Reversible Data Perturbation Techniques for Multi-level Privacy-preserving Data Publication

Chao Li, Balaji Palanisamy, Prashant Krishnamurthy

University of Pittsburgh
{chl205,bpalan,prashk}@pitt.edu

Abstract. The amount of digital data generated in the Big Data age is increasingly rapidly. Privacy-preserving data publishing techniques based on differential privacy through data perturbation provide a safe release of datasets such that sensitive information present in the dataset cannot be inferred from the published data. Existing privacy-preserving data publishing solutions have focused on publishing a single snapshot of the data with the assumption that all users of the data share the same level of privilege and access the data with a fixed privacy level. Thus, such schemes do not directly support data release in cases when data users have different levels of access on the published data. While a straightforward approach of releasing a separate snapshot of the data for each possible data access level can allow multi-level access, it can result in a higher storage cost requiring separate storage space for each instance of the published data. In this paper, we develop a set of reversible data perturbation techniques for large bipartite association graphs that use perturbation keys to control the sequential generation of multiple snapshots of the data to offer multi-level access based on privacy levels. The proposed schemes enable multi-level data privacy, allowing selective deperturbation of the published data when suitable access credentials are provided. We evaluate the techniques through extensive experiments on a large real-world association graph dataset and our experiments show that the proposed techniques are efficient, scalable and effectively support multi-level data privacy on the published data.

1 Introduction

Rapid advancements in data mining and data analytics techniques have made it possible to extract insights previously considered impossible. There is thus a significant incentive for collection and analysis of user information, some of which could be potentially sensitive and private [3, 8]. Data Privacy is a crucial barrier to data analysis due to privacy risks [1, 24]. Private data often arises in the form of associations between entities in real world such as medicines purchased by patients in a pharmacy store, films rated by users in a movie rating website or products purchased online by users. Such real-world associations are commonly represented as large, sparse bipartite graphs [5] with nodes representing the entities (e.g., patients and medicines) and edges representing the associations between them (e.g., medicines purchases by patients).

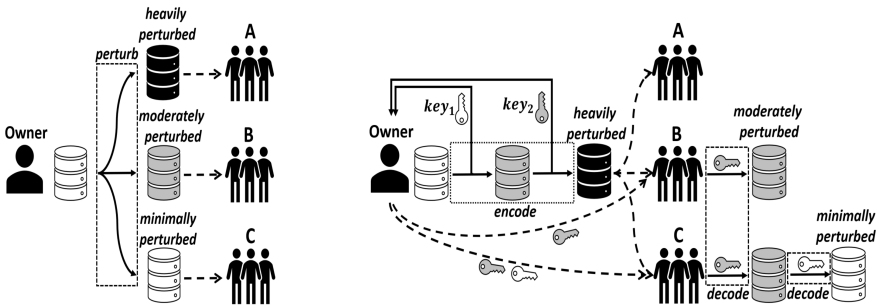
Privacy-preserving data disclosure schemes aim at publishing sensitive datasets such that the private information contained in the published data can not be inferred. These techniques primarily perturb the raw datasets to meet the privacy requirements before the data is published. While there have been several efforts on privacy-preserving data publishing in the past, most solutions have focused on publishing a single snapshot of a dataset that offers a fixed privacy level with the assumption that all users of the data share the same level of privilege to access the data [7, 9, 12, 13, 16, 17, 20–22]. Here a privacy level may directly correspond to an access privilege level. Hence, such schemes do not directly support data release in cases when data users have different levels of access on the published dataset.

In this paper, we develop a set of reversible data perturbation techniques for large bipartite association graphs that use perturbation keys to control the sequential generation of multiple snapshots of the perturbed data to offer multi-level access based on privacy levels. We assume that sensitive information in a dataset may arise either as: (i) an individual sensitive value indicating an individual's private information (e.g., did buyer 'Alice' purchase the medicine 'Citalo'?) or (ii) a statistical value representing some sensitive statistics about a group/subgroup of individuals (e.g., the total number of antidepressant medicines purchased by buyers in a given neighborhood represented by a zipcode). In many real-world scenarios, users of a dataset may have different privileges and may need to access the same dataset at different privacy/utility levels requiring multi-level access on the published data. For example, a data owner may prefer to share a sensitive dataset with a reputed data analytics team with a low degree of perturbation. However, the same data owner may release the dataset with a higher level of perturbation to a less privileged data analyst and the data may be disclosed to the general public with an even higher level perturbation to protect the data privacy further. The proposed reversible data perturbation schemes enable multi-level data access, allowing selective de-perturbation of the published dataset to reduce the degree of perturbation when suitable access credentials are provided. We evaluate the proposed techniques through extensive experiments on a large real-world association graph dataset. Our experiments show that the proposed techniques are efficient, scalable and effectively enable multi-level data privacy on the published data.

The rest of the paper is organized as follows. We introduce the key concepts and the differential privacy model in Section 2. The proposed reversible data perturbation techniques is discussed in Section 3. We present the experimental evaluation of the proposed techniques in Section 4. The related work is discussed in Section 5 and we finally conclude in Section 6.

2 Concepts and models

In this section, we first present the general idea behind the multi-level privacy-preserving data publishing problem. We then introduce the graph representation of association data used in our work and briefly review the notion of differential privacy.



