Non-homogeneous Generalization in Privacy Preserving Data Publishing[*]

W. K. Wong, Nikos Mamoulis and David W. Cheung
Department of Computer Science, The University of Hong Kong
Pokfulam Road, Hong Kong
{wkwong2,nikos,dcheung}@cs.hku.hk

ABSTRACT

Most previous research on privacy-preserving data publishing, based on the k-anonymity model, has followed the simplistic approach of homogeneously giving the same generalized value in all quasi-identifiers within a partition. We observe that the anonymization error can be reduced if we follow a non-homogeneous generalization approach for groups of size larger than k. Such an approach would allow tuples within a partition to take different generalized quasi-identifier values. Anonymization following this model is not trivial, as its direct application can easily violate k-anonymity. In addition, non-homogeneous generalization allows for additional types of attack, which should be considered in the process. We provide a methodology for verifying whether a non-homogeneous generalization violates k-anonymity. Then, we propose a technique that generates a non-homogeneous generalization for a partition and show that its result satisfies k-anonymity, however by straightforwardly applying it, privacy can be compromised if the attacker knows the anonymization algorithm. Based on this, we propose a randomization method that prevents this type of attack and show that k-anonymity is not compromised by it. Non-homogeneous generalization can be used on top of any existing partitioning approach to improve its utility. In addition, we show that a new partitioning technique tailored for non-homogeneous generalization can further improve quality. A thorough experimental evaluation demonstrates that our methodology greatly improves the utility of anonymized data in practice.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Security, integrity, and protection

General Terms

Algorithms

Keywords

Non-homogeneous generalization, privacy, anonymization

[*]Supported by grant HKU 715108E from Hong Kong RGC.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.
Copyright 2010 ACM 978-1-4503-0032-2/10/06 ...$10.00.

1. INTRODUCTION

The problem of privacy-preserving data publishing has been extensively studied since it was first introduced in [20, 21]. Consider a large table which has to be released to the public for research purposes. Privacy is typically compromised by careless publishing of the table [3], since sensitive information may be leaked. Thus, the goal of data publishing is to transform the table, such that individuals may not be linked to specific tuples with high certainty. At the same time, the published data should still be useful, so an optimization problem arises: anonymize the data such that a certain degree of privacy is preserved while data utility is maximized.

In the table to be published, apart from the keys that are suppressed before publication, there is a set of attributes called the *quasi-identifier* (QID). The QID of each tuple is known to the attacker and may be used to identify an individual. A typical example of QID is {ZIP code, gender, date of birth}, which can uniquely identify 63% of the population in 2000 US Census data [8]. The popular k-anonymity principle [20, 21] requires that the probability of an adversary being able to find out the identity of an anonymized tuple is at most $\frac{1}{k}$. The most common technique for achieving k-anonymity is generalization [13, 14, 10, 6]. The table is divided into groups having k tuples or more and the QID values in each group are generalized to a range containing all original values. Table 2 shows an exemplary 2-anonymized table using generalization. The original data are shown in Table 1. (t'_i in Table 2 is the generalized version of t_i in Table 1 for easy reference.) For example, the age of t_3 is originally 15 and after generalization, it is replaced by the range 15-30. Apart from microdata publication, k-anonymity has been largely adopted in applications like location-based services [19, 11], to protect the identity of query issuers.

A wide range of algorithms using generalization are proposed for addressing k-anonymity [10, 14, 6]. They share a common framework: first partition the tuples into groups, then assign the same generalized QID to tuples in the same group. The group of tuples with the same QID is called *equivalence class*. Such an approach, to which we refer as *homogeneous generalization*, raises an important question: does generalization have to be homoge-

	QID			Sens. attribute
Tuple ID	Zip code	Gender	Age	Disease
t_1	901152	M	30	Flu
t_2	901157	F	28	Cancer
t_3	901578	M	15	Cancer
t_4	902398	M	48	AIDS
t_5	902301	M	20	None

Table 1: Original table

Tuple ID	Zip code	Gender	Age	Disease
t'_1	901***	*	15-30	Flu
t'_2	901***	*	15-30	Cancer
t'_3	901***	*	15-30	Cancer
t'_4	9023**	M	20-48	AIDS
t'_5	9023**	M	20-48	None

Table 2: 2-anonymity using homogeneous generalization

Tuple ID	Zip code	Gender	Age	Disease
t'_1	90115*	*	28-30	Flu
t'_2	901***	*	15-28	Cancer
t'_3	901***	M	15-30	Cancer
t'_4	9023**	M	20-48	AIDS
t'_5	9023**	M	20-48	None

Table 3: 2-anonymity using non-homogeneous generalization

neous? For example, consider the possible publication of Table 1, as shown in Table 3. t'_1, t'_2 and t'_3 have a different generalized QID. This generalization is *non-homogeneous*. Assuming the adversary knows the QIDs of all individuals contained in Table 1, he can find out the identity of any anonymized tuple in Table 3 with probability at most $\frac{1}{2}$. Hence, 2-anonymity is satisfied, as this is also the case for Table 2. On the other hand, if we compare the utility of the two tables, we can observe that Table 3 is better than Table 2, regardless of the utility measure used; for each tuple and QID attribute of Table 3, the generalized range is smaller than or equal to the corresponding range in the corresponding tuple and attribute in Table 2. This example shows that it is possible to achieve higher utility using non-homogeneous generalization. The idea of non-homogeneous generalization was first introduced in [7], which studies techniques with a guarantee that an adversary cannot associate a generalized tuple to less than k individuals. However, the proposed solutions do not offer bounds for the probability of each association. Hence, some individuals may have higher probability to be associated to an anonymized tuple than others and this may lead to privacy breaches.

In this paper, we systematically study the use of non-homogeneous generalization in anonymizing tables. We provide a methodology for verifying whether a non-homogeneous generalization violates k-anonymity. Then, we propose a technique that generates a non-homogeneous generalization and show that its result satisfies k-anonymity, however by straightforwardly applying it, privacy can be compromised if the attacker knows additionally the anonymization algorithm. Based on this, we propose a randomization method that prevents this type of attack and show that k-anonymity is not compromised by it. Although non-homogeneous generalization can be used on top of any existing partitioning approach to improve its utility, we show that a new partitioning technique tailored for non-homogeneous generalization can further improve quality. Our main focus throughout the paper is k-anonymity, however, we also discuss how our methodology can be extended to improve utility for other privacy principles. A thorough experimental evaluation demonstrates that our methodology greatly improves the quality of anonymized data in practice.

The rest of the paper is organized as follows. The next section reviews related work and positions it against this paper. In Section 3, we formally define the k-anonymity problem and provide a partial ordering mechanism for comparing the utility of different anonymization results. Section 4 discusses the main challenges of non-homogeneous generalization and provides some properties that can be used to define a good generalization. Our methodology is described in Section 5. Section 6 discusses the extension of our methodology for l-diversity [16] and in Section 7 we experimentally evaluate it. Finally, Section 8 concludes the paper.

2. RELATED WORK

A privacy principle, k-anonymity, is developed in [20, 21] to guard against adversaries having the QIDs of individuals as background knowledge. The goal of k-anonymity is to prevent an adversary from identifying an individual with a probability higher than $\frac{1}{k}$. Generalization and suppression are used to protect privacy. Generalization replaces the exact QID value by a less concrete form. For example, value 15 is generalized to range [15-30]. Suppression removes some values or the entire tuple from T. The most popular method studied by the community for k-anonymity has been homogeneous generalization. The tuples in the table are partitioned into groups called equivalence classes. The QID of tuples in the same equivalence class are generalized to be the same. As finding the best partitioning that achieves k-anonymity, while maximizing utility is NP-hard [18], different fast heuristics are developed. These can be classified into two approaches: (i) global recording (e.g., [14, 13]): if any two tuples have the same QID value, they must take the same generalized QID; (ii) local recoding (e.g., [10, 28, 6]): two tuples having the same QID may be generalized differently. Local recording generally gives published tables of higher utility, due to its flexibility.

Apart from k-anonymity, there are other principles (e.g., [16, 26, 15, 25, 27, 24, 22, 17]) that target different privacy concerns and/or different adversary assumptions. The relation to be anonymized typically contains a sensitive attribute. Even if the adversary cannot associate the tuples in the published table with individuals (i.e., k-anonymity is satisfied), he may associate an individual to a particular sensitive value with high probability if there are multiple occurrences of the same sensitive value in the equivalence class where the QID of the individual belongs. For example, suppose Bob is a male of age 15 (t_3 in Table 1). Although there are 3 possible tuples $\{t'_1, t'_2, t'_3\}$ for Bob in Table 2, an attacker can derive that Bob is likely (with the high probability of $\frac{2}{3}$) to have cancer. The l-diversity principle [16, 26] aims to bound the maximum of this inference probability to be $\frac{1}{l}$. The t-closeness principle [15], on the other hand, aims to control the inference probability so that it is similar to the general distribution of the sensitive values. For example, if 90% of population in T do not smoke, the goal is to ensure that 90% of individuals in each equivalence class are non-smoking. Both l-diversity and t-closeness assume the same basic adversary capability: knowing the QID of individuals. Some works assume that an adversary may obtain additional background knowledge. For instance, [24] assumes the adversary may know the algorithm used in generalization. In [22], the adversary may corrupt some individuals, obtain the sensitive values of them, and use them to infer the remaining sensitive values in the equivalence class. Nevertheless solutions to all above problems may suffer from privacy breaches; [12] has demonstrated how to breach privacy using de-Finetti's theorem.

