k-Jump Strategy for Preserving Privacy in Micro-Data Disclosure

Wen Ming Liu and Lingyu Wang
Concordia Institute for Information Systems Engineering
Concordia University
Montreal, QC H3G 1M8, Canada
{l_wenmin,wang}@ciise.concordia.ca

Lei Zhang
Center for Secure Information Systems
George Mason University
Fairfax, VA 22030-4444, USA
lzhang8@gmu.edu

ABSTRACT

In disclosing micro-data with sensitive attributes, the goal is usually two fold. First, the data utility of disclosed data should be maximized for analysis purposes. Second, the private information contained in such data must be limited to an acceptable level. Recent studies show that adversarial inferences using knowledge about a disclosure algorithm can usually render the algorithm unsafe. In this paper, we show that an existing unsafe algorithm can be transformed into a large family of distinct safe algorithms, namely, k-jump algorithms. We prove that the data utility of different k-jump algorithms is generally incomparable. Therefore, a secret choice can be made among all k-jump algorithms to eliminate adversarial inferences while improving the data utility of disclosed micro-data.

1. INTRODUCTION

The issue of preserving privacy in micro-data disclosure has attracted much attention lately. Data owners, such as the Census Bureau, may need to disclose micro-data tables to the public to facilitate useful analysis. There are two seemingly conflicting goals during such a disclosure. First, the utility of disclosed data should be maximized to facilitate useful analysis. Second, the sensitive information about individuals contained in the data must be limited to an acceptable level due to privacy concerns.

The upper left tabular of Table 1 shows a toy example of micro-data table t_0. Suppose each patient's medical condition is considered as sensitive information. Simply deleting the *identifier* Name is not sufficient because the *sensitive attribute* Condition may still potentially be linked to a unique person through the *quasi-identifier* Age (more realistically, a quasi-identifier can be a combination of attributes, such as Age, Gender, and Zip Code). Nonetheless, we shall not include identifiers in the remainder of the paper.

A Micro-Data Table t_0

Name	DoB	Condition
Alice	1990	flu
Bob	1985	cold
Charlie	1974	cancer
David	1962	cancer
Eve	1953	headache
Fen	1941	toothache

Generalization $g_1(t_0)$

DoB	Condition
1980$\sim$1999	flu cold
1960$\sim$1979	cancer cancer
1940$\sim$1959	headache toothache

Generalization $g_2(t_0)$

DoB	Condition
1970$\sim$1999	flu cold cancer
1940$\sim$1969	cancer headache toothache

Generalization $g_3(t_0)$

DoB	Condition
1960$\sim$1999	flu cold cancer cancer
1940$\sim$1959	headache toothache

Table 1: A Micro-Data Table and Three Generalizations

To prevent the above *linking attack*, the micro-data table can be *generalized* to satisfy *k-anonymity* [28]. The upper right tabular in Table 1 shows a generalization $g_1(t_0)$ that satisfies 2-anonymity. That is, each generalized quasi-identifier value is now shared by at least two tuples. Therefore, a linking attack can no longer bind a person to a unique tuple through the quasi-identifier.

Nonetheless, k-anonymity by itself is not sufficient since linking a person to the second group in $g_1(t_0)$ already reveals his/her condition to be cancer. To avoid such a situation, the generalization must also ensure enough diversity inside each group of sensitive values, namely, to satisfy the l-diversity property [23]. For example, assume 2-diversity is desired. If the generalization $g_2(t_0)$ is disclosed, a person can at best be linked to a group with three different conditions among which each is equally likely to be that person's real condition. The desired privacy property is thus satisfied.