(a) Conventional data perturbation

(b) Reversible data perturbation

Fig. 1: Data perturbation schemes to support multi-level privacy-preserving data publication

2.1 Multi-level Data Access using Reversible data perturbation

Privacy-preserving data publishing (PPDP) schemes are designed to prevent the inference of sensitive information in published datasets from data users accessing the published information. Dataset owners use privacy-preserving data publishing (PPDP) techniques to perturb their datasets prior to publishing. In many real-world scenarios, users of a dataset may have different privileges and may need to access the same dataset at different privacy/utility levels requiring multi-level access on the published data. The multi-level privacy-preserving data publishing requirement can be accomplished with a straight-forward approach of releasing a snapshot of the dataset for each privacy level using conventional data perturbation techniques such as differential privacy [7]. In the example shown in Figure 1(a), the data owner is willing to share a minimally perturbed snapshot of the data with data user C and wants to share a moderately perturbed snapshot with data user B and a heavily perturbed snapshot with data user A. The perturbed snapshots for each of the data users can be generated by running a conventional data perturbation technique for each data user with the privacy parameters corresponding to the privilege level of the user. While this straight-forward approach achieves the multi-level data disclosure objective, it can however result in a higher storage cost requiring a replication of the dataset for each possible privacy level.

In this paper, we propose the concept of reversible data perturbation (Figure 1(b)) that allows a data publisher to release data at multiple privacy levels using a single instance of the perturbed dataset. The reversible data perturbation approach consists of an encoding phase and a decoding phase. The dataset encoding is performed by the dataset owner. During the encoding process, the perturbed snapshots are first sequentially generated in order from the lowest privacy level with the least perturbation to the highest privacy level that requires a larger degree of perturbation. Between any two adjacent snapshots in the sequence, a perturbation key is used to pseudo-randomly generate the sequence of perturbation operations that transform the given snapshot into the next snapshot in the sequence.

In Figure 1(b), we find that the data owner performs the encoding process to obtain the heavily perturbed dataset along with the two perturbation keys,

key_1 and key_2, used in the process. After the encoding process, only the two keys need to be shared by the data owner and the heavily perturbed dataset is published to all users, namely users A, B and C. Later, when the data needs to be accessed at a privilege level, the data owner shares the relevant perturbation keys with the data user. In the example shown in Figure 1(b), the data owner shares key_2 with data user B and key_1 and key_2 with data user C. Data user B can use key_2 to remove the noise and perturbation pseudo-randomly injected during the encoding process. Similarly, using key_1 and key_2, data user C can de-perturb the dataset further to obtain the minimally perturbed snapshot. Thus, in the reversible data perturbation approach, the data owner only creates one perturbed version of the dataset and uses perturbation keys that significantly reduce the storage cost associated with multi-level data sharing.

2.2 Bipartite Association Graphs

We use bipartite association graph datasets as the candidate data in the proposed reversible perturbation techniques. Several private data in real world arise in the form of associations between entities such as the drugs purchased by patients in a pharmacy store or the movies rated by viewers in a movie rating database or the products purchased by buyers in an online shopping website [4, 10, 11]. Such associations are best captured as bipartite association graphs with nodes representing the entities (e.g., drugs and patients) and the edges correspond to the associations between them (e.g., Patient Bob purchased the Insulin drug). A bipartite graph can be represented as $BG = (V, W, E)$, which consists of $m = |V|$ nodes of a first type, $n = |W|$ of a second type and a set of edges $E \subseteq V \times W$. Thus, a bipartite graph can represent a set of two-node pairings, where a two-node pairing (a, b) represents an edge between node $a \in V$ and node $b \in W$.

2.3 Differential privacy

Next, we define the notion of Differential privacy that we use in the reversible data perturbation approach. Differential privacy [7] is a state-of-the-art privacy paradigm that makes conservative assumptions about the adversary's background knowledge and protects a single individual's information in a dataset by considering adjacent datasets which differ only in one record.

Definition 1 (Differential privacy [7]). *A randomized algorithm $\mathcal{A}$ guarantees ϵ-differential privacy if for all adjacent data sets D_1 and D_2 differing by at most one record, and for all possible results $\mathcal{S} \subseteq Range(\mathcal{A})$, $Pr[\mathcal{A}(D_1) = \mathcal{S}] \leq e^{\epsilon} \times Pr[\mathcal{A}(D_2) = \mathcal{S}]$*

The conventional (individual) differential privacy protects the inference of a single individual's information in a dataset. For example, in a bipartite graph representing the associations between drugs and patients, such a single individual's protection may correspond to the inference of the graph edge representing a patient (e.g., 'Alice') purchasing a drug (e.g.,'Citalo'). For the purpose of protecting sensitive information of a group of individuals (e.g., the total number of

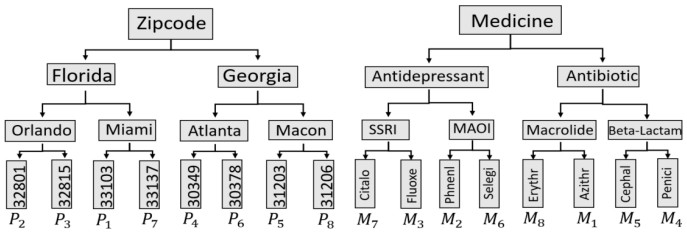


Fig. 2: Taxonomy trees

cancer medicines purchased by patients in a specific neighborhood), differential privacy can be further extended to support group data protection based on the notion of group differential privacy [19].