Perturbation [1, 5] is another technique that has been used to preserve privacy in data publishing. Recent works also use a hybrid approach, combining perturbation and generalization to preserve privacy [22]. In perturbation, noise is added to the original data, such that the resulting values randomly deviate from the original ones. Compared to generalization, Perturbation may introduce high error, especially for aggregate queries with small ranges. In addition, noise filtering techniques may be used to breach privacy [9].

The closest piece of work related to ours is [7], where non-homogeneous generalization is introduced. The principle of global $(1,k)$-anonymity is proposed, which guarantees that an individual is not associated to less than k generalized tuples. In addition, a generalization technique for global $(1,k)$-anonymity is developed. However, this work suffers from two major drawbacks. First, the principle does not ensure that an adversary associates an individual to at least k tuples with *even* probability. As a result, an anonymized tuple may have probability $> \frac{1}{k}$ to be associated with an individual; thus, global $(1,k)$-anonymity has a weaker privacy level compared to k-anonymity. Second, the proposed algorithm has a high complexity of $O(k^2 n^{2.5})$ and is thus not suitable in practice.

In this paper, our goal is to develop a methodology for non-homogeneous generalization, which improves utility while maintaining an adequate level of privacy. Our study is mainly focused on the basic k-anonymity model, the reasons being that: (i) algorithms for k-anonymity are simple and many works (e.g., [16]) have adapted them for different principles; (ii) k-anonymity is commonly used in applications like location-based services [19, 11], where there are no additional (sensitive) attributes. Apart from the basic k-anonymity model, we also consider the scenarios with stronger adversaries, with knowledge of generation algorithm (Section 4.2) and identities of some generalized tuples (corruption, Section 5.2)

3. PROBLEM DEFINITION

Consider a relational table T, in which there are 3 classes of attributes: (i) attributes that are keys in T: such attributes are removed in the published table to prevent immediate identification of individuals; (ii) attributes that are part of the quasi-identifier (QID): the QID of every individual is known to the attacker as background knowledge and can be used to link tuples in the table to individuals; (iii) attributes that are not part of a key or QID: the values of such attributes are retained in the published table.

Our goal is to generate a publishable table T^* such that (i) the k-anonymity privacy constraint is satisfied; and (ii) utility is maximized. In Section 3.1, we describe our assumptions about the adversary and define k-anonymity. In Section 3.2, we describe how the utility of different anonymized tables can be compared.

3.1 Adversary assumption and k-anonymity

We assume an adversary may obtain the value of QID and the identification of any tuple in T by sources other than T^* (e.g., a public voters table). Let H be the adversary's knowledge containing the QID and identity of all known individuals. In the worst case, the adversary may have access to the QID of every individual, thus by joining H and T^* on QID, tuples t' in T^* may be linked to individuals. k-anonymity aims at preventing the adversary from finding an individual's identity with a probability higher than $\frac{1}{k}$.

DEFINITION 1. *(Identity notion) Given two tables H, T^*, if a tuple $t_i \in H$ and a tuple $t'_j \in T^*$ belong to the same individual, we say they have the same identity, denoted as $t_i =_I t'_j$.*

DEFINITION 2. *(k-anonymity) Given a table T, assume that H is the projection of T on key and QID attributes. We say k-anonymity is preserved in an anonymized table T^* if $\forall t_i \in H, \forall t'_j \in T^*, Pr(t_i =_I t'_j) \leq \frac{1}{k}$.*

3.2 Measuring and comparing utility

Measuring the utility of an anonymized table is usually done by means of an objective information loss measure that compares T^* with T. Popular measures include the *discernibility metric* [4], which sums the squares of the equivalence class cardinalities, the *normalized certainty penalty* (NCP) [28] which is defined by the sum of QID attribute ranges in each equivalence class, and the *global certainty penalty* (GCP) [6], which is a normalized version of NCP. In Section 5.3, we provide a definition of GCP, which we use in this paper, as it affects the functionality of our data partitioning algorithm. In general, the utility of the anonymized data may not be easily captured by specific measures, as it depends on the application of the published data. Our purpose is not to limit our study to a particular utility metric but to develop a new methodology, which generally improves the utility of existing methods that apply homogeneous generalization. As generalization converts precise data to uncertain data, a method that restricts the uncertainty of each tuple compared to the result of another method is certainly better. Definition 3 formally states when one anonymized table T^*_a is strictly better than another T^*_b in terms of utility (denoted by $T^*_a \succ T^*_b$); we can use it to define a partial order for anonymization results. In this paper, we aim at finding a local-optimal solution T^*, i.e., $\forall T^*_i$ such that $T^*_i \succ T^*$, T^*_i violates k-anonymity.

DEFINITION 3. *(utility-based ordering) Consider two anonymized tuples t'_1, t'_2. We say that t'_1 preserves a better utility than t'_2, denoted by $t'_1 \succ t'_2$ if for all attributes i in the QID, $t'_1[i] \subseteq t'_2[i]$, and there is at least one attribute j in the QID, for which $t'_1[j] \subset t'_2[j]$, e.g., ⟨90115*,*,28-30⟩ (t'_1 in Table 3) $\succ$ ⟨901***,*,15-30⟩ (t'_1 in Table 2). Consider two anonymized tables T^*_a, T^*_b, both with n tuples, such that tuples $T^*_a[i]$ and $T^*_b[i]$ originate from $T[i]$, for all $i \in [1,n]$. We say T^*_a preserves a better utility than T^*_b, denoted by $T^*_a \succ T^*_b$, if $\forall i \in [1,n]$, $T^*_a[i] \succ T^*_b[i]$ or $T^*_a[i] = T^*_b[i]$ and $\exists j \in [1,n]$, such that $T^*_a[j] \succ T^*_b[j]$.*

As discussed earlier, k-anonymity can be achieved by first partitioning the data into groups and then uniformly transform the QIDs of all records in the same group to take the same generalized value. Homogeneous generalization may not produce results of the highest possible quality. In the next section, we discuss the challenges of a non-homogeneous generalization technique that could be applied alternatively.

4. CHALLENGES IN NON-HOMOGENEOUS GENERALIZATION

Assume that the background knowledge of the adversary is a table H containing the QID of every individual. Given a published table T^*, the adversary performs a *linking attack* by joining T^* with H; for each tuple t in H, the adversary finds all tuples t' in T^* such that $t[QID]$ is included in the generalized $t'[QID]$. For example, if the QID of $t \in T$ is 10 and there is a tuple $t' \in T^*$ with $t'[QID] =$ [5-20], then t' is a possible generalization of t. We call the pair $\langle t, t' \rangle$ a *match*. A *valid assignment* is a maximal 1-to-1 assignment between tuples of H and T^*. In this paper, we identify two challenges to non-homogeneous generalization. We will discuss the issue of ineffective matches when joining H and T^* in Section 4.1 and how these can be identified and eliminated. In Section 4.2, we will discuss a privacy threat, for the case where the adversary knows the algorithm, which is used to generate the anonymized table.

4.1 Pruning of ineffective matches

Intuitively, k-anonymity can be satisfied if an adversary finds that there are at least k matches related to the same tuple in H/T^*. This is the case for homogeneous generalization; given a generalized QID, any of the k or more tuples from the original table T that

Tuple ID	QID
t_1	1
t_2	2
t_3	3
t_4	4
t_5	5

(a) original table

Tuple ID	QID
t'_1	1-5
t'_2	2-3
t'_3	3-4
t'_4	3-4
t'_5	1-5

(b) anonymized table

Table 4: Non-homogeneous generalization

were grouped together and match that QID, have the same probability (at most $1/k$) to match any tuple in T^* with that generalized QID. However, the same does not apply in the non-homogeneous case. Consider the original table T shown in Table 4a and an anonymized table T^* using non-homogeneous generalization, as shown in Table 4b. For every t_i in T, there are at least two matches in T^* and vice versa. However, 2-anonymity is not satisfied. Both t_1 and t_5 in T match t'_1, and t'_5 in T^*. t_2 matches t'_1, t'_2 and t'_5 in T^*. Since t'_1 and t'_5 must be either t_1 or t_5, t_2 can only be matched to t'_2 in a valid assignment, violating 2-anonymity. We say that the match of t_2 to t'_1 (and t'_5) is *ineffective* if an adversary can eliminate such a possibility.

DEFINITION 4. *(Match and assignment) Given a table T and its anonymized table T^*, a match m is a 2-tuple $\langle t_i, t'_j \rangle$ where $t_i \in T$, $t'_j \in T^*$ and the QID of t_i is included in that of t'_j. An assignment a is a set of matches $m_i = \langle t_{x_i}, t'_{y_j} \rangle$, where $t_{x_i} \in T$, $t'_{y_j} \in T^*$, $|a| = |T|$, and for each pair of matches $m_i \in a, m_j \in a$, $x_i \neq x_j$ and $y_i \neq y_j$.*

DEFINITION 5. *(Effective match) Given a table T and its anonymized table T^*, let H be the projection of T on key and QID attributes. Given two tuples $t_i \in H$, $t'_j \in T^*$, a match $m = \langle t_i, t'_j \rangle$ is said to be effective, if and only if there exists an assignment a such that $m \in a$.*

An assignment represents the scenario that an adversary gives a unique identity to every anonymized tuple in T^* and vice versa. If a match cannot be found in any of the assignments, then the adversary will happily remove this match. If all ineffective matches are removed and there are less than k matches left for a tuple in H, k-anonymity is violated. In the following, we first discuss how to determine if a match is effective (Section 4.1.1). Then, we present the property that the generalized table should satisfy in order for all matches to be effective (Section 4.1.2).

