \), of centers (i.e., vertices) by randomly replacing a vertex in \( cts \), obtain a new subgraph set \( G_{new} \) based on centers in \( CTS \), and estimate the anonymization cost of the set \( G_{new} \) via Eq. (3) (lines 6-9). We choose the partitioning set (either \( G \) or \( G_{new} \)) with lower cost (lines 10-13). Finally, we return the set \( G \) with the minimal cost from \( \text{ite} \) iterations (line 14).

5.2 Pruning Strategy

In the candidate set detection phase in Section 4.2.1 (i.e., Algorithm 6 in Appendix A.6 of our supplemental materials, available online), given a vertex \( v_i \in V(G) \), in order to obtain its candidate set \( v_i \).CS, we need to compute the graph edit distance \( \text{ged}(\text{HN}(v_i, n), \text{HN}(v_m, n)) \) between \( n \)-hop neighbors of \( v_i \) and each vertex \( v_m \in V(G) \), which is rather costly. Assume that there is a vertex \( v_p \) whose \( n \)-hop neighbors \( \text{HN}(v_p, n) \) can be used as a pivot subgraph. Then, we can reduce the distance computation cost above via the triangle inequality by utilizing the pivot subgraph \( \text{HN}(v_p, n) \) to filter out those vertices \( v_m \) that cannot be in the candidate set \( v_i \).CS.

The Pruning via Pivots. We have the pruning lemma below to rule out vertices \( v_m \in G \) whose \( n \)-hop neighbor subgraphs are dissimilar to \( \text{HN}(v_i, n) \).

**Lemma 5.1 (Pivot Pruning).** Given two vertex \( n \)-hop neighbor subgraphs \( \text{HN}(v_i, n) \) and \( \text{HN}(v_m, n) \), and a pivot \( n \)-hop neighbor subgraph \( \text{HN}(v_p, n) \), vertex \( v_m \) can be safely pruned (i.e., \( v_m \notin v_i \).CS), if it holds that \( |\text{ged}(\text{HN}(v_i, n), \text{HN}(v_p, n)) - \text{ged}(\text{HN}(v_m, n), \text{HN}(v_p, n))| > \epsilon \).

**Proof.** Please refer to Appendix A.2 in the supplemental materials, available online. \( \square \)

In practice, we can select a set, \( PVTs \), of pivots \( v_p \in G \) whose \( n \)-hop neighbor subgraphs \( \text{HN}(v_p, n) \) can be adopted to enable the pivot pruning (as discussed in Lemma 5.1). If \( v_m \) can be pruned via any pivot \( v_p \in PVTs \), then \( v_m \) is not a candidate of the set \( v_i \).CS. Please refer to Appendix A.8 in our supplemental materials, available online, for the details of our pivot set selection strategy.

5.3 kt-Tree

Based on the selected pivot set \( PVTs \) (via Algorithm 7 in Appendix A.8 in our supplemental materials, available online), we propose a tree synopsis, \( kt \)-tree (\( kt \)-T), to further accelerate the retrieval process of candidates in \( v_i \).CS for each vertex \( v_i \in G \).

**kt-Tree Synopsis.** The \( kt \)-tree, \( kt \)-T, is a tree index built over graph \( G \), where each non-leaf node \( N \) contains \( m \) children nodes \( N_i \) (for \( 1 \leq i \leq m \)), and each child node \( N_i \) exclusively includes around \((1/m \times 100)\%\) similar vertices in \( N \). Specifically, each leaf node in \( kt \)-T includes vertices \( v \in G \) with the same or similar quasi-identifiers. Moreover, we use a hash function, \( f(v, A_i) \), to hash value \( v, A_i \) of a vertex \( v \in G \) on non-sensitive attributes \( A_i \) (for \( 1 \leq i \leq d - 1 \)) into a position...
TABLE 2
The Tested Data Sets

| Data Sets  | No. of Nodes | No. of Edges |
|------------|--------------|--------------|
| Cora [27]  | 2,708        | 5,278        |
| DBLP [37]  | 79,593       | 201,334      |
| Epinions [26] | 22,166     | 355,813      |
| Facebook   | 4,039        | 88,234       |
| Wikipedia  | 7,115        | 103,689      |
| Arxiv      | 23,133       | 93,497       |
| Uniform    | 10,000       | 97,779       |
| Gaussian   | 10,000       | 163,973      |
| Zipf       | 10,000       | 56,067       |

in a bit vector with size $B$, where $B > 1$ is an integer value. Each node $N \in ktT$ contains two information:

1. $(d - 1)$ bit vectors, $A_j.vec$, obtained via bit OR operators over all hash bit values of $v.A_j$ of vertex $v \in N$; and
2. a distance interval, $pvt.I$, of $n$-hop neighbor subgraphs in vertices in $N$ and tree node $N$.