Definition 2 (Group differential privacy [19]). *A randomized algorithm $\mathcal{A}$ guarantees ϵ_g-group differential privacy if for all adjacent data sets D_1 and D_2 differing by at most one group $G_i \in G$, and for all possible results $\mathcal{S} \subseteq Range(\mathcal{A})$, $Pr[\mathcal{A}(D_1) = \mathcal{S}] \leq e^{\epsilon_g} \times Pr[\mathcal{A}(D_2) = \mathcal{S}]$*

Group differential privacy protects sensitive aggregate information about groups of records using higher noise injection and perturbation. When records of a dataset are grouped into larger groups, the transformed dataset will provide coarser aggregate information and the privacy offered by group differential privacy will be stronger. Therefore, by grouping the records of a dataset into multiple granularity levels, different privacy levels can be offered by implementing group differential privacy at different granularity levels in the dataset. In our work, we employ both individual and group differential privacy in the proposed reversible data perturbation process to provide multi-level disclosure of the data using a single instance of the perturbed dataset.

3 Reversible data perturbation techniques

In this section, we present the details of the reversible data perturbation process and illustrate the encoding and decoding steps involved in the data perturbation.

3.1 Overview of dataset encoding process

The overall encoding phase in the reversible data perturbation process can be viewed as a sequence of permutation and noise injection steps. Figure 3 illustrates the process with an example bipartite graph dataset where the original version of the bipartite graph is shown as snapshot S_0, which consists of eight left (patient) nodes denoted by *PID*, eight right (medicine) nodes denoted by *MID* and eleven edges representing which medicine was purchased by which patient. To protect group differential privacy, the bipartite graph is first partitioned into multiple levels of subgraphs representing different granularity levels based on the taxonomy tree shown in Figure 2. If such a taxonomy tree is not available apriori, the dataset corresponding to the bipartite graph can be partitioned through a sequence of specializations based on granular subgraph generation techniques such as the one presented in [19]. In the Figure 3 example, at level $L_{2\times2}$, both the left and right nodes are grouped into groups of two nodes and thus it generates sixteen subgraphs. At level $L_{4\times4}$, nodes are grouped into groups of four nodes

and therefore it generates four subgraphs. At level $L_{8\times8}$, nodes are grouped into groups of eight nodes, which results in a single graph. Based on the partitioning, dataset owners can choose to make a low perturbed snapshot, S_1 at $L_{2\times2}$, a moderately perturbed snapshot, S_2 at $L_{4\times4}$ and a heavily perturbed snapshot, S_3 at $L_{8\times8}$.

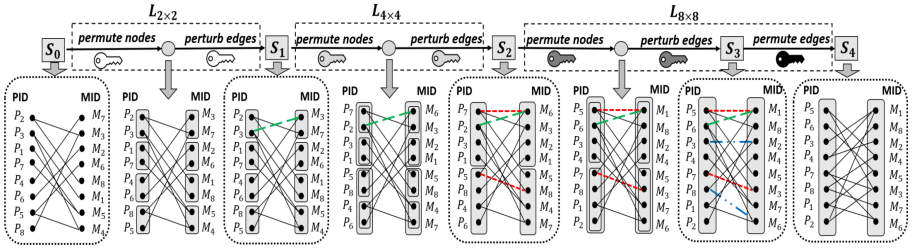


Fig. 3: Steps to encode a bipartite graph dataset

To generate each perturbed snapshot mentioned above, there is one step of (node) permutation followed by one step of (edge) perturbation. The purpose of node permutation is to ensure information generalization. For example, at snapshot S_0, left nodes P_2, P_3 and right nodes M_7, M_3 form a subgraph contained by $L_{2\times2}$, which has a single edge (P_2, M_3). Without node permutation, specific information in S_0, such as the edge (P_2, M_3) that indicates P_2 purchased M_3, can be viewed by users who only have privilege to view S_1 to learn generalized information about subgraphs at $L_{2\times2}$. In contrast, by permuting P_2, P_3 and also by permuting M_7, M_3 within their size-two groups, the label M_3 is swapped with M_7. Thus, instead of edge (P_2, M_3), a fake edge (P_2, M_7) indicating incorrect specific information is contained in S_1, whereas generalized information about the subgraph (e.g., one patient in Orlando purchased one SSRI medicine) is still maintained in S_1. This process is followed by the edge perturbation process which aims to prevent specific information to be inferred from the generalized information in the exposed snapshot. For example, after node shuffling, the subgraph between P_2, P_3 and M_3, M_7 shows generalized information that one patient in Orlando has purchased an SSRI medicine. It has four possibilities, namely (P_2, M_3), (P_2, M_7), (P_3, M_3) and (P_3, M_7). An adversary with some background knowledge may infer that (P_2, M_7), (P_3, M_3) and (P_3, M_7) cannot exist and therefore will be able to conclude the existence of edge (P_2, M_3) from the generalized information. To address this concern, edge perturbation can be used to perturb the edges of each subgraph based on randomized mechanisms (e.g., Laplace Mechanism [7]). In the example, users receiving S_1 can also view the injected edge (P_3, M_3) and thus learn that two patients in Orlando purchased SSRI medicine. Since the noise injection is based on differential privacy, the users viewing this data cannot tell whether the two edges are true edges or injected edges and therefore feel uncertain to conclude the existence of (P_2, M_3). After the two steps, S_1 can be generated, which reveals generalized information about subgraphs at $L_{2\times2}$ while protecting individual information in S_0. Similarly, after another round of two steps, S_2 reveals generalized information of $L_{4\times4}$ while protecting specific information of $L_{2\times2}$. Similarly, S_3 reveals $L_{8\times8}$ information while protecting information of $L_{4\times4}$. At the end of the encoding phase, if S_3 still

contains sensitive information about $L_{8\times8}$ that the data owner is not willing to share to all possible users, edge permutation can be executed over S_3 to permute all the edges in S_3 so that the obtained S_4 is safe for disclosure.