4.1.1 Necessary condition for effective match

In order to satisfy k-anonymity, we must have at least k effective matches for each tuple in T^*. In order to determine if a match is effective, we use an *assignment graph* which is used to visualize the matches.

DEFINITION 6. *(Assignment graph) Consider a table T and its anonymized table T^*. Assume that the tuples in both tables are ordered, such that the assignment $a = \{\langle t_i, t'_i \rangle\}$ is valid, for all $t_i \in T, t'_i \in T^*$. An assignment graph $G = (V, E)$ is a directed graph with $|T|$ vertices. For $i = 1$ to $|T|$, $n_i \in V$ represents $t_i \in T$ and $t'_i \in T^*$, An edge $n_i \rightarrow n_j$ is present if and only if the QID of t_i is included in that of t'_j.*

Figure 1 shows the assignment graph constructed for Table 4b. Each edge in the assignment graph represents a match that can be found by joining T and T^*. For example, the edge from n_3 to n_4 means that t_3 in T joins to t'_4 in T^*. Next we show how to verify the effectiveness of a match in the assignment graph.

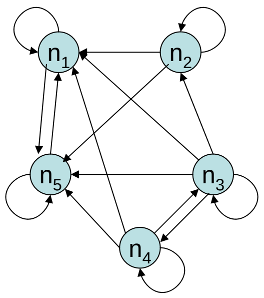

Figure 1: Assignment graph of Table 4b

THEOREM 1. *Consider a table T, its anonymized table T^*, and the corresponding assignment graph $G = (V, E)$. The match $\langle t_i, t'_j \rangle$ (corresponding to edge $(n_i, n_j) \in E$) is effective if and only if n_i is reachable from n_j.*

PROOF. If part. Note that if a match is not effective, we cannot find an assignment containing the match. So, we can prove the statement by showing how to construct an assignment that contains the target match $\langle t_i, t'_j \rangle$. Without loss of generality, we assume $j > i$ and a path from n_j to n_i is $\{n_j, n_{j-1}, ..., n_{i+1}, n_i\}$. Note that a possible assignment is $a = \{\langle t_i, t'_i \rangle\}$. For $k > j$ and $k < i$, the match $\langle t_k, t'_k \rangle$ is added to assignment a'. For $k = j$ to $i + 1$, since there is an edge from t_k to t_{k-1}, we can add $\langle t_k, t'_{k-1} \rangle$ to a'. Finally, we include our target match $\langle t_i, t'_j \rangle$ to a'. Hence, an assignment containing the target match is produced.

Only if part. Given an effective match $\langle t_i, t'_j \rangle$, there is an assignment a' that contains this match. Each match in a' is an edge in G, thus a' is a subset of E. We now show that we can find a path from n_i to n_j using the matches in a'. Consider a subgraph $G' = (V, a')$ that only contains matches in a'. Each node in G' has exactly one outgoing edge and one incoming edge. Hence, G' must be composed of cycles. The match $\langle t_i, t'_j \rangle$ is represented by the edge (n_i, n_j) which lies on a cycle as well. So, n_i is reachable from n_j by traveling through the cycle. □

For example in Figure 1, n_2 is not reachable from n_1. This means the match of $\langle t_2, t'_1 \rangle$ is not effective and cannot appear in a valid assignment. Thus, all ineffective matches can be identified and removed from the assignment graph by the adversary. This results in a reduced assignment graph. Figure 2 shows the graph derived from the initial one shown in Figure 1, containing only the effective matches. This gives a clearer picture why Table 4b does not satisfy 2-anonymity, as t_2 can only be mapped to t'_2 in a valid assignment.

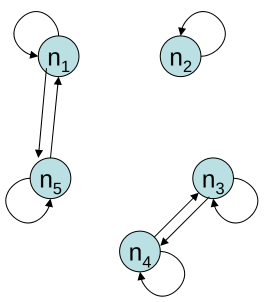

Figure 2: Effective matches in the graph of Table 4b

4.1.2 *Impact of effective match on generalization*

From Theorem 1, we know that the effectiveness of a match can be determined by looking at the connectivity of nodes in a graph. In fact, if we keep only effective matches, the graph will degenerate to the set of its strongly connected components.

THEOREM 2. *Consider a table T, its anonymized table T^*, and the corresponding assignment graph $G = (V, E)$. If all matches are effective, G is a set of strongly connected components, such that there are no edges between any two components.*

PROOF. A graph can always be decomposed to a number of strongly connected components. We prove the theorem by showing that each component in G is independent, i.e., there is no edge between any two components. We prove the statement by contradiction. Without loss of generality, we assume C_1 and C_2 are two components in G and there is a path from u to v where $u \in C_1$ and $v \in C_2$. Thus any node $x \in C_1$ can reach any node $y \in C_2$ via the path from u to v. Since there are effective matches only, there must be a path from v to u due to Theorem 1. Hence, from every node $y \in C_2$, we can reach any node $x \in C_1$. C_1 and C_2 are a single strongly connected component, which contradicts the assumption. □

Theorem 2 leads to an interesting observation: tuples are partitioned to strongly connected components in a non-homogeneous generalization. Note that the complexity of finding the strongly connected components is linear in the number of edges in G, due to Tarjan's algorithm [23].

4.2 Randomization in generalization

With non-homogeneous generalization, the generalized QID of tuples in a partition (i.e., equivalent class) may vary. For example, in Table 3, t'_1, t'_2 and t'_3 have a unique generalized QID. This offers additional information to an adversary in his quest for the identities of the anonymized tuples. If we use a deterministic non-homogeneous generalization approach, the generalized value of each tuple in the table would be the same for every possible run of such a method. Therefore if the adversary knows the generalization algorithm, he can apply it on H, compare the result with the anonymized table and infer the original QID of the anonymized tuples and therefore their identities. Due to this problem, randomization is necessary when anonymizing a table with non-homogeneous generalization.

A good randomization technique implies that when an adversary finds k tuples in H joined with an anonymized tuple t'_i in T^*, the probability of each of these k tuples being the real identity of t'_i is the same ($= \frac{1}{k}$). We can achieve this goal by first computing the generalized QID of tuples deterministically and then assigning each generalized QID to a tuple in T in a randomized way. Figure 3 is an example, illustrating this process for 2-anonymity. First, we generate for each original tuple t_i in T a generalized QID t'_i, which contains t_i and $k-1$ additional ones from T. The QID generation function, denoted by gen, takes as input a set of tuples and gives a generalized QID range. Next, for each generalized QID, we assign a random identity to it with a probability of $\frac{1}{k}$. In the example, we have picked $t_2 =_I t'_1$, $t_3 =_I t'_2$, and $t_1 =_I t'_3$. The other attributes are copied to the anonymized table accordingly.

Thus, the generalization procedure is divided into two steps: (i) generalized QID generation; (ii) random assignment generation. The QID generation determines whether k-anonymity can be achieved and affects the possible assignments that we can choose from in the randomization. For example, if the QID generation for Table 4a is done as shown in Table 4b, it is not possible to achieve

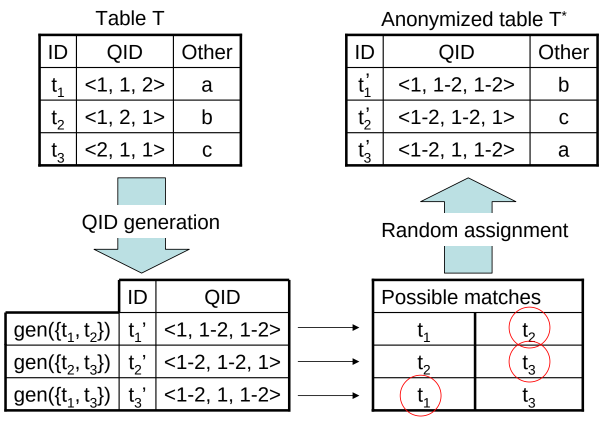

Table T

ID	QID	Other
t_1	<1, 1, 2>	a
t_2	<1, 2, 1>	b
t_3	<2, 1, 1>	c

	ID	QID
gen({t_1, t_2})	t_1'	<1, 1-2, 1-2>
gen({t_2, t_3})	t_2'	<1-2, 1-2, 1>
gen({t_1, t_3})	t_3'	<1-2, 1, 1-2>

Possible matches	
t_1	t_2
t_2	t_3
t_1	t_3

Anonymized table T*

ID	QID	Other
t'_1	<1, 1-2, 1-2>	b
t'_2	<1-2, 1-2, 1>	c
t'_3	<1-2, 1, 1-2>	a

Figure 3: The generalization process in achieving 2-anonymity

k-anonymity, as we have shown in Section 4.1.1. In the following subsection, we will discuss how we can guarantee k-anonymity in the QID generation process.