Index Construction. We build the $kt$-tree index, $ktT$, by clustering vertices in $G$ in a bottom-up strategy. First, we cluster all vertices $v \in G$ based on their quasi-identifier $v.QI$, and treat these clusters as the leaf nodes of $ktT$. For each leaf node including vertices $v$ with the same quasi-identifiers, we can get $(d - 1)$ vectors $A_j.vec$ via hash function $f(v.A_j)$. Given $[PVTs]$ pivots $pvt$ obtained from Algorithm 7 , we can group leaf nodes (containing vertices with the same quasi-identifier) into $[PVTs]$ parent nodes via clustering, based on the graph edit distance between $n$-hop neighbor subgraphs of vertices $v$ and pivots $piv$. Furthermore, we select $\sqrt{|PVTs|}$ pivots from the $[PVTs]$ nodes, and cluster these nodes into $\sqrt{|PVTs|}$ supernodes with similar sizes. This process terminates until we reach the root of the $ktT$. After we build the $ktT$, for each node in $ktT$, we calculate its stored information (i.e., $A_j.vec$ and $pvt.I$).

Vertex Pruning via $ktT$. With the $ktT$, for each vertex $v \in G$, when we retrieve its candidate set $v.CS$, we can prune vertices $v'$ in the leaf nodes $N \in ktT$ that cannot contain the same quasi-identifier $v'.QI$ as $v.QI$, where bit and operators over the $(d - 1)$ vectors $A_j.vec$ of $v$ and $N$ return false. For vertices $v'$ in the un-pruned leaf nodes $N \in ktT$, we calculate the graph edit distance between the $n$-hop neighbor subgraph of vertex $v$ and pivot $pvt$ (of node $N$), and then check whether or not $v'$ can be pruned via the pivot $pvt$ and the correspondingly stored distance interval $pvt.I$ of node $N$ (Lemma 5.1).

6 EXPERIMENTAL EVALUATION

6.1 Experimental Settings

Real/Synthetic Data Sets. We evaluate the performance of our $kt$-safe approach on 6 real and 3 synthetic data sets, as depicted in Table 2. Please refer to Appendix B in our supplemental materials, available online, for the detailed description of these tested data sets.

Competitors. We compare our $kt$-safe graph generation approach, denoted as partition + pruning, with four baseline approaches, namely $kl$-graph, partition, pruning, and none, where partition + pruning is our approach that applies both graph partition and pruning strategies ($n = 1$ by default), $kl$-graph uses an existing $k$-degree-$l$-diversity [33] graph mechanism ($l = 2$ by default), partition considers our proposed graph partitioning approach only, pruning uses our proposed pruning strategy only, and none do not leverage any optimization method.

Parameter Settings. Table 3 depicts experimental settings, where default values are in bold. We run our experiments on a PC with Intel(R) Core(TM) i7-6600U CPU 2.70 GHz and 32 GB memory. All algorithms were implemented by C++. The code is available at http://www.cs.kent.edu/~wren/kt-safety/.

6.2 Utility Evaluation

Due to space limitations, we report the utility result of our $kt$-safe graph generation approach on Epinions in this section and put the results of the other 8 data sets to Appendix C.2 in our supplemental materials, available online. In order to improve readability, we only report the utility evaluation results of the approaches partition + pruning and pruning, since the anonymized graphs via partition + pruning and pruning have the same utility performance as that of partition and none, resp.