To generate snapshot S_i from S_{i-1}, a *perturbation key* is used to first pseudo-randomly permute the two sides of nodes of each subgraph and then pseudo-randomly perturb edges within each subgraph. Also, edge permutation at the last step is pseudo-randomly implemented using a perturbation key. Next, we will show how to use perturbation keys to perform edge perturbation, node permutation and edge permutation steps in the reversible perturbation approach so that legitimate users can use perturbation keys to decode S_4 to any previous snapshots (e.g., S_0, S_1, S_2, S_3) containing finer information.

Algorithm 1: Noise injection

Input : Bipartite graph $BG = (V, W, E)$, sensitivity $\triangle f$, budget ϵ, key K.
Output: Perturbed bipartite graph $\widetilde{BG}$.

$n = \lfloor LaplaceRandom(0, \triangle f/\epsilon, K) \rfloor$;
Initialize counter $c = 0$, index $i = 0$, new edge recorder $\overline{NE}$, skipped index recorder SI;
while $c < n$ **do**
  $ne = (rand(2i, K)\ mod\ |V|, rand(2i+1, K)\ mod\ |W|)$;
  if $ne \notin E \cup \overline{NE}$ **then**
    $\overline{NE} \leftarrow ne$; $c++$;
  end
  else
    $\overline{SI} \leftarrow i$;
  end
  i++;
end
$\widetilde{BG} = (V, W, E \cup \overline{NE})$;

Algorithm 2: Noise removal

Input : Perturbed bipartite graph $\widetilde{BG}$, sensitivity $\triangle f$, budget ϵ, key K, skipped index recorder SI.
Output: Bipartite graph BG.

$n = \lfloor LaplaceRandom(0, \triangle f/\epsilon, K) \rfloor$;
Initialize index $i = 0$;
while $i < n + |SI|$ && $i \notin SI$ **do**
  $re = (rand(2i, K)\ mod\ |V|, rand(2i+1, K)\ mod\ |W|)$;
  Remove edge re from $\widetilde{BG}$;
  i++;
end
$BG = \widetilde{BG}$;

3.2 Reversible edge perturbation

For each subgraph, the reversible edge perturbation step first uses the perturbation key to pseudo-randomly sample a noise using the Laplace Mechanism [7]. When the sampled noise is positive, the procedures of noise injection and noise

removal are performed as shown in Algorithm 1 and Algorithm 2 respectively. During noise injection, the number of injected edges is sampled from Laplace pseudo-random value generator with mean 0, variance $\triangle f/\epsilon$ and seed K, where K is the key (line 1). Then, during each loop (line 3-12), two pseudo-random numbers are used to select one left node and one right node from the subgraph to form a new edge ne (line 4). If ne is not an existing edge, its selection will be confirmed (line 5-7); otherwise, this iteration will be skipped to avoid collision and this skipped index will be recorded into a list that will be attached with the key to be used during the decoding process later (line 8-10). The algorithm complexity is $O(n)$. After this process, legitimate users can receive the perturbation key to reversibly remove the injected noise using Algorithm 2. With the same seed K, same n can be generated (line 1), which can then select and remove the same sequence of edges with assistance of SI (line 3-7). The complexity of this algorithm is $O(n + |SI|)$.

However, when noises are negative, instead of using $|SI|$ to record the skipped iterations, we need to record all removed edges using the perturbation key.

Algorithm 3: Node permutation: encoding

Input : Bipartite graph $BG = (V, W, E)$, key K.
Output: Permuted bipartite graph $\overline{BG}$.

1 $R = PseudoRandom(K)$;
2 **for** $i = 0; i < |V|; i++$ **do**
3 | Swap node $V[i]$ and node $V[R[i]\ mod\ |V|]$;
4 **end**
5 **for** $i = |V|; i < |V| + |W|; i++$ **do**
6 | Swap node $W[i - |V|]$ and node $W[R[i]\ mod\ |W|]$;
7 **end**

Algorithm 4: Node permutation: decoding

Input : Permuted bipartite graph $\overline{BG} = (V, W, E)$, key K.
Output: Bipartite graph BG.

1 $R = PseudoRandom(K)$;
2 **for** $i = |V| - 1; i \geq 0; i--$ **do**
3 | Swap node $V[i]$ and node $V[R[i]\ mod\ |V|]$;
4 **end**
5 **for** $i = |V| + |W| - 1; i \geq |V|; i--$ **do**
6 | Swap node $W[i - |V|]$ and node $W[R[i]\ mod\ |W|]$;
7 **end**

3.3 Reversible node permutation

For each subgraph, the reversible node permutation step uses the perturbation key to pseudo-randomly shuffle node labels (e.g., PID, MID) during the encoding phase and later uses the same key to recover their order during the decoding process. We show its procedures in the encoding phase and decoding phase in Algorithm 3 and Algorithm 4 respectively.