4.2.1 *A sufficient condition for k-anonymity*

In order to preserve k-anonymity, we have already shown that a necessary condition is to have at least k effective matches for each tuple. However, this condition alone cannot guarantee k-anonymity. Consider the assignment graph shown in Figure 4. For simplicity, self-loops are omitted and reciprocal edges connecting the same pair of nodes are merged to a single bidirectional edge. K_{3a} and K_{3b} are complete graphs of 3 nodes. Note that every edge in the graph represents an effective match and every node has at least three incoming and outgoing edges. However, 3-anonymity cannot be achieved by randomization on top of this assignment graph. For an edge $n_1 \to n_i$ where $i \neq 1$, the path from n_i to n_1 must go through edge $n_5 \to n_6$. So, if $t_1 \neq_I t'_1$, we know that $t_5 =_I t'_6$. From this, we can draw a conclusion that either $t_1 =_I t'_1$ or $t_5 =_I t'_6$ is true (a probability of $\frac{1}{2}$ to breach privacy).

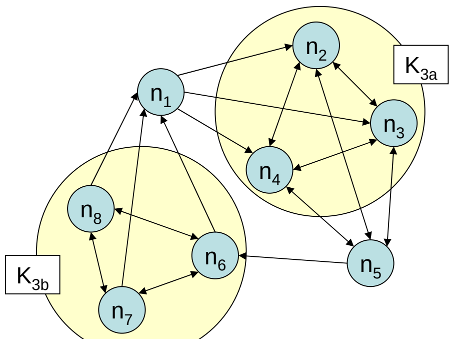

Figure 4: An example of assignment graph that has 3 effective matches for each tuple but violates 3-anonymity (self-loops are removed for simplicity)

In the above example, each of the possible assignments contains either the match $\langle t_1, t'_1 \rangle$ or $\langle t_5, t'_6 \rangle$. Thus, we can find at most 2 assignments with no overlapping matches. In fact, if there are $k = 3$ assignments with no overlapping matches, we can achieve k-anonymity. First, we define the concept of match-different condition.

DEFINITION 7. *(match-different assignments) Given two assignments a_i, a_j. a_i is match-different to a_j if $a_i \cap a_j = \emptyset$.*

Having k match-different assignments, we can randomly pick one of them as the resulting assignment of the randomization process. For each anonymized tuple t'_j, there are k different possible identities in total. Each identity of t'_j is in a different match-different assignment. Since each assignment has the same chance to be picked, t'_j is assigned to a particular tuple in T with the same $\frac{1}{k}$ chance; hence, k-anonymity can be achieved. To ensure that there are k match-different assignments in the set of generalized QID, we prove that it is sufficient that in the assignment graph with only effective matches each node has the same number ($\geq k$) of incoming edges and outgoing edges.

LEMMA 1. *Consider an assignment graph with only effective matches, where each node has k outgoing edges and k incoming edges. Given d match-different assignments a_1, a_2, ..., a_d, where $1 < d < k$, we can always find an assignment a_{d+1} such that a_{d+1} is match-different to a_i for $i = 1$ to d.*

PROOF. See Appendix A. □

5. ANONYMIZATION USING NON-HOMOGENEOUS GENERALIZATION

In this section, we discuss how we can generate a k-anonymized table using non-homogeneous generalization, building on the observations from the previous section. Although we can apply non-homogeneous generalization directly on T, due to the large scale of the data, the high cost of the necessary randomization, and the natural partitions that possibly exist in the data, we first partition the data into groups of k or more tuples and then apply non-homogeneous generalization to each group. In a nutshell we follow the following framework:

1. Divide the tuples into partitions
2. Generalize the QID of each tuple in each partition
3. Assign generalized QIDs to tuples, based on a random assignment

We first discuss how we generalize the QID of each tuple (step 2) in Section 5.1. Then, we explain our randomization technique (step 3) in Section 5.2. Finally, we outline our partitioning method (step 1) in Section 5.3.

5.1 Ring generalization

Assuming that the data are partitioned, non-homogeneous generalization should be applied to each partition. In fact, we need to determine for each tuple, which $k-1$ other tuples will be included in the generalization. Then, we can extend the QID of a tuple to include the QID of the other tuples. Let *gen* be a QID generalization function that takes as input a set of tuples and returns a generalized range on QID. For example, considering Table 4a, $gen(\{t_2, t_4, t_5\})$ = [2-5]. Let $PS(t_i)$ be the set of tuples in T that is used to produce a generalized QID for t_i. In the above example, $PS(t_2) = \{t_2, t_4, t_5\}$. Based on Lemma 1, we should have $|PS(t_i)| = |PS(t_j)|$ for all i, j. In order to minimize information loss in the generalized QIDs and satisfy k-anonymity, we design a generalization with $|PS(t_i)| = k$ for all t_i.

Consider the set of tuples in a partition P and assume that the tuples are ordered as $t_1, t_2, ..., t_{|P|}$. An easy way to construct $PS(t_i)$ is to assign the k consecutive tuples $gen(t_i, t_{i+1}, \ldots, t_{i+k-1})$ to t_i. (Note that if $i + j > |P|$, we use $i + j - |P|$ instead.) We call this *ring generalization*, as the assignment graph resulting from it looks like a ring. Figure 5 illustrates the ring generalization for a partition with 5 tuples and $k = 3$. The upper-left graph in Figure 6 is the corresponding assignment graph.

Tuple in T	$PS(t_i)$			Tuple in T^*
t_1	t_1	t_2	t_3	$t'_1 = gen(t_1, t_2, t_3)$
t_2	t_2	t_3	t_4	$t'_2 = gen(t_2, t_3, t_4)$
t_3	t_3	t_4	t_5	$t'_3 = gen(t_3, t_4, t_5)$
t_4	t_4	t_5	t_1	$t'_4 = gen(t_1, t_4, t_5)$
t_5	t_5	t_1	t_2	$t'_5 = gen(t_1, t_2, t_5)$

Figure 5: Ring generalization for a partition with 5 tuples and k = 3

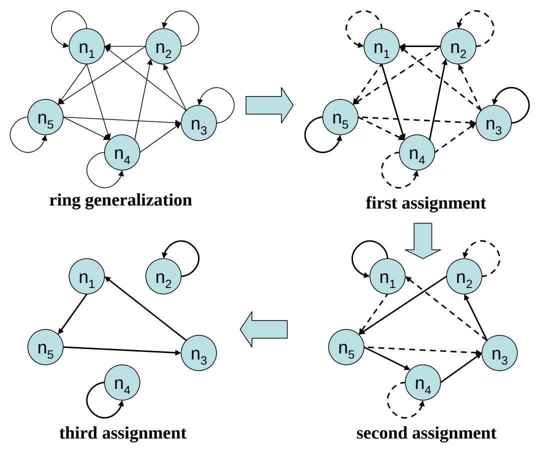

Figure 6: Generating three random assignments

Note that every match in the ring generalization is effective, as it is part of a cycle. In addition, since each node in the assignment graph represented by ring generalization has k incoming edges and k outgoing edges, we can find k match-different assignments and k-anonymity can be assured by randomly picking one of them as the actual assignment (Lemma 1). The ring generalization is a local optimal solution, because we cannot remove any more edges from the graph. In addition, we can easily show that it can give equal or better utility compared to any homogeneous generalization. If the partition size is k, the ring generalization degenerates to a homogeneous generalization, where all original tuples in the group match with all generalized tuples. If the partition size $|P|$ is greater than k, in the ring generalization, every tuple will match a set S of k tuples as opposed to $|P|$ in the homogeneous case. Since $S \subset P$, and all utility metrics are monotonic to subset relationships, only better utility can be achieved by the ring generalization.

LEMMA 2. *Ring generalization gives a k-anonymized partition of equal or better utility than that given by a homogeneous generalization on the same partition.*

An additional benefit of this generalization is that, given a proper ordering of the tuples (e.g., using Hilbert curves), the k values in each generalized QID will be close to each other with high probability, maximizing the utility gain compared to a homogeneous generalization.

5.2 Randomization

In this section, we will describe how we randomly assign each generalized QID to a tuple in T (and replace the original QID of the tuple by the generalized one). An intuitive idea is to generate all possible assignments and pick one uniformly at random. Unfortunately, such an approach may violate k-anonymity. For example, in a partition with 5 tuples, the generalized QIDs using ring generalization for 3-anonymity will have 13 different possible assignments. Note that 13 is not divisible by 3, meaning that some matches are contained in more assignments than other matches. This allows the adversary to infer these matches with probability higher than $1/k$. In the example of Figure 5, matches $\langle t_i, t'_{i-1}\rangle$ have higher probability $(5/13)$ than matches $\langle t_i, t'_i\rangle$ and $\langle t_i, t'_{i-2}\rangle$ (with probability $4/13$).

As discussed in Section 4.2.1, the solution is to define k match-different assignments, and randomly select one of them. One easy construction of k match-different assignments is to set $a_i = \{\langle t_i, t'_{i-j}\rangle\}$ for $j = 0$ to $k-1$. (Note that if $i-j < 1$, we use $i-j+|P|$ instead.) Consider the example in Figure 5, and assume that we pick each column of $PS(t_i)$ as an assignment. By setting one of these assignment as the real assignment randomly, there is a chance of $\frac{1}{k}$ a column is chosen. Thus the probability $Pr(t_i =_I t'_{i-j})$ is $\frac{1}{k}$ and k-anonymity is preserved.