Utility Evaluation versus Vertex Degree Distribution. Fig. 4a evaluates the degree distributions of our proposed $kt$-safe graph generation approach over Epinions. Specifically, we illustrates the degree distribution of the original graph, which

| Parameters                    | Values |
|-------------------------------|--------|
| identity privacy threshold $k$ | 5, 10, 15, 20 |
| sensitive privacy threshold $\alpha$ | 0.1, 0.2, 0.3 |
| graph edit distance threshold $\epsilon$ | 3, 4, 5, 6, 7 |
| the size, $|QI|$, of vertices’ quasi-identifiers | 10, 20, 40, 80, 100 |

TABLE 3 The Parameter Settings

| Parameters                    | Values |
|-------------------------------|--------|
| the maximum size, $\gamma$, of subgraphs | 500, 1000, 1500, 2000 |
| the number, $s$, of graph partitions | 2, 3, 4, 5, 6 |
| the size, $|V(G)|$, of the graph | 10K, 20K, 40K, 80K, 100K |
can be regarded as a standard. In figures, we can see that the generated $k$-safe graphs by our methods (including the 3 variants) vividly reveal the degree distribution trend of original graph. Moreover, from figures, partition + pruning and partition add more fake nodes in the anonymization process than pruning and none, since they need more fake nodes to make each partitioned subgraph as $k$-safe.

Utility Evaluation versus Shortest Path Length Distribution.

Fig. 3b demonstrates the distributions of shortest path lengths among 1,000 randomly selected vertex pairs over Epinions. From figures, all the four approaches perform well for revealing the shortest path length distribution of the unanonymized graph. This is reasonable, since we only consider addition operations in graphs.

Please refer to Appendix C.2 in our supplemental materials, available online, for the evaluation of distributions of degrees and shortest path length among 1,000 randomly selected vertices and vertex pairs resp., and the largest component size over real/synthetic data.

### 6.3 Comparisons of Privacy Preservation Power

Table 4 depicts the percentage of $k$-safety anonymized nodes after applying $kl$-graph technique [33]. From the results, we can see that the anonymized graphs via $kl$-graph cannot fully satisfy our $k$-safety requirement, since $kl$-graph does not consider the intersection attack between quasi-identifiers (from both nodes and their neighbors) and graph structure. Note that, for an anonymized node via our $k$-safety approach, we cannot guarantee that it has at least $k$ candidates with the same degrees. However, since we do not lease the statistics of graph edit operators, the attackers cannot uniquely identify any target vertex and its sensitive attributes. Thus, $k$-safety is a better, feasible graph privacy mechanism (with stronger privacy preservation power) than $kl$-graph for attributed graphs.

### 6.4 Efficiency Evaluation

Similar to utility evaluation, we only report the time and anonymization cost on Cora, DBLP, and Epinions, and put the result on the other 6 graphs to Appendix C.3 in our supplemental materials, available online.

Efficiency Evaluation versus Data Sets.

Fig. 4a illustrates the wall clock time of anonymizing the graph for Cora, DBLP, and Epinions. From the figure, we can see that our approach, partition + pruning, can achieve the lowest wall clock time on DBLP and Epinions, which indicates the efficiency of our graph anonymization approach. Note that, partition + pruning needs more time cost than pruning on Cora, since it needs to partition graph into subgraphs prior to the $k$-safe anonymization process. This indicates that we may not need to partition graphs in the preprocessing phase for small graphs; by setting the subgraph size threshold $\gamma$ as a large value. For the three variants of our methods, partition (via divide-and-conquer) has the smallest wall clock time, which indicates the necessity of graph partitioning, especially for large graphs. Moreover, none is expected to perform the worst. Please refer to Appendix C.3 in our supplemental materials, available online, for the break-up cost analysis of partition + pruning on Cora, DBLP, and Epinions.

Anonymization Cost Evaluation.

Fig. 4b illustrates the anonymization cost of all the approaches on Cora, DBLP and Epinions. From the figure, partition + pruning and partition have slightly higher cost than pruning and none, since they need to add more fake nodes to the partitioned subgraphs than that of the original single graph. However, our approach requires lower time cost for anonymizing large graphs (as shown in Fig. 4a). Therefore, partition + pruning can be used for anonymizing large graphs (e.g., DBLP), whereas pruning can be adopted over small graphs (e.g., Cora).

Time and Anonymization Cost Evaluation versus the Size, $|QI|$, of Vertices’ Quasi-Identifiers.