During the encoding phase, the perturbation key is used as a seed in the pseudo-random stream generator to generate a sequence of pseudo-random numbers denoted as R (line 1). Then, the first $|V|$ pseudo-random numbers in R are

used to shuffle the left nodes in BG (line 2-4) while the pseudo-random numbers generated later, namely $|W|$ are used to shuffle the right nodes (line 5-7). Each pseudo-random number swaps two left (right) nodes. The first node is selected from top to bottom along with its position while the second node is pseudo-randomly selected by the pseudo-random number using modular arithmetic. At the end of Algorithm 3, both left nodes and right nodes are shuffled in a reversible manner. Later, during decoding phase, given the same key, the same R can be obtained (line 1). The same two groups of pseudo-random numbers in R are used to recover left nodes (line 2-4) and right nodes (line 5-7) respectively. The process within each loop is quite similar to that of the encoding process. However, instead of starting from top to bottom, during decoding process, the loops start from bottom to top with a reverse order so that operations implemented during encoding can be reversibly implemented during decoding, which results in the recovery of the original subgraph. Here, both the algorithms have a complexity of $O(|V| + |W|)$.

In Figure 4, the labels of the nodes are permuted through reversible node permutation while the edges are permuted through reversible edge permutation (to be discussed later). In the example, Alice uses a perturbation key as a seed to the pseudo-random stream generator to get a sequence of pseudo-random numbers R. Then, the first six pseudo-random numbers (assumed to be $[35, 18, 46, 12, 27, 57]$) and second six pseudo-random numbers (assumed to be $[7, 18, 24, 29, 62, 67]$) in R are used to shuffle the left and right nodes of the bipartite graph respectively. Based on Algorithm 3, the first pseudo-random number $R_1 = 35$ swaps P_2 and P_3, followed by 18 swapping P_3 and P_6, 46 swapping P_4 and P_5, 12 swapping P_6 and P_5, 27 swapping P_4 and P_6, 57 swapping P_2 and P_4. As a result, left nodes in the left bipartite graph are permuted to the order in the right bipartite graph. Then, in Figure 5, Bob gets the permutation key from Alice and uses the key as a seed and generates the same R as generated by Alice. Among the first six pseudo-random numbers $[35, 18, 46, 12, 27, 57]$, $R_6 = 57$ is first picked to swap P_2 and P_4, followed by 27 swapping P_4 and P_6, 12 to swapping P_6 and P_5 and so on. Therefore, the original order of the left nodes can be recovered.

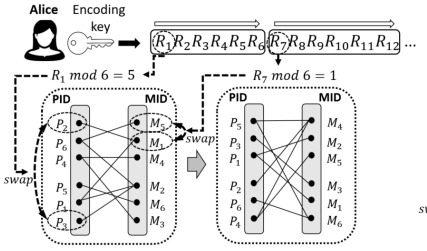


Fig. 4: Encoding

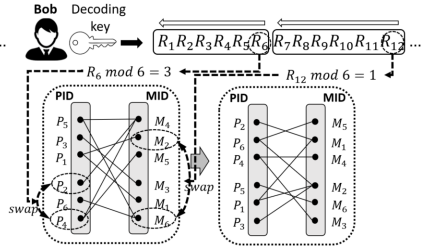


Fig. 5: Decoding

3.4 Reversible edge permutation

Edge permutation is implemented as the last step in the encoding phase and therefore it represents the first step during the decoding phase. The edges of the bipartite graph are represented using an adjacency matrix. For example, the

adjacency matrix of the left bipartite graph in Figure 4 is shown as the matrix E below, where the first row represents that P_2 is linked with M_1. Here, the edges can be shuffled by simply permuting the adjacency matrix.

$$E = \begin{bmatrix} 0&1&0&0&0&0 \\ 1&0&0&1&0&0 \\ 0&0&1&0&0&1 \\ 0&0&0&0&1&0 \\ 0&1&0&1&0&0 \\ 0&0&0&1&0&0 \end{bmatrix} \quad \overline{E} = \begin{bmatrix} 1&0&0&0&0&1 \\ 0&0&0&1&0&0 \\ 0&1&0&0&1&0 \\ 1&0&0&0&0&0 \\ 0&0&0&0&0&1 \\ 1&0&1&0&0&0 \end{bmatrix}$$

The encoding and decoding parts of the process are shown in Algorithm 5 and Algorithm 6 respectively. Similar to node permutation, in both the algorithms, same R can be obtained through the perturbation key (line 1). Then, we use the first $|V||W|$ pseudo-random numbers in R to perform $|V||W|$ rounds of swap operation (line 2-4). Each time, the first edge is selected based on a fixed order (top to bottom and left to right in Algorithm 5, right to left and bottom to top in Algorithm 6) and the second edge is pseudo-randomly selected by the pseudo-random number using modular arithmetic. In this way, by reversibly performing the swap operation during the decoding phase, the original order of the edges can be recovered. Here, the algorithms have a complexity of $O(|V||W|)$.