However, if we apply this approach, privacy can easily be compromised when the adversary knows the identity of one anonymized tuple as background knowledge. In practice, such background knowledge can easily be acquired. For example, using the generalization of Figure 5, if an adversary knows that $t_3 =_I t'_2$, he knows that the second column is the real assignment. Hence, he can find out the identities of all anonymized tuples, e.g., $t_2 =_I t'_1$. This type of attack is called *corruption* and has been studied in [22].

In order to increase the resistance to corruption, the k match-different assignments are defined randomly. The generation process shares a similar framework as the proof of Lemma 1. The pseudocode of an algorithm that generates a random assignment is shown in Figure 7. In the followings, we briefly describe the basic idea of the algorithm. The algorithm is run k times to generate the k assignments. At each run, it operates on the set of matches M that are not present in assignments generated in previous runs. It tries to find a set of cycles in the graph that cover all nodes by random walks and use them to define an assignment. The cycles are found incrementally, starting from an unassigned node. After a cycle has been found, all its nodes are marked as "processed" and searching for a new cycle starts until all nodes are processed, in which case the assignment is committed and returned. Cycles are not directly committed in the assignment once found, because some of them, when removed, may result in a graph where there does not exist any cycle. Thus, while finding a new cycle, matches set by previous cycles may change. In addition, we limit each node to be visited at most once in each random walk (by remembering the nodes traveled in U). The algorithm backtracks when it reaches a dead end. Lemma 1 guarantees correctness and termination.

Figure 6 exemplifies three runs of the algorithm on the ring generalization of Figure 5 shown at the upper left of Figure 6. The solid edges show the matches that are chosen for the current assignment. For example, the first assignment, containing cycles $n_1 \rightarrow n_4 \rightarrow n_2 \rightarrow n_1$, $n_3 \rightarrow n_3$, and $n_5 \rightarrow n_5$, will assign t_1 to t'_4, t_4 to t'_2, t_2 to t'_1, t_3 to t'_3, and t_5 to t'_5. After the first run, the corresponding edges are removed and the algorithm is run again to generate the second assignment.

Regarding corruption, ring generalization gives a $(k-1)$-vertex-connected assignment graph, i.e., the graph is still connected after any $k-1$ nodes in the graph are removed. Assume an adversary obtains background knowledge about the identity of a set of anonymized tuples Q, belonging to the same partition. The adversary can remove the corresponding nodes from the assignment graph. If $|Q| < k-1$, the assignment graph is still connected, i.e., all matches are still effective. There are at least $k-|Q|$ outgoing edges and $k-|Q|$ incoming edges for each remaining node. So, there are at least $k-|Q|$ possible identities for each anonymized tuple. Due to randomization, an adversary cannot find out the actual identity of an anonymized tuple. Therefore, non-homogeneous generalization with randomization offers a similar privacy protection to k-anonymity corruption as homogeneous generalization.

Input: A partition of tuples P; A set of anonymized tuples Q; A set of possible matches M (excluding matches already used in other assignments).
Output: An assignment $a \subseteq M$.

```
1.  $a = \{\langle t_i, t'_i\rangle\}$ // initial assignment (possibly invalid)
2.  $L = P$ // $L$ represents the set of unprocessed nodes
3.  While ($L \neq \emptyset$)
4.    // pick an edge to start a loop
5.    Pick $t_i \in L$ at random
6.    Pick $t'_j \in Q$ randomly such that $\langle t_i, t'_j\rangle \in M$
7.    $U = \{t'_j\}$ // $U$ remembers the nodes traveled
8.    While ($t'_i \notin U$) // find a cycle by random walk
9.      // $t_i$ is assigned to $t'_j$, so the one that is assigned
10.     // to $t'_j$ before has to find another pair
11.     Select $t_x$ where $\langle t_x, t'_j\rangle \in a$
12.     Pick $t'_y \notin U$ randomly where $\langle t_x, t'_y\rangle \in M$
13.     if such $t'_y$ does not exist
14.       $t'_j = t'_j$'s parent // backtracking
15.     else
16.       Add $t'_y$ to $U$ and set $t'_j$ as $t'_y$'s parent
17.   End while
18.   // a loop is found
19.   update $a$ and remove nodes in the loop from $L$
20. End while
21. return $a$
```

Figure 7: Algorithm for generating a random assignment

5.2.1 Cost analysis and optimizations

The cost in randomization for a partition P is dominated by generating the k match-different assignments. When generating a new assignment, we maintain a list of unprocessed nodes L (line 2). Then, we find a cycle in the assignment graph which starts with a node in L by a depth-first random walk. Note that each node can be visited at most once. The complexity for that is $O(|V| + |E|)$ where $|V|$ is number of nodes and $|E|$ is the number of edges in the graph. $|V| = |P|$ and $|E| = k|P|$ in the assignment graph. Hence, it takes $O(k|P|)$ to generate a random cycle. Note that each new cycle contains at least one node in L. In the worst case, we need $|P|$ iterations to assign every node in L. So, the overall cost to generate a match-different assignment is $O(k|P|^2)$.

Since $|P| \geq k$, randomization becomes expensive for large values of k, or when the partitions are very large compared to k. Therefore a good partitioning strategy should avoid generating huge groups. We now describe two simple optimizations to reduce this cost in practice. In Section 7, we experimentally evaluate the cost of the optimized randomization algorithm and show that it is bearable in practical cases, as it only depends on the size of the partitions and not the database size.

Reducing the number of generated match-different assignments. In the randomization process, we first generate k match-different assignments, and then choose one of them randomly. Let a_1, a_2, ... a_k be the generated assignments in order. Since which assignment will be picked is independent of the generation process, we can first determine which a_i of the k assignments will be picked to be the real assignment and then generate up to the i-th assignment. This will, on average, reduce half of the randomization cost. Note that the assignments before the i-th should be generated, as they determine which edges remain at the time of the generation of the i-th assignment. Generating and picking always the first assignment would result in the selection of some matches with higher probability and is not acceptable, as discussed in the beginning of Section 5.2.

Using a random permutation to generate initial assignment. The goal of the algorithm in Figure 7 is to generate a random match-different assignment. A fast Monte Carlo way is to use a random permutation. The resulting permutation may not be a valid assignment because some of the matches may not be in the set of possible matches M. However, some matches in the permutation may be valid. We use this as the initial assignment to the algorithm (line 1 in Figure 7). L is initialized to be the set of anonymized tuples that do not have a valid match. This reduces the initial size of L and hence the computational cost of assignment generation.

5.3 Partitioning

In this section, we discuss how to choose a good partitioning strategy for non-homogeneous generalization. Before this, we will provide an appropriate measure for utility, which we adopt from previous work. Based on our discussion so far, value ranges are used to define a generalized QID of tuples. For example, for a QID which is generated by the three values $15, 20, 48$, we use range [15-48]. This format is compact and is easy to use in data analysis, however, it introduces some unnecessary information loss, as values within the range but not present in the generating set of values are included in it. For example, value 17 is implicitly included in the generalized range [15-48]. In fact, the QID generalization model that has the minimum information loss is the set representation, e.g., set $\{15, 20, 48\}$ is used as the generalized QID. The set representation offers a significant improvement in utility and it is also more general, as it is appropriate for both ordered and nominal attributes. Therefore, we adopt it in this paper and use it in subsequent discussions. In addition, we use the Global Certainty Penalty (GCP) [6] as a measure for utility.

DEFINITION 8. *(Information loss metric - GCP) Let t'_i be an anonymized tuple in anonymized table T^* using set representation. Let A be a QID attribute, $|A|$ be the cardinality of A, and $count_A(t_i)$ be the number of distinct values of A in t'_i. The normalized certainty penalty NCP of t'_i on attribute A is $NCP_A(t'_i) = \frac{count_A(t_i)-1}{|A|-1}$. $NCP(T^*) = \sum_{A \in QID} \sum_{t'_i \in T^*} NCP_A(t'_i)$, for the whole table T^*. Finally, $GCP(T^*)$ is defined as $\frac{NCP(T^*)}{d|T^*|}$ where d is the number of attributes in the QID.*

As discussed, similar to the homogeneous case, for scalability reasons we should divide the tuples of T into partitions before applying non-homogeneous generalization to each of them. One option is to use an off-the-shelf partitioning method for homogeneous generalization (e.g., [14]) and then apply our ring generalization at each partition. However, existing partitioning strategies may not be the most appropriate as they do not take into account the use of non-homogeneous generalization. The main difference between homogeneous and non-homogeneous generalization is that, in the former, it is always better to divide a large group into two. For example, consider a set of four QID $t_1 = \langle 1, 1, 1 \rangle$, $t_2 = \langle 1, 1, 2 \rangle$, $t_3 = \langle 2, 1, 2 \rangle$, $t_4 = \langle 1, 2, 1 \rangle$ and assume the domains of all QID attributes are the same. Suppose that we put partition t_1, t_2 into group G_1 and t_3, t_4 into another group G_2. t_1 and t_2 differ in one value, whereas t_3 and t_4 differ in 3 values. Thus, with this grouping the value of NCP is $\frac{8}{|A|-1}$. If we apply ring generalization directly on the four tuples, without partitioning, the value of NCP is $\frac{1+1+3+1}{|A|-1} = \frac{6}{|A|-1}$. Hence, we obtain a higher information loss after we partition the tuples. In the non-homogeneous case, even if the partition size is much larger than k, each generalized QID is generated from exactly k tuples. Thus, low information loss can be still achieved in large partitions, as opposed to homogeneous generalization, which suffers from high information loss if the size of partitions is large. On the other hand, the size of the partitions affects the cost of randomization in our approach (see Section 5.2.1), therefore it should be controlled.