Fig. 5 reports the performance of our partition + pruning approach on three synthetic data sets, by varying $|QI|$ from 10 to 100, where other parameters are set to their default values in Table 3. From the figures, we can see that larger $|QI|$ leads to both higher time and anonymization costs. This is reasonable, since larger $|QI|$ will generate a more complex (stricter) neighborhood environment for each vertex to be anonymized. Therefore, for each vertex, we need more time to check its $n$-hop neighborhood and add more fake nodes to make each vertex $k$-safe.

### 6.3 Comparisons of Privacy Preservation Power

#### Table 4: The Percentage of $k$-Safety Anonymized Nodes After Applying the $kl$-Graph

| Data Sets | Cora | DBLP | Epinions | Facebook | Wikipedia | Arxiv | Uniform | Gaussian | Zipf |
|-----------|------|------|----------|----------|-----------|-------|---------|----------|------|
| Ratio (%) | 56.53 | 45.36 | 38.32    | 38.1     | 24.53     | 51.42 | 32.1    | 34.33    | 29.51|

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
7 RELATED WORK

Privacy Preservation for Tabular Data. The k-anonymity [24] mechanism was first proposed to study the privacy issue of the released tabular table. In particular, it requires that any tuple t in the released tabular table have a candidate set t.C containing at least k indistinguishable candidate records. After that, many works with various privacy preservation constraints have been proposed, including l-diversity [21], t-closeness [18], β-preservation [22], m-invariance [30], β-likeliness [3], differential privacy [9], and so on.

Differential privacy (DP) [9] is a more rigorous privacy preservation mechanism, requiring the inference probability of the presence of any element in a data set should be smaller than \( \exp(\epsilon) \), where \( \epsilon \in (0, 1) \) is a privacy indicator. There are many existing works of the DP variant (e.g., [14]) and applications (e.g., [6], [19], [32], [35]).

Our work focuses on the privacy preservation over graph data (instead of tabular tables), which contain both vertex attributes and graph structures. We cannot directly borrow previous techniques for tabular data, due to the privacy preservation with graph structures.

Privacy Preservation for Graph Data. Existing works on the graph privacy can be classified into two categories: the ones applying the differential privacy (e.g., [4], [7], [10], [12], [13], [31]) and those adopting other privacy mechanisms like k-anonymity (e.g., [2], [5], [20], [33], [34], [36], [38]). Moreover, there are some works (e.g., [1], [11]) studying possible attacker models for anonymized graphs.

Graph privacy via differential privacy. Hay et al. [10] first applied the differential privacy on graph data, and proposed two privacy-preservation mechanisms, edge-based and node-based differential privacies, which require the presence of any edge (e.g., relationship) and node (e.g., person) not be disclosed with high inferred confidences, respectively. Then, related works on differential privacy over graphs have two directions: publishing graph statistics (e.g., degree distribution [7]) and publishing the entire graph [4], [10], [12], [13], [31].

Although differential privacy is a very robust privacy mechanism (e.g., future proofed), it may not be a good choice for an attributed graph under our attacker model (Definition 2), which is the intersection of quasi-identifiers and n-hop neighbors (including graph structure and their quasi-identifiers). Therefore, we do not consider differential privacy as a competitor/baseline in our experiments.

Graph privacy via other privacy mechanisms. Other existing works (e.g., [2], [5], [20], [33], [34], [36], [38]) mainly leveraged the idea of k-anonymity [24]. Specifically, Song et al. [23] proposed l-sensitive-label-diversity mechanism, for protecting from 1-hop neighborhood attack, but cannot resist from an attacker with n-hop (\( n > 1 \)) neighbor information, which is the focus of our work.

Prior works mainly focused on the graph structure attack only (i.e., n-hop neighbor attack) [5], [38]. Although some existing works [23], [33], [34] studied both 1-hop neighbors and attribute attacks (i.e., quasi-identifier of a vertex), they cannot protect the graph privacy if an attacker knows the vertex’s n-hop (\( n > 1 \)) neighbor information (containing both structural and quasi-identifier information). Thus, their proposed techniques cannot be directly applied to our kt-safe graph anonymization problem.