However, adversarial knowledge about a generalization algorithm itself may cause additional complications [31, 34]. First, without considering such knowledge, an adversary looking at $g_2(t_0)$ in Table 1 can guess that the three persons in each group may have the three conditions in any given order. Therefore, the adversary's mental image of t_0 is a set of totally $3! \times 3! = 36$ micro-data tables each of which is equally likely to be t_0 (a common assumption is that the quasi-identifier at-

ICDT 2010, March 22–25, 2010, Lausanne, Switzerland.

tribute, such as Age in t_0, is public knowledge). We shall call this set of tables the *permutation set*. The left-hand side of Table 2 shows two example tables in the permutation set (with the identifier Name deleted).

t_1

DoB	Condition
1990	cancer
1985	flu
1974	cold
1962	cancer
1953	headache
1941	toothache

$g_1(t_1)$

DoB	Condition
1980∼1999	cancer flu
1960∼1979	cold cancer
1940∼1959	headache toothache

t_2

DoB	Condition
1990	cold
1985	flu
1974	cancer
1962	cancer
1953	headache
1941	toothache

$g_1(t_2)$

DoB	Condition
1980∼1999	cold flu
1960∼1979	cancer cancer
1940∼1959	headache toothache

Table 2: Two Tables in the Permutation Set and Their Corresponding Generalizations under g_1

Next, assume an adversary knows the generalization algorithm has considered $g_1(t_0)$ before it discloses $g_2(t_0)$. This knowledge will enable the adversary to exclude some invalid guesses from the permutation set. For example, in Table 2, t_1 is not a valid guess, because $g_1(t_1)$ satisfies 2-diversity and should have been disclosed instead of $g_2(t_0)$. On the other hand, t_2 is a valid guess since $g_1(t_2)$ fails 2-diversity. Consequently, the adversary can refine his/her guess of t_0 to a smaller set of tables, namely, the *disclosure set*, as shown in Table 3. Since each table in the disclosure set is equally like to be t_0, the desired 2-diversity should be measured on each row of sensitive values (as a multiset). Clearly, 2-diversity is violated.

DoB	Condition			
1990	flu	cold	flu	cold
1985	cold	flu	cold	flu
1974	cancer	cancer	cancer	cancer
1962	cancer	cancer	cancer	cancer
1953	headache	headache	toothache	toothache
1941	toothache	toothache	headache	headache

Table 3: A Disclosure Set

A natural solution to the above problem is for generalization algorithms to evaluate the desired privacy property, such as l-diversity, on disclosure set in order to determine whether a generalization is safe to disclose. For example, consider how we can compute the disclosure set of next generalization, $g_3(t_0)$, in Table 1. We need to exclude every table t in the permutation set of $g_3(t_0)$, if either $g_1(t)$ or $g_2(t)$ satisfies 2-diversity. However, to determine whether $g_2(t)$ satisfies 2-diversity, we would have to compute the disclosure set of $g_2(t)$ (which is different from the disclosure set of $g_2(t_0)$ shown in Table 3). Clearly, such a recursive process is deemed to have a high cost.

The contribution of this paper is three fold. First, we show that a given generalization algorithm can be transformed into a large family of distinct algorithms under a novel strategy, called *k-jump strategy*. Intuitively, the k-jump strategy penalizes cases where recursion is required to compute the disclosure set. Therefore, algorithms may be more efficient under the k-jump strategy in contrast to the above safe strategy. Second, we prove that different algorithms under the k-jump strategy generally lead to incomparable data utility (which is also incomparable to that of algorithms under the above safe strategy). This result is somehow surprising since the k-jump strategy adopts a more drastic approach than the above safe strategy. Third, the result on data utility also has a practical impact. Specifically, while all the k-jump algorithms are still publicly known, the choice among these algorithms can be randomly chosen and kept secret, analogous to choosing a cryptographic key. The large number of k-jump algorithms will render brute force adversarial inferences infeasible.

The rest of the paper is organized as follows. Section 2 gives our model of two existing algorithms. Section 3 then introduces the k-jump strategy and discusses its properties. Section 4 presents our results on the data utility of k-jump algorithms. Section 5 reviews related work and Section 6 concludes the paper.