5.3.1 Partitioning based on lexicographical order

We now discuss our partitioning strategy for non-homogeneous generalization. As discussed, we consider a set representation for the QIDs, i.e., in an anonymized tuple t'_i each QID attribute takes the set of values of that attribute in the k tuples that generate the QID. Hence, the distinct count $count_A(t'_i)$ of A's values in t'_i is less than or equal to k. According to the NCP measure, if the domain size of A is small, we lose more information if more than one values of A exist in a generalized tuple. For example, consider attributes 'sex' and 'birthday' with domain sizes 2 and 366, respectively. If we put two tuples in a group with different 'sex' values the NCP for that attribute in the group will be maximized, but if we put two tuples with different birthday, the introduced NCP error is small. Thus, during partitioning, we should prioritize the reduction of $count_A(t_i)$ for attributes with small domains.

To achieve this goal, we order the attributes according to their domain size; then the tuples in T are sorted in lexicographical QID order, based on the attribute ordering. The tuples are partitioned in a top-down fashion. First, we consider attribute A_1 with the smallest domain size and put tuples with the same A-value in the same partition. This results in a set of partitions P_1, P_2, ... P_m, such that the NCP of A_1 will be 0 in all partitions. However, some partitions may have less than k tuples. For each such partition P_j, we find a neighboring partition P_x, either P_{j-1} or P_{j+1}, that can either be merged with P_j or some tuples can be moved from P_x to P_j in order for both of them to have at least k tuples. If $|P_x| + |P_j| < 2k$, we merge P_x with P_j; otherwise, we move tuples from P_x to P_j such that $|P_j| = k$. After we are done with A_1, we recursively partition the resulting groups using the next attribute in order (i.e., A_2). In some partitions, the tuples may have different A_1 values (due to merging). For such partitions, we do not attempt to further decompose them recursively using another attribute. The partitioning strategy is repeated until all partitions are finalized or there are no more attributes that can be used for recursive partitioning. Figure 8 shows a pseudocode of the partitioning algorithm. On each finalized partition, we apply non-homogeneous generalization, as explained in Sections 5.1 and 5.2.

5.3.2 Cost analysis of partitioning

Before we apply the partitioning algorithm, shown in Figure 8, we need to sort the attributes and the tuples. Assuming that the number of attributes in the QID is negligible compared to the number of tuples, sorting costs $O(|T|log(|T|))$. Moving data between partitions or merging, is applied only for consecutive partitions. Each partition defined by the first attribute can recursively be repartitioned up to d times, assuming a d-dimensional QID. As the data having the same values in attributes $A_1, A_2, \ldots A_i$ are sorted

Input: a set of tuples P; parameter in k-anonymity k; Attribute used for partitioning A_x;
Precondition: (i) attributes in QID are sorted in ascending order of domain size $A_1, A_2, ..., A_{|QID|}$; (ii) tuples are sorted in lexicographical order according to attribute-order; (iii) all tuples in P have the same value on A_1, A_2, ... A_{x-1}.

```
 1. // minimize the uncertainty for attribute $A_i$
 2. Partition $P$ into $P_1$, $P_2$, ... $P_m$ using $A_i$
 3. // if there are not enough tuples in a group
 4. For each $P_j$ where $|P_j| < k$
 5.    Find $P_x$ as a neighboring partition of $P_j$
 6.    If $|P_j| + |P_x| > 2k$
 7.       move $k - |P_j|$ tuples from $P_x$ to $P_j$
 8.    Else
 9.       Merge $P_j$ and $P_x$
10. End for
11. For each $P_j$
12.    If $(\exists t_a, t_b \in P_j, t_a[A_i] \neq t_b[A_i])$ or $(A_i = A_{|QID|})$
13.       Apply non-homogeneous generalization on $P_j$
14.    Else
15.       partition($P_j$, $k$, $A_{i+1}$) // recursive call
16. End for
```

Figure 8: Partitioning Algorithm

w.r.t. attribute A_{i+1}, no additional sorting is required. In the worst case, where all tuples have the same values in all attributes the table will be read d times, so the worst-case cost is $|T|(d + log|T|)$.

6. EXTENSION TO L-DIVERSITY

In this section, we discuss how we can apply non-homogeneous generalization for other privacy principles. In particular, we focus on l-diversity, described in Definition 9.

DEFINITION 9. *(l-diversity) Let T be a table with a sensitive attribute S, and H be the projection of T on key and QID attributes. l-diversity is preserved by an anonymized table T^* if $\forall t_i \in H, \forall s \in S, Pr(t_i[S] = s) \leq \frac{1}{l}$ in a linking attack by joining H and T^*.*

By definition, in order to satisfy l-diversity each partition P should not contain a sensitive value that occurs more than $\frac{|P|}{l}$ times. In such a partition, we can order the tuples in P such that no consecutive l tuples have a sensitive value that occurs more than once. For example, consider a partition of size 4, where there are 2 sensitive values s_1 and s_2, each appearing 2 times in the partitions. In order to achieve 2-diversity, we can arrange the tuples with sensitive values in the order of $\{s_1, s_2, s_1, s_2\}$. By applying ring generalization to the ordered tuples, each generalized QID will cover two tuples with different sensitive values. Hence, 2-diversity is satisfied. Thus, non-homogeneous generalization can directly be applied on the partitions generated by any existing algorithm for l-diversity, like [6, 26]. The utility will be higher than or equal to that of applying homogeneous generalization. However, as existing algorithms do not take into account non-homogeneous generalization, they tend to generate partitions with minimal sizes, so the utility improvement by non-homogeneous generalization may not be maximal.

We now discuss how our partitioning strategy (described in Section 5.3) can be adapted to generate l-diverse partitions. Note that, the original table should satisfy l-diversity, otherwise it cannot be split to partitions which all satisfy l-diversity [26]. Recall that our partitioning strategy recursively divides the tuples using one attribute at a time. In each iteration, we obtain a set of partitions P_1, P_2, ..., P_m. For a partition P_j that does not satisfy l-diversity, we find a partition P_x such that l-diversity is satisfied by the merged partition $P_j \cup P_x$. If such a partition cannot be found, we merge P_j with a random partition and merging continues until the resulting partition satisfies l-diversity. In the worst case, the algorithm will merge all partitions into a single one, which must satisfy l-diversity. Hence, this scheme always gives a set of partitions that satisfy l-diversity. Then, for each partition, we order the tuples and apply non-homogeneous generalization.

As we have discussed in Section 5.2.1, the cost of randomization is highly correlated to the partition size. So, if a huge partition is generated by the above process, the generalization cost for it may be extremely high. In order to reduce the cost, we perform arbitrary splits to such partitions, making sure that each sensitive value appears in each smaller partition at most once. This way, each partition has a maximum size equal to the domain of the sensitive attribute, which is usually small in practice.

7. EMPIRICAL EVALUATION

In this section, we experimentally compare our developed anonymization scheme, denoted by NH (non-homogeneous generalization), with the state-of-the-art k-anonymity algorithm of [6]. The algorithm of [6] sorts the the tuples based on their values on a Hilbert curve and then generates partitions using a dynamic programming algorithm that is optimal for homogeneous generalization of 1-dimensional QID. Although the resulting partitions may not be optimal, they have low information loss, since two nearby values on the Hilbert curve are also near in the original high dimensional space with high probability. Range representation is used for generalized QIDs in [6], however, a set representation can directly be applied on the partitions to improve utility. We use HR to denote the original algorithm of [6] with range representation and HS its version with set representation. Our NH algorithm applies non-homogeneous generalization on top of the partitioning scheme proposed in Section 5.3 and uses set representation for the generalized QID. All optimizations for randomization as described in Section 5.2.1 are implemented in NH. In addition, we implemented a method that uses our partitioning strategy followed by homogeneous generalization (using set representation), to verify whether any improvement in the information loss is achieved due to our partitioning strategy or due to non-homogeneous generalization. We denote this method by HP (homogeneous partitioning). HP, after partitioning, uses a single generalized QID to represent all data in each partition. Randomization is not applied by this algorithm, since generalization is homogeneous. All algorithms are implemented in C++ and the experiments are run on an Intel Core 2 Duo 2.8GHz machine with 2GB RAM, running Windows.

Our experiments are done mainly on a real dataset CENSUS (downloadable from 'http://www.ipums.org') that is widely used in the literature (e.g., in [27, 26, 6]). The dataset contains information about 500K individuals. A summary of the attributes in the dataset is shown in Table 5. Note that the majority of attributes are nominal, indicating that a set representation for a generalized QID is more appropriate than a range representation, since there is no natural order for most of the attributes. In the experiments, we vary the following parameters: (i) number of tuples n: we sample the CENSUS dataset to generate input tables of varying size; (ii) number of attributes d in the QID: we use the first d attributes as the QID while other attributes are treated as 'others'; (iii) value of k in k-anonymity. The range of each parameter and the default value is shown in Table 6.