Algorithm 5: Edge permutation: encoding

Input : Bipartite graph $BG = (V, W, E[|V|][|W|])$, key K.
Output: Permuted bipartite graph $\overline{BG}$.

1 $R = PseudoRandom(K)$;
2 **for** $i = 0; i < |V||W|; i++$ **do**
3 Swap edge $E[\lfloor \frac{i}{|W|} \rfloor][i \ mod \ |W|]$ and edge $E[\lfloor \frac{R[i] \ mod \ |V||W|}{|W|} \rfloor][R[i] \ mod \ |V||W|) \ mod \ |W|]$;
4 **end**

Algorithm 6: Edge permutation: decoding

Input : Permuted bipartite graph $\overline{BG} = (V, W, E[|V|][|W|])$, key K.
Output: Bipartite graph BG.

1 $R = PseudoRandom(K)$;
2 **for** $i = |V||W| - 1; i \geq 0; i--$ **do**
3 Swap edge $E[\lfloor \frac{i}{|W|} \rfloor][i \ mod \ |W|]$ and edge $E[\lfloor \frac{R[i] \ mod \ |V||W|}{|W|} \rfloor][R[i] \ mod \ |V||W|) \ mod \ |W|]$;
4 **end**

In Figure 4, if the first and second pseudo-random numbers generated by a key are 53 and 71, we first use 53 to swap $E[\lfloor \frac{0}{6} \rfloor][0 \ mod \ 6] = E[0][0]$ and $E[\lfloor \frac{53 \ mod \ 36}{6} \rfloor][(53 \ mod \ 36) \ mod \ 6] = E[2][5]$. Then, we use 71 to swap $E[0][1]$ and $E[5][5]$. By repeating this for all the 36 pseudo-random numbers, the adjacency matrix can be transformed as $\overline{E}$ to represent the right bipartite graph in Figure 4.

4 Experimental Evaluation

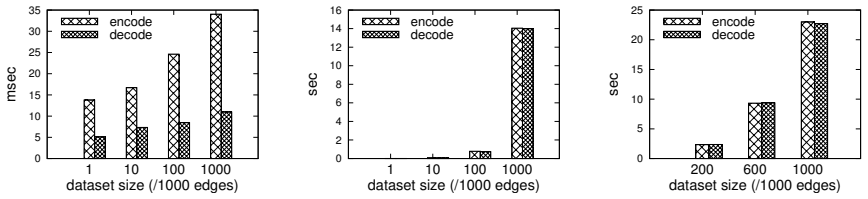
In this section, we present the experimental study on the performance of the proposed reversible data perturbation techniques. We first briefly describe the experimental setup.

4.1 Experimental setup

Our experiment setup was implemented in Java with an Intel Core i7 2.70GHz 16GB RAM PC. The bipartite graph dataset used in this work is the MovieLens dataset [10] which consists of 6,040 users (left nodes), 3,706 movies (right nodes) and 1,000,209 edges describing rating of movies made by users.

4.2 Experimental results

The experimental results are organized into three parts. First, we evaluate the performance efficiency of the three key components of the reversible perturbation process separately, namely edge perturbation, node permutation and edge permutation. Then, we integrate the three components and evaluate the performance of the complete reversible data perturbation process. In our experiments, we generate three granularity levels and evaluate the time and space consumption for each granularity level during encoding and decoding phases. Finally, we evaluate the utility and privacy protection offered by the data perturbation process. We demonstrate that the noise injected for protecting differential privacy does not substantially reduce the utility of the data.

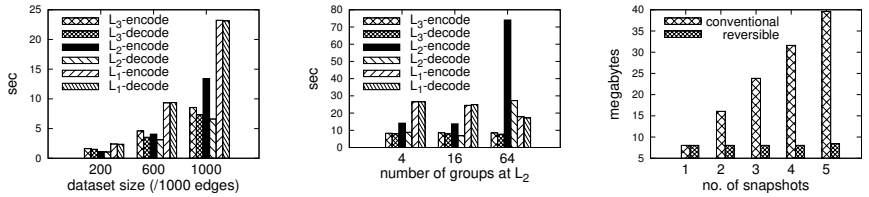


(a) edge perturbation (b) node permutation (c) edge permutation

Fig. 6: Algorithm performance

Algorithm performance The first set of experiments evaluates the performance efficiency of edge perturbation, node permutation and edge permutation separately. We evaluate the scalability of these algorithms by varying the size of dataset and we measure the time taken for their execution both in encoding and decoding phases. In Figure 6(a), edge perturbation is evaluated. The dataset size is changed from one thousand edges to one million edges. Specifically, the one-million-edge dataset represents the entire MovieLens dataset. The results show that both noise injection (encode) and removal (decode) processes have significantly low time consumption cost and demonstrate high scalability. Even when the dataset size increases 1000 times, the time consumption increases only by a factor of 2. For a dataset with one million edges, the noise injection and removal processes cost only about 35ms and 10ms respectively. Here, compared with noise injection, the noise removal process usually has a lower time consumption. This is because the process of noise removal employs some meta data

information attached to the perturbed dataset which significantly accelerates its speed of execution. Next, in Figure 6(b), we evaluate the node permutation process with the same experiment setting. Unlike edge perturbation, although the time consumption of node permutation is significantly small for small datasets, it becomes acceptably larger for the one-million-edge dataset, which is about 14s. Finally, in Figure 6(c), we measure the time taken by the edge permutation process using dataset sizes that vary from 0.2 million edges to 1 million edges. The results show that the time taken by the process for the one-million-edge dataset is about 23s, which is quite acceptable as the edge shuffling process is only required to be run once during the entire process.