2. THE MODEL

We first introduce the basic model of micro-data table and generalization algorithm in Section 2.1. We then review two existing strategies and related concepts in Section 2.2. Table 4 lists our main notations which will be defined in this section.

t_0, t	Micro-data table
a, a_{naive}, a_{safe}	Generalization algorithm
$g_i(.)$, $g_i(t)$	Generalization (function)
$p(.)$	Privacy property
$per(.)$, $per(g_i(t))$, per_i, per_i^k	Permutation set
$ds(.)$, $ds(g_i(t))$, ds_i, ds_i^k	Disclosure set
$path(.)$	Evaluation path

Table 4: The Notation Table

2.1 The Basic Model

We are given the secret *micro-data table* (or simply a table) as a relation $t_0(QID, S)$ where QID and S is the *quasi-identifier attribute* and *sensitive attribute*, respectively (note that each of these can also be a sequence of attributes). We make the worst case assumption that each tuple in t_0 can be linked to a unique identifier (which has been deleted from t_0) through the QID value (if some tuples are to be deemed as not sensitive, they can be simply disregarded by the algorithm). Denote by T the set of all tables with the same schema, the same set of QID values, and the same multiset of sensitive values as those of t_0.

We are also given a *generalization algorithm* a that defines a *privacy property* $p(.) : 2^T \rightarrow \{true, false\}$ [1] and a sequence of *generalization functions* $g_i(.) : T \rightarrow G$ ($1 \leq i \leq n$) where G denotes the set of all possible *generalizations* over T (we follow the widely accepted notion of generalization given in [28]). Given t_0 as the input to the algorithm a, either a

[1] The discussion about Table 3 in Section 1 has explained why $p(.)$ should be evaluated on a set of, instead of one, tables.

generalization $g_i(t_0)$ will be the output and then disclosed, or ϕ will be the output and nothing is disclosed (we assume the adversary does not know about this fact).

Note that in a real world generalization algorithm, a generalization function may take an implicit form, such as a cut of the taxonomy tree [31]. Moreover, the sequence of generalization functions to be applied to a given table is typically decided on the fly. Our simplified model is reasonable as long as such a decision is based on the quasi-identifier (which is true in, for example, the Incognito [18]), because an adversary who knows both the quasi-identifier and the generalization algorithm can simulate the latter's execution to determine the sequence of generalization functions for the disclosed generalization.

2.2 The Algorithms a_{naive} and a_{safe}

When adversarial knowledge about a generalization algorithm is not taken into account, the algorithm can take the following *naive strategy*. Given a table t_0 and the generalization functions $g_i(.)$ ($1 \leq i \leq n$) already sorted in a non-increasing order of data utility [2], the algorithm will then evaluate the privacy property $p(.)$ on each of the n generalizations $g_i(t_0)$ ($1 \leq i \leq n$) in the given order. The first generalization $g_i(t_0)$ satisfying $p(g_i(t_0)) = true$ will be disclosed, which also maximizes the data utility.

We call $\{t : g_i(t) = g_i(t_0)\}$ the *permutation set*, and denote it by a function $per(.) : G \to 2^T$ as $per(g_i(t_0))$ (also written as per_i when both g_i and t_0 are clear from context). It is easily seen that evaluating the privacy property $p(.)$ on a generalization $g_i(t_0)$ is equivalent to evaluating $p(.)$ on the permutation set $per(g_i(t_0))$. We can thus describe the above algorithm as a_{naive} shown in Table 5. The algorithm a_{naive} defines a function $path(.) : T \to 2^{[1,n]}$ that represents the sequence of evaluated permutation sets, namely, the *evaluation path* (note that although $path(t_0)$ is defined as a set, the indices naturally form a sequence). We shall need this concept for later discussions.

```
Input: Table $t_0$;
Output: Generalization $g$ or $\phi$;
Method:
1.  Let $path(t_0) = \phi$;
2.  For $i = 1$ to $n$
3.      Let $path(t_0) = path(t_0) \cup \{i\}$;
4.      If $p(per(g_i(t_0))) = true$ then
5.          Return $g_i(t_0)$;
6.  Return $\phi$;
```

Table 5: The Algorithm a_{naive}

Unfortunately, the naive strategy leads to an unsafe algorithm (that is, an algorithm that fails to satisfy the desired privacy property). First, we need to switch to the adversary's point of view. Specifically, consider an adversary who knows the quasi-identifier $\Pi_{QID}(t_0)$, the above algorithm a_{naive}, and the disclosed generalization $g_i(t_0)$ for some $i \in [1, n]$. Given any table t, by simulating the algorithm's execution, the adversary also knows $path(t)$.