We measure the information loss of the generalized tables using GCP (defined in Section 5.3), which is a commonly used metric

Attribute	Cardinality
Age	79
Gender	2
Education Level	17
Marital Status	6
Race	9
Work Class	10
Country	83
Occupation	50

Table 5: Attributes in CENSUS dataset

Parameter	Values
Size of table n	**100K**, 200K, 300K, 400K, 500K
QID dimensionality d	2, **3**, 4, 5, 6, 7
Privacy parameter k	2, 4, 6, 8, **10**, 20, 40, 60, 80, 100

Table 6: Parameters used in experiments on CENSUS. Default values in bold font.

[6, 28]. In addition, we measure the computational cost of the algorithms. We also use another widely used dataset (ADULT) from the UCI repository [2]. The dataset contains information of 32561 individuals and has 15 attributes. The entire dataset is used in each experiment and we vary d and k.

7.1 Varying the dataset size

The information loss and the computational cost of each algorithm, as a function of the table size, are shown in Figure 9. As HR and HS differ only in the final representation of the QID, there is no difference in the execution time. So, we only compare HS with NH and HP when evaluating computational cost.

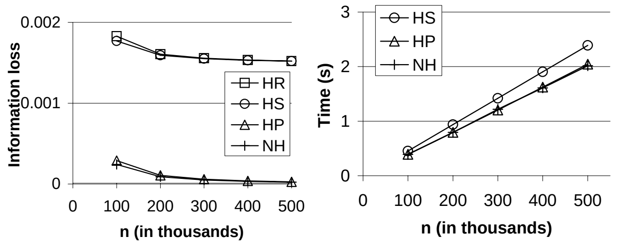

Figure 9: Information loss and execution time; varying n on CENSUS dataset; $k = 10$, $d = 3$

The information loss of all methods is low in all cases and further decreases with n. This is because when the database size increases, more tuples are sharing the same or very similar QID. As a result, it is easier for the partitioning strategies to find very similar tuples and generalize them. We observe a huge difference between HR/HS and NH/HP. Since the set representation is always better than range generalization, a drop of information loss from HR to HS is expected. However, the improvement of HS over HR is only marginal, indicating that the partitioning strategy of [6] may not be the most appropriate for set generalization. On the other hand, our partitioning scheme is more suitable for set representation and non-homogeneous generalization, as indicated by the plot. There is only a slight difference between HP and NH. This happens because the partitions for this dataset happen to have size close to k, or the tuples in the partition have very few differences in the QID. Hence, the difference between homogeneous and non-homogeneous generalization is not significant on our partitioning scheme in this setting. However, we remark that this case is not general; in subsequent experiments we will see cases where HP does not behave well. The execution cost of the algorithms is linear to the number of tuples and NH/HP have slightly lower cost than HS. All methods scale well with the database size.

7.2 Varying QID dimensionality

Figure 10 shows the information loss and the computational cost of each algorithm, for different QID dimensionality values. The information loss of the algorithms increases in general with d (except for HR and HS which have the minimum information loss at $d = 3$). This is due to the increase of data sparsity, as the QID domain is multiplied by the domains of additional attributes. NH/HP sustains a big improvement over HS/HR and the gap between HS and NH is increasing with d. HS/HR and NH follow similar trends on the ADULT dataset (see Figure 11). However, HP does not have stable performance. We observe a sudden increase in the GCP of HP for $d = 5$. This happens because there is a sudden increase in the average partition size for $d = 5$. The GCP of HP is even higher than that of HR and HS. As our partitioning does not aim at minimizing the average partition size, there could be cases of large partitions that result in a high information loss when homogeneous generalization is applied on them. On the other hand, NH consistently achieves a low information loss as it applies non-homogeneous generalization on these large partitions.

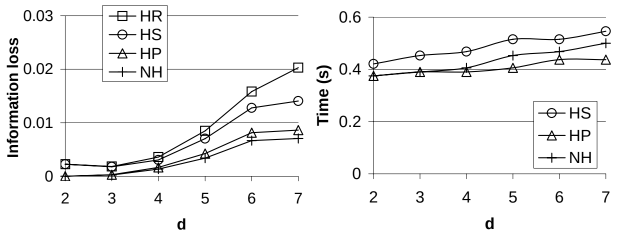

Figure 10: Information loss and execution time; varying d on CENSUS dataset; $n = 100$K, $k = 10$

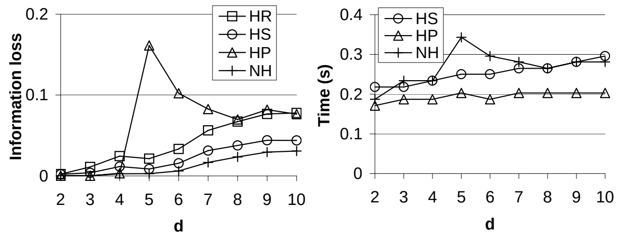

Figure 11: Information loss and execution time; varying d on ADULT dataset; $k = 10$

The execution time of the algorithms is only slightly affected by dimensionality and NH has a similar cost to HS. This result is expected, as the main cost factor is the size of the database n, due to sorting; all methods are capable of handling well data with high-dimensional QIDs. We only observe a spike in the cost of NH for $d = 5$ in the ADULT dataset, which is due to the large partitions generated in this setting. Large partitions, increase the cost of randomization, as discussed in Section 5.2.1.

7.3 Varying privacy level

As Figure 12 and Figure 13 show, the GCP of all algorithms grows linearly with k. This is reasonable, as the generalized QIDs should include more tuples. NH has the lowest GCP among the

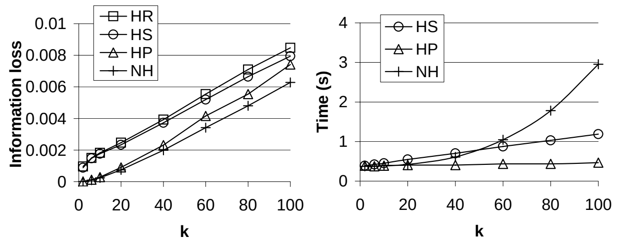

Figure 12: Information loss and execution time; varying k on CENSUS dataset; $n = 100$K, $d = 3$

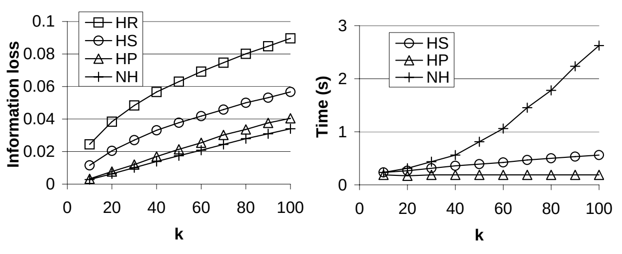

Figure 13: Information loss and execution time; varying k on ADULT dataset; $d = 4$

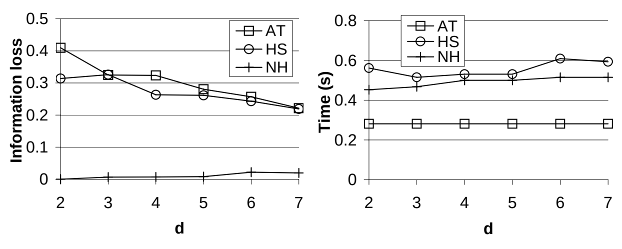

Figure 14: Information loss and execution time; varying d on CENSUS dataset; $n = 100$K, $d = 3$

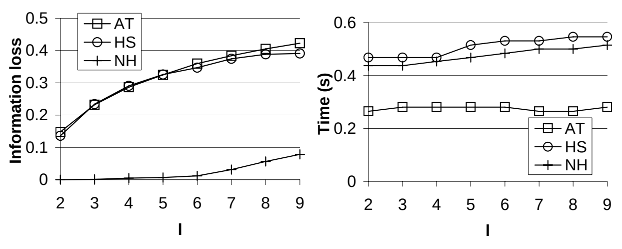

Figure 15: Information loss and execution time; varying l on CENSUS dataset; $n = 100$K, $l = 5$

four and the absolute improvement over HS/HR is steady. On the other hand, the computational cost of NH has a super-linear increase with k. This is due to cost of randomization; recall that the cost of generating a random assignment is $O(k|P|^2)$ (see Section 5.2.1). Since $|P| \geq k$, the worst case complexity is in the order of k^3. The cost of HP reflects the cost of partitioning in NH; therefore for $k \geq 40$ the cost of randomization becomes the dominant factor in NH. However, even at $k = 100$, the execution time of the entire anaonymization process is 2.95s, which is reasonable. In typical applications, we do not expect values of k much greater than 100, so NH is expected to be applicable in practice, especially since its execution cost scales well with the database size (see Figure 9).