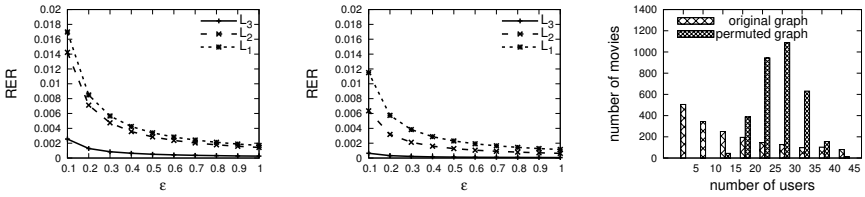


(a) time consumption w/ varying size (b) time consumption w/ varying partitioning (c) comparison of storage cost

Fig. 7: Multi-level performance

Multi-level performance The second set of experiments evaluate the performance of the multiple levels of perturbation during the process. In this part, we processed the dataset to generate three granularity levels of subgraphs, denoted as L_1, L_2 and L_3 respectively. We applied the partitioning algorithm proposed in [19] to generate the granularity levels. The algorithm runs several rounds of specializations and each specialization can partition a bipartite graph into four non-overlapping subgraphs. Therefore, after n rounds of specializations, the original bipartite graph has been partitioned to 4^n non-overlapped subgraphs. In this experiment, we use the MovieLens dataset and we consider the entire graph as level L_1, the 16 (4^2) subgraphs generated by two specializations as level L_2 and the 256 (4^4) subgraphs generated by four specializations as level L_3. For ease of understanding, L_1, L_2 and L_3 can be considered to roughly correspond with $L_{8\times8}$, $L_{4\times4}$ and $L_{2\times2}$ in the example of Figure 3. In Figure 7(a), we evaluate the encoding and decoding time for each granularity level when the dataset size is varied from 0.2 million edges to 1 million edges. As can be seen, as the dataset size increases, the time taken by all the three granularity levels also show a reasonable increase. Level L_3 needs to run edge perturbation and node permutation over the 256 subgraphs. Due to the very small subgraph size and the low sensitivity for protecting differential privacy for individual edges, L_3 has the lowest time consumption. At level L_2, although the number of subgraphs reduces to 16, the corresponding increase in subgraph size makes its time consumption higher than that of L_3 for large dataset size. Finally, the time consumption of level L_1 is dominated by edge permutation, which follows the same trend as shown in Figure 6(c). In Figure 7(b), we fix the dataset size as one million edges while changing the number of subgraphs at level L_2 from 16 to 4 and 64. This change

at L_2, as shown in Figure 7(b), has no influence on the results of L_3. The reduction from 16 to 4 makes an increase for both L_2 and L_1 while the increase from 16 to 64 makes results at L_2 significantly increased and results at L_1 obviously decreased. These results show that instead of the average size of subgraphs, the time consumptions of granularity levels are much more sensitive to the amount of the injected noises. Finally, in Figure 7(c), we compare the storage cost required by conventional framework and reversible framework. Based on Figure 3, three granularity levels can generate at most five snapshots. As can be seen, using the conventional framework, the storage cost is linearly increased with the number of generated snapshots as data owner needs to store all of them. However, the proposed reversible framework efficiently employs the use of perturbation keys to allow all snapshots to be recovered from a single published snapshot protected with the highest privacy level. Thus, the data owner only needs to store one snapshot. The size of the perturbation keys and the stored metadata for noise injection have little influence on the overall storage cost.



(a) noise error (256 groups) (b) noise error (16 groups) (c) edge shuffling

Fig. 8: Utility and security

Utility and security In the final set of experiments, we evaluate the utility and privacy offered by the reversible multi-level data perturbation scheme. In the edge perturbation process, a large quantity of noise may have an impact on the utility of published dataset. To evaluate the data utility, we measured the relative error rate (RER) that represents the ratio of the sum of the error caused by the noise in each subgraph of a given level and the overall number of edges in the original bipartite graph. In Figure 8(a), we measure RER of the three granularity levels when L_2 and L_3 have 16 and 256 subgraphs respectively. As can be seen, when the privacy budget ϵ is 1, RER of all the three levels is very small. Even if ϵ is decreased to a very strict value 0.1, the highest RER 0.017 appears at L_1, which is still acceptable. In Figure 8(b), we reduce number of subgraphs at L_2 and L_3 to 4 and 16 respectively. The results show that this reduction makes RER of all the three levels lower. Finally, in Figure 8(c), we evaluate the effectiveness of the edge permutation process by measuring the distribution of the degree of nodes before and after edge permutation. As can be seen, the distribution in the perturbed graph after edge permutation is substantially different than that of the original graph. This makes it harder to infer useful or true information after the edge permutation process.