First, by only looking at the disclosed generalization $g_i(t_0)$, the adversary can deduce t_0 must be one of the tables in the *permutation set* $per(g_i(t_0))$. This inference itself does not violate the privacy property $p(.)$ since the algorithm a_{naive} does ensure $p(per(g_i(t_0)) = true$ holds before it discloses $g_i(t_0)$. However, for any $t \in per(g_i(t_0))$, the adversary can decide whether $i \in path(t)$ by simulating the algorithm's execution with t as its input.

Clearly, any $t \in per(g_i(t_0))$ can be a valid guess of the unknown t_0, only if $i \in path(t)$ is true. By excluding all invalid guesses, the adversary can obtain a smaller subset of $per(g_i(t_0))$, namely, the *disclosure set*. Formally, we define the function $ds(.) : G \to 2^T$ as $ds(g_i(t_0)) = per(g_i(t_0)) \setminus \{t : i \notin path(t)\}$, and we say $ds(g_i(t_0))$ is the disclosure set of $g_i(t_0)$.

A natural way to fix the unsafe a_{naive} is to replace the permutation set with the corresponding disclosure set in the evaluation of a privacy property. From above discussions, after $g_i(t_0)$ is disclosed, the adversary's mental image about t_0 is $ds(g_i(t_0))$. Therefore, we can simply let the algorithm to ensure $p(ds(g_i(t_0))) = true$ before it discloses any $g_i(t_0)$. We call this the *safe strategy*, and formally describe it as algorithm a_{safe} in Table 6.

```
Input: Table $t_0$;
Output: Generalization $g$ or $\phi$;
Method:
1.  Let $path(t_0) = \phi$;
2.  For $i = 1$ to $n$
3.      Let $path(t_0) = path(t_0) \cup \{i\}$;
4.      If $p(ds(g_i(t_0))) = true$ then
5.          Return $g_i(t_0)$;
6.  Return $\phi$;
```

Table 6: The Algorithm a_{safe}

Taking the adversary's point of view again, when $g_i(t_0)$ is disclosed under a_{safe}, the adversary can repeat the aforementioned process to exclude invalid guesses from $per(g_i(t_0))$, except that now ds_j ($j < i$) will be used instead of per_j. As the result, he/she will conclude that t_0 must be within the set $per(g_i(t)) \setminus \{t' : i \notin path(t')\}$, which, not surprisingly, coincides with $ds(g_i(t_0))$ (that is, the result of the adversary's inference is $t_0 \in ds(g_i(t_0))$). Since a_{safe} has ensured $p(ds(g_i(t_0))) = true$, the adversary's inference will not violate the privacy property $p(.)$. That is, a_{safe} is indeed a safe algorithm.

A subtlety here is that the definition of disclosure set may seem to be a circular definition: $ds(.)$ is defined using $path(.)$, $path(.)$ using the algorithm a_{safe}, which in turn depends on $ds(.)$. However, this is not the case. In defining the disclosure set, $ds(g_i(t))$ depends on the truth value of the condition $i \notin path(t)$. In table 6, we can observe that this truth value can be decided in line 3, right before $ds(g_i(t))$ is needed (in line 4). Therefore, both concepts are well defined.

On the other hand, we can see that for computing $ds(g_i(t_0))$, we must compute the truth value of the condition $i \notin path(t)$

[2] Our discussion does not depend on specific utility measures as long as the measure is defined based on quasi-identifiers.

for every $t \in per(g_i(t_0))$. Moreover, to construct $path(t)$ requires us to simulate the execution of a_{safe} with t as the input. Therefore, to compute $ds(g_i(t_0))$, we will have to compute $ds(g_j(t))$ for all $t \in per(g_i(t_0))$ and $j = 1, 2, \dots, i-1$. Clearly, this is an expensive process. In next section, we shall investigate a novel family of algorithms for reducing the cost.

3. K-JUMP STRATEGY

We first introduce the k-jump strategy in Section 3.1. We then discuss its properties in Section 3.2.