8 CONCLUSION

This paper formalizes the kt-safety graph anonymization problem for protecting the privacy of released graphs. We prove the NP-hardness of this problem, and propose a heuristic kt-safe graph generation framework that can be optimized via a cost-model-based graph partitioning strategy, and pruning and indexing mechanisms. We verified the performance of our proposed approach through extensive experiments.
REFERENCES

[1] L. Backstrom, C. Dwork, and J. Kleinberg, “Wherefore art thou r3579x? Anonymized social networks, hidden patterns, and structural steganography,” in Proc. 16th Int. Conf. World Wide Web, 2007, pp. 181–190.

[2] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa, “Injecting uncertainty in graphs for identity obfuscation,” Proc. VLDB Endowment, vol. 5, pp. 1376–1387, 2012.

[3] J. Cao and P. Karras, “Publishing microdata with a robust privacy guarantee,” 2012, arXiv:1208.0220.

[4] R. Chen, B. C. Fung, P. S. Yu, and B. C. Desai, “Correlated network data publication via differential privacy,” Int. J. Very Large Data Bases, vol. 23, pp. 653–676, 2014.

[5] J. Cheng, A. W.-C. Fu, and J. Liu, “K-isomorphism: Privacy preserving network publication against structural attacks,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2010, pp. 459–470.

[6] G. Cormode, T. Kulkarni, and D. Srivastava, “Marginal release under local differential privacy,” in Proc. Int. Conf. Manage. Data, 2018, pp. 131–146.

[7] W.-Y. Day, N. Li, and M. Lyu, “Publishing graph degree distribution with node differential privacy,” in Proc. Int. Conf. Manage. Data, 2016, pp. 123–138.

[8] K. Doka, M. Xue, D. Tsoumakos, and P. Karras, “k-anonymization by formerror generalization,” in Proc. 10th ACM Int. Conf. Comput. Commun. Secur., 2015, pp. 519–530.

[9] C. Dwork, “Differential privacy,” in Encyclopedia of Cryptography and Security. Berlin, Springer, 2011.

[10] M. Hay, C. Li, G. Miklau, and D. Jensen, “Accurate estimation of the degree distribution of private networks,” in Proc. IEEE 9th Int. Conf. Data Mining, 2009, pp. 169–178.

[11] M. Hay, G. Miklau, D. Jensen, D. Towsley, and P. Weis, “Resisting structural re-identification in anonymized social networks,” Proc. VLDB Endowment, vol. 1, pp. 102–114, 2008.

[12] Z. Jorgensen, T. Yu, and G. Cormode, “Publishing attributed social graphs with formal privacy guarantees,” in Proc. Int. Conf. Manage. Data, 2016, pp. 107–122.

[13] D. Kifer and A. Machanavajjhala, “No free lunch in data privacy,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2011, pp. 193–204.

[14] I. Kotsogiannis, S. Doudalis, S. Haney, A. Machanavajjhala, and S. Mehrotra, “One-sided differential privacy,” in Proc. IEEE 36th Int. Conf. Data Eng., 2020, pp. 493–504.

[15] J. Leskovec, D. Huttenlocher, and J. Kleinberg, “Predicting positive and negative links in online social networks,” in Proc. 19th Int. Conf. World Wide Web, 2010, pp. 641–650.

[16] J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graph evolution: Densification and shrinking diameters,” ACM Trans. Knowl. Discov. Data, vol. 1, pp. 2–es, 2007.

[17] J. Leskovec and J. J. Mcauley, “Learning to discover social circles in ego networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 539–547.

[18] N. Li, T. Li, and S. Venkatasubramanian, “t-Closeness: Privacy beyond k-anonymity and l-diversity,” in Proc. IEEE 23rd Int. Conf. Data Eng., 2007, pp. 106–115.

[19] Z. Li, T. Wang, M. Lopuhaä-Zwakenberg, N. Li, and B. Škoric, “Estimating numerical distributions under local differential privacy,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2020, pp. 621–635.

[20] K. Liu and E. Terzi, “Towards identity anonymization on graphs,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2008, pp. 93–106.

[21] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam, “l-Diversity: Privacy beyond k-anonymity,” in Proc. IEEE 22nd Int. Conf. Data Eng., 2006, pp. 24–24.

[22] M. E. Nergiz, M. Atzori, and C. Clifton, “Hiding the presence of individuals from shared databases,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2007, pp. 665–676.