7.4 l-diversity

We now evaluate the extension of our methodology for l-diversity, described in Section 6. As before, we use NH to denote our algorithm. We implemented two algorithms as competitors: (i) l-diversification with homogeneous generalization based on Hilbert curve transformation [6], denoted by HS; (ii) anatomy [26], denoted by AT. A set representation is used for the generalized QIDs in all cases, as it always has lower GCP than a range representation. In HS, the tuples are first ordered using a Hilbert curve mapping and then a greedy partitioning algorithm is used to group together l or more tuples with different sensitive values. AT randomly picks from the dataset a group of l tuples with different sensitive values such that the remaining mircodata still satisfy l-diversity. This process is repeated until less than l tuples remain. Each remaining tuple t is assigned to a random group, such that no other tuple in the group has the same sensitive value as t.

For our experiments, we used a random sample of CENSUS with 100K tuples. We assume the last attribute (Occupation) is the sensitive one. We vary d from 2 to 7 and set $l = 5$ and then vary l from 2 to 9 and set $d = 3$. Figures 14 and 15 show the results.

NH outperforms both HS and AT by a large factor in terms of GCP. As AT groups tuples at random, it is expected that tuples with very different QID are grouped together, which results in a high information loss. However, HS shows a similar performance as AT in general, meaning that the partitioning used in HS fails to group tuples having the same values in QID attributes. The primary focus of the partitioning in HS is to form groups with size close to l and different sensitive attributes. Thus, the selection of groups (which should be as small as possible) is not primarily based on the common QID attribute values. On the other hand, our partitioning strategy brings together tuples for which a large subset of QID attributes have common values without placing a hard constraint on the group sizes. The average partition size is high in NH (e.g., at 11.2 at $l = 5$, $d = 2$), but the partitions are formed based on QID similarity. With the help of non-homogeneous generalization, we can achieve low information loss as each generalized QID is derived from l tuples only, irrespectively to the size of the group. Summing up, our non-homogeneous generalization strategy on larger partitions can achieve much higher utility compared to homogeneous generalization on small groups. In terms of computational cost, all algorithms are very efficient and AT is the fastest method in all cases, due to its simplicity.

8. CONCLUSIONS

In this paper, we developed a novel anonymization framework based on non-homogeneous generalization. We have shown that it is not necessary to uniformly give all tuples the same QID in a partition, in order to achieve k-anonymity. For partitions of size greater than k, non-homogeneous generalization can always lead to better utility. Applying non-homogeneous generalization is not trivial, as the adversary can identify and remove invalid matches between a public table and the anonymized one, or infer that some matches have higher probability than others. We identify these problems and propose solution based on deterministic a ring genereralization scheme and a randomization process, which ensures that the result has the same security level as homogeneously generalized partitions. In addition, we proposed a partitioning algorithm, which is tailored for non-homogeneous generalization. Our experimental results show that our methodology can greatly reduce the information loss compared to the state-of-the-art, while its execution cost is only sensitive to high values of k, therefore it is scalable for small to medium values of k. We also implemented and tested an adaptation of our technique for l-diversity and experimentally showed that it may achieve even higher improvement in terms of information loss, compared to k-anonymity. In the future, we plan to investigate additional partitioning schemes targeting the further reduction of the information loss and alternative randomization approaches of reduced cost (e.g., based on non-greedy assignment generation), in order to improve scalability with k.

9. REFERENCES

[1] R. Agrawal and R. Srikant. Privacy-preserving data mining. In *SIGMOD*, 2000.

[2] A. Asuncion and D. Newman. UCI Machine Learning Repository, 2007.

[3] M. Barbaro and T. Zeller. A Face Is Exposed for AOL Searcher No. 4417749. The New York Times, 2006. http://www.nytimes.com/2006/08/09/technology/09aol.html.

[4] R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, 2005.

[5] A. Evfimievski, R. Srikant, R. Agrawal, and J. Gehrke. Privacy preserving mining of association rules. In *KDD*, 2002.

[6] G. Ghinita, P. Karras, P. Kalnis, and N. Mamoulis. Fast data anonymization with low information loss. In *VLDB*, 2007.

[7] A. Gionis, A. Mazza, and T. Tassa. k-anonymization revisited. In *ICDE*, 2008.

[8] P. Golle. Revisiting the uniqueness of simple demographics in the US population. In *WPES*, 2006.

[9] Z. Huang, W. Du, and B. Chen. Deriving private information from randomized data. In *SIGMOD*, 2005.

[10] T. Iwuchukwu and J. F. Naughton. k-anonymization as spatial indexing: toward scalable and incremental anonymization. In *VLDB*, 2007.

[11] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing location-based identity inference in anonymous spatial queries. *IEEE Trans. Knowl. Data Eng.*, 19(12), 2007.

[12] D. Kifer. Attacks on privacy and definetti's theorem. In *SIGMOD*, 2009.

[13] K. Lefevre, D. J. Dewitt, and R. Ramakrishnan. Incognito: efficient full-domain k-anonymity. In *SIGMOD*, 2005.

[14] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *ICDE*, 2006.

[15] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, 2007.

[16] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, 2006.

[17] D. J. Martin, D. Kifer, A. Machanavajjhala, J. Gehrke, and J. Y. Halpern. Worst-case background knowledge in privacy. In *ICDE*, 2007.

[18] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *PODS*, 2004.

[19] M. F. Mokbel, C. Y. Chow, and W. G. Aref. The new casper: Query processing for location services without compromising privacy. In *VLDB*, 2006.

[20] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. Knowl. Data Eng.*, 13(6):1010–1027, 2001.

[21] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5), 2002.

[22] Y. Tao, X. Xiao, J. Li, and D. Zhang. On anti-corruption privacy preserving publication. In *ICDE*, 2008.

[23] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[24] R. C.-W. Wong, A. W.-C. Fu, K. Wang, and J. Pei. Minimality attack in privacy preserving data publishing. In *VLDB*, 2007.

[25] R. C.-W. Wong, J. Li, A. W.-C. Fu, and K. Wang. (α, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing. In *KDD*, 2006.

[26] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *VLDB*, 2006.

[27] X. Xiao and Y. Tao. m-invariance: Towards privacy preserving re-publication of dynamic datasets. In *SIGMOD*, 2007.

[28] J. Xu, W. Wang, J. Pei, X. Wang, B. Shi, and A. W.-C. Fu. Utility-based anonymization using local recoding. In *SIGKDD*, 2006.

APPENDIX

A. PROOF OF LEMMA 1

All possible matches are visualized as edges in the assignment graph. Due to the match-different requirement, a_{d+1} cannot contain any match in a_i for $i = 1$ to d. We remove the corresponding edges from the graph. Each assignment contains $|T|$ matches, each defined by a different tuple in the original table and a different tuple in the anonymized table. So, for each assignment we remove $|T|$ edges: one incoming edge and one outgoing edge for each node. In the resulting graph G', there will be $k - d$ incoming edges and $k - d$ outgoing edges for each node. First, we show that each edge is part of a loop in G'.

LEMMA 3. *Given a directed graph such that each node has x incoming edges and x outgoing edges, where $x \geq 1$. If there is an edge $n_i \rightarrow n_j$, then there is a path from n_j to n_i.*

PROOF. We prove the lemma by contradiction. Assume that there is an edge $(n_i \rightarrow n_j)$ such that we cannot reach n_i from n_j. We classify the nodes in two groups: (i) U: the set of nodes that can reach n_i; W: the set of nodes that can be reached from n_j. Since there is no path from n_j to n_i, $U \cap W = \emptyset$. Considering only nodes in W, there are $|W|x$ incoming edges and $|W|x$ outgoing edges. Note that for each node in W, all outgoing edges should point to a node in W (otherwise n_i can be reached from n_j). On the other hand, there is an incoming edge to W $(n_i \rightarrow n_j)$, which is from outside W. Hence, there are at most $|W|x - 1$ incoming edges to nodes in W from nodes in W, but there are $|W_j|x$ outgoing edges from nodes in W to nodes in W. This leads to contradiction by pigeonhole principle. □

Now we show how we can construct a_{d+1}. The objective is to find a set of cycles that cover all nodes in G'. First we initialize $a_{d+1} = \{\langle t_i, t'_i \rangle\}$. This assignment is not necessarily match-different, as some $n_i \rightarrow n_i$ edges may have been taken by previous assignments (i.e., may not be present in G'). We need to pick another match for these tuples; we mark all these tuples as "unassigned". We pick an unassigned tuple and one of its outgoing edges at random and find a path back to the starting node. Assume the loop contains l nodes: n_{y_1}, n_{y_2}, ..., n_{y_l}. An edge in the loop $n_{y_i} \rightarrow n_{y_{i+1}}$ represents the match $\langle t_{y_i}, t_{y_{i+1}} \rangle$. We use the matches to replace the corresponding matches in a_{d+1}, i.e., $\langle t_{y_i}, t_{y_i} \rangle$ in a_{d+1} is replaced by $\langle t_{y_i}, t_{y'_{i+1}} \rangle$. This gives us an updated assignment a_{d+1}. Note that by updating the assignment, the represented tuples by a node in the assignment graph are updated as well; thus, cycles in later iterations are computed on the updated assignment graph. The l matches satisfy the match-different requirement since they are defined by edges in G'. While there exist more unassigned tuples, we repeat this process by selecting another unassigned tuple t_y and one of its outgoing edges at random, finding a loop, and updating a_{d+1}. The new loop ensures that the updated a_{d+1} will result in at least one new matched node, so the process is guaranteed to terminate with an assignment a_{d+1}, where all edges are present in G', i.e., a match-different assignment.