3.1 The Algorithm Family $a_{jump}(\vec{k})$

In the previous section, we have shown that the naive strategy is unsafe, and the safe strategy is safe but may incur a high cost due to the inherently recursive process. First, we more closely examine the limitation of these algorithms in order to build intuitions toward our new solution. In Figure 1, the upper and middle chart shows the decision process of the previous two algorithms, a_{naive} and a_{safe}, respectively. Each box represents the i^{th} iteration of the algorithm. Each diamond represents an evaluation of the privacy property $p(.)$ on the set inside the diamond, and the symbol Y and N denotes the result of such an evaluation to be $true$ and $false$, respectively.

Comparing the two charts, we can have four different cases in each iteration of the algorithm (some iterations actually have less possibilities, as we shall show later):

- First, if $p(per_i) = p(ds_i) = false$ (recall that per_i is an abbreviation of $per(g_i(t_0))$), then clearly, both algorithms will immediately move to the next iteration.
- Second, if $p(per_i) = p(ds_i) = true$, both algorithms will disclose $g_i(t_0)$ and terminates.
- Third, we delay the discussion of the case of $p(per_i) = false \wedge p(ds_i) = true$ to later sections.
- Finally, we can see the last case, $p(per_i) = true \wedge p(ds_i) = false$, is the main reason that a_{naive} is unsafe, and that a_{safe} must compute the disclosure set and consequently result in an expensive recursive process.

Therefore, informally, we penalize the last case, by *jumping* over the next $k-1$ iterations of the algorithm. As a result, we have the *k-jump strategy* as illustrated in the lower chart of Figure 1. More formally, the family of algorithms under the k-jump strategy is shown in Table 7.

There are two main differences between $a_{jump}(\vec{k})$ and a_{safe}. First, since now in each iteration the algorithm may evaluate per_i and ds_i, or per_i only, we slightly change the definition of evaluation path to be $path(.) : T \to 2^{[1,n]\times\{0,1\}}$ so $(i,0)$ stands for per_i and $(i,1)$ for ds_i. Consequently, the definition of a disclosure set also needs to be revised by replacing the condition $i \notin path(t)$ with $(i,1) \notin path(t)$.

Second, the algorithm family $a_{jump}(\vec{k})$ takes an additional input, an n-dimensional vector $\vec{k} \in [1,n]^n$, namely, the *jump distance* vector. In the case of $p(per_i) = true \wedge p(ds_i) = false$, the algorithm will directly jump to the $(i + \vec{k}[i])^{th}$ iteration (note that jumping to the i^{th} iteration for any $i > n$ will simply lead to line 10 of the algorithm, that is, to disclose nothing). In the special case that $\forall i \in [1,n]$ $\vec{k}[i] = k$ for some integer k, we shall abuse the notation to simply use k for $\vec{k}$.

```
Input: Table $t_0$, vector $\vec{k} \in [1,n]^n$;
Output: Generalization $g$ or $\phi$;
Method:
1.  Let $path(t_0) = \phi$;
2.  Let $i = 1$;
3.  While ($i \leq n$)
4.          Let $path(t_0) = path(t_0) \cup \{(i,0)\}$;
                //the pair $(i,0)$ represents $per_i$
5.          If $p(per(g_i(t_0))) = true$ then
6.              Let $path(t_0) = path(t_0) \cup \{(i,1)\}$;
                    //the pair $(i,1)$ represents $ds_i$
7.              If $p(ds(g_i(t_0))) = true$ then
8.                  Return $g_i(t_0)$;
9.              Else
10.                 Let $i = i + \vec{k}[i]$;
                        //$\vec{k}[i]$ is the $i^{th}$ element of $\vec{k}$
11. Return $\phi$;
```

Table 7: The Algorithm Family $a_{jump}(\vec{k})$

Despite the difference between a_{safe} and $a_{jump}(\vec{k})$, the final condition for disclosing a generalization remains the same, that is, $p(ds_i) = true$. This simple fact suffices to show $a_{jump}(\vec{k})$ to be a safe family of algorithms.

3.2 Properties of $a_{jump}(\vec{k})$

We discuss several properties of the algorithms $a_{jump}(\vec{k})$ in the following.

- **Computation of the Disclosure Set** Again, the disclosure set is well defined under $a_{jump}(\vec{k})$, although it may seem to be a circular definition at first glance. First, $ds(g_i(t))$ depends on the truth value of the condition $(i,1) \notin path(t)$. In table 7, we can then observe that this value can be decided in line 5, right before $ds(g_i(t))$ is needed (in line 6).