5 Related Work

The problem of information disclosure has been studied extensively in the framework of statistical databases. Samarati and Sweeney [21, 22] introduced the k-

anonymity approach which has led to some new techniques and definitions such as l-diversity [17] and t-closeness [16]. There had been some work on anonymizing graph datasets with the goal of publishing statistical information without revealing information of individual records. Backstrom et al. [2] show that in fully censored graphs where identifiers are removed, a large enough known subgraph can be located in the overall graph with high probability. The safe grouping techniques proposed in [5] consider the scenario of retaining graph structure but aim at protecting privacy when labeled graphs are released. But, as mentioned earlier, these existing schemes have been focused on publishing a single instance of the perturbed dataset with a fixed privacy level without considering the requirements of multiple access levels. The key focus of this work is on developing a reversible data perturbation approach for bipartite association graph data that can facilitate the release of multiple levels of information using a single instance of the perturbed data, similar to the reversible location perturbation techniques recently proposed in [14, 15].

Based on the concept of differential privacy [7], there had been many works focused on publishing sensitive datasets through differential privacy constraints [6, 9, 12, 20, 23]. Recent work had focused on publishing graph datasets through differential privacy constraints so that the published graph maintains as many structural properties as possible while providing the required privacy [20]. However, these existing schemes do not support multi-level access to the published dataset. The notion of group differential privacy and granular subgraph generation algorithms for bipartite graphs is recently introduced in [18, 19]. This paper extends the work presented in [18, 19] with a suite of reversible data perturbation techniques that provides a more scalable and cost-effective solution to releasing bipartite association graph data at multiple privacy levels using a single instance of the perturbed dataset.

6 Conclusion

Privacy-preserving data publishing techniques are critical for protecting sensitive information in published datasets. Existing solutions have focused on publishing a single snapshot of the perturbed dataset that offers a fixed privacy level with the assumption that all users of the data share the same privilege level to access it. In cases when data users have different levels of access on the published data, such schemes will require multiple snapshots corresponding to different privacy levels to be published and maintained, resulting in higher storage cost for the data. In this paper, we develop a set of reversible data perturbation techniques for large bipartite association graphs that use perturbation keys to control the sequential generation of multiple snapshots of the perturbed data to offer multi-level access to the data based on privacy levels. To support multi-level privacy, the proposed techniques require only a single snapshot of the data to be maintained which significantly reduces the storage cost. We evaluate the proposed reversible data perturbation techniques through experiments on a real large bipartite association graph dataset. The experiments demonstrate that the proposed techniques are scalable, effective and efficiently support multi-level data access using a single snapshot of the perturbed data.

Reference

1. Bigdata and future of privacy. https://epic.org/privacy/big-data/.
2. Lars Backstrom et al. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW*, pages 181–190, 2007.
3. Michael Batty. Big data, smart cities and city planning. *Dialogues in Human Geography*, 3(3):274–279, 2013.
4. O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.
5. Graham Cormode et al. Anonymizing bipartite graph data using safe groupings. *Proceedings of the VLDB Endowment*, 1(1):833–844, 2008.
6. Wei-Yen Day, Ninghui Li, and Min Lyu. Publishing graph degree distribution with node differential privacy. In *ICMD*, pages 123–138. ACM, 2016.
7. Cynthia Dwork et al. Calibrating noise to sensitivity in private data analysis. In *TCC*, volume 3876, pages 265–284. Springer, 2006.
8. Wei Fan and Albert Bifet. Mining big data: current status, and forecast to the future. *ACM SIGKDD Explorations Newsletter*, 14(2):1–5, 2013.
9. Arik Friedman and Assaf Schuster. Data mining with differential privacy. In *SIGKDD*, pages 493–502. ACM, 2010.
10. F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM TIIS*, 5(4):19, 2016.
11. Ruining He et al. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*, pages 507–517, 2016.
12. Vishesh Karwa et al. Private analysis of graph structure. *Proceedings of the VLDB Endowment*, 4(11):1146–1157, 2011.
13. Shiva Prasad Kasiviswanathan et al. Analyzing graphs with node differential privacy. In *Theory of Cryptography*, pages 457–476. Springer, 2013.
14. Chao Li and Balaji Palanisamy. De-anonymizable location cloaking for privacy-controlled mobile systems. In *NSS*, pages 449–458. Springer, 2015.
15. Chao Li and Balaji Palanisamy. Reversecloak: Protecting multi-level location privacy over road networks. In *ACM CIKM*, pages 673–682. ACM, 2015.
16. Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115. IEEE, 2007.
17. Ashwin Machanavajjhala et al. l-diversity: Privacy beyond k-anonymity. In *ICDE*, pages 24–24. IEEE, 2006.
18. Balaji Palanisamy, Chao Li, and Prashant Krishnamurthy. Group differential privacy-preserving disclosure of multi-level association graphs. In *ICDCS*, pages 2587–2588. IEEE, 2017.
19. Balaji Palanisamy, Chao Li, and Prashant Krishnamurthy. Group privacy-aware disclosure of association graph data. *IEEE Big Data*, 2017.
20. Alessandra Sala et al. Sharing graphs using differentially private graph models. In *SIGCOMM*, pages 81–98. ACM, 2011.
21. Pierangela Samarati. Protecting respondents identities in microdata release. *IEEE transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.
22. Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.
23. Qian Wang et al. Rescuedp: Real-time spatio-temporal crowd-sourced data publishing with differential privacy. In *INFOCOM*, pages 1–9. IEEE, 2016.
24. Xindong Wu et al. Data mining with big data. *IEEE transactions on knowledge and data engineering*, 26(1):97–107, 2014.