[23] Y. Song, P. Karras, Q. Xiao, and S. Bressan, “Sensitive label privacy protection on social network data,” in Proc. Int. Conf. Sci. Statist. Database Manage., 2012, pp. 562–571.

[24] L. Sweeney, “k-anonymity: A model for protecting privacy,” Int. J. Uncertain. Fuzziness Knowl.-Based Syst., vol. 10, pp. 537–570, 2002.

[25] C.-H. Tai, P.-J. Tseng, S. Y. Philip, and M.-S. Chen, “Identity protection in sequential releases of dynamic networks,” IEEE Trans. Knowl. Data Eng., vol. 26, no. 3, pp. 635–651, Mar. 2014.

[26] J. Tang, H. Gao, H. Liu, and A. Das Sarma, “eTrust: Understanding trust evolution in an online world,” in Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2012, pp. 253–261.

[27] C. Wang, S. Pan, R. Hu, G. Long, J. Jiang, and C. Zhang, “Attributed graph clustering: A deep attentional embedding approach,” 2019, arXiv:1906.06532.

[28] L. Wang, Y. Xiao, B. Shao, and H. Wang, “How to partition a billion-node graph,” in Proc. IEEE 30th Int. Conf. Data Eng., 2014, pp. 569–579.

[29] R. C.-W. Wong, A. W.-C. Fu, K. Wang, and J. Pei, “Minimality attack in privacy preserving data publishing,” in Proc. 33rd Int. Conf. Very Large Data Bases, 2007, pp. 543–554.

[30] X. Xiao and Y. Tao, “M-invariance: Towards privacy preserving re-publication of dynamic datasets,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2007, pp. 689–700.

[31] Y. Yang, Z. Zhang, G. Miklau, M. Winslett, and X. Xiao, “Differential privacy in data publication and analysis,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2012, pp. 601–606.

[32] L. Yu, L. Liu, C. Pu, M. E. Gursoy, and S. Truex, “Differentially private model publishing for deep learning,” in Proc. IEEE Symp. Secur. Privacy, 2019, pp. 332–349.

[33] M. Yuan, L. Chen, S. Y. Philip, and T. Yu, “Protecting sensitive labels in social network data anonymization,” IEEE Trans. Knowl. Data Eng., vol. 25, no. 3, pp. 633–647, Mar. 2013.

[34] M. Yuan, L. Chen, and P. S. Yu, “Personalized privacy protection in social networks,” Proc. VLDB Endowment, vol. 4, pp. 141–150, 2010.

[35] J. Zhang, X. Xiao, and X. Xie, “PrivTree: A differentially private algorithm for hierarchical decompositions,” in Proc. Int. Conf. Manage. Data, 2016, pp. 155–170.

[36] A. Zhou and J. Pei, “Preserving privacy in social networks against neighborhood attacks,” in Proc. IEEE 24th Int. Conf. Data Eng., 2008, pp. 506–515.

[37] Y. Zhou, H. Cheng, and J. X. Yu, “Graph clustering based on structural/attribute similarities,” Proc. VLDB Endowment, vol. 2, pp. 718–729, 2009.

[38] L. Zou, L. Chen, and M. T. Özsu, “K-automorphism: A general framework for privacy preserving network publication,” Proc. VLDB Endowment, vol. 2, pp. 946–957, 2009.

Weilong Ren received the PhD degree in computer science from Kent State University, in 2021. He is now a research scientist with the Shenzhen Institute of Computing Sciences, Shenzhen, China. His current research interests include incomplete data management, data integration, and data privacy.

Kambiz Ghazinour is an associate professor with The State University of New York, Canton. He is directing the Advanced Information Security and Privacy Research Lab and has advised more than 40 research projects related to data security and privacy. He has more than 60 peer reviewed publications and is actively working on privacy-awareness and educating people with usable security and privacy techniques and detect privacy breaches and recommend better privacy practices for the users whose information is widely (and mostly unintentionally) available throughout the Web.

Xiang Lian received the BS degree from the Department of Computer Science and Technology, Nanjing University, China, and the PhD degree in computer science from the Hong Kong University of Science and Technology, Hong Kong. He is now an associate professor with the Computer Science Department, Kent State University, Kent, Ohio. His research interests include probabilistic/uncertain/inconsistent databases and uncertain/certain graph databases.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.