  Although computing disclosure sets under $a_{jump}(\vec{k})$ is similar to that under a_{safe}, the former is generally more efficient. Specifically, recall that under a_{safe}, to compute $ds(g_i(t_0))$ we must first compute $ds(g_j(t))$ for all $t \in per(g_i(t_0))$ and $j = 1, 2, \dots, i-1$. In contrast, this expensive recursive process in not always necessary under $a_{jump}(\vec{k})$.

  Referring to the lower chart in Figure 1, to compute $ds(g_i(t_0))$ for any $2 < i < 2 + k$, we no longer need to always compute $ds(g_2(t))$ for every $t \in per_i$. By definition, $ds(g_i(t_0)) = per(g_i(t_0)) \setminus \{t : (i,1) \notin path(t)\}$. From the chart, it is evident that $(i,1) \notin path(t)$ is true as long as $p(per(g_2(t))) = true$ (in which case $path(t)$ will either terminates at ds_2 or jump over the i^{th} iteration). Therefore, for any such table

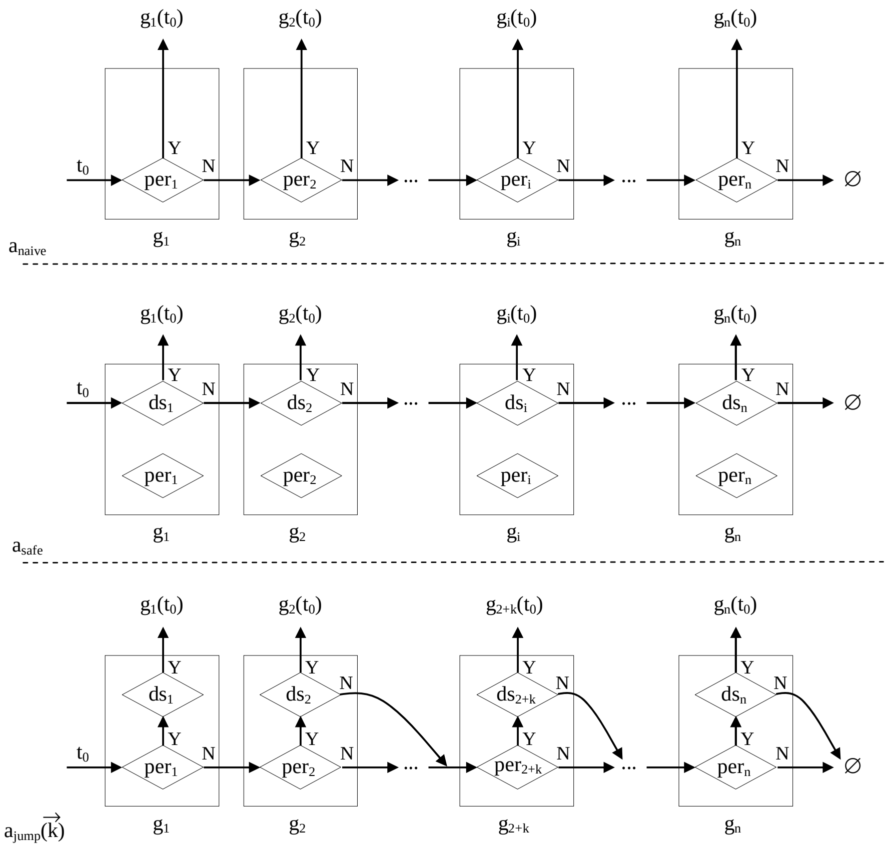

Figure 1: The Decision Process of Different Strategies

t, we do not need to compute $ds(g_2(t))$ in computing $ds(g_i(t_0))$.

As an extreme case, when the jump distance vector is $(n, n-1, \ldots, 1)$, all the jumps end at ϕ (disclosing noting). In this case, the computation of disclosure set is no longer a recursive process. To compute $ds(g_i(t_0))$, it suffices to only compute $per(g_j(t))$ for $t \in per(g_i(t_0))$ and $j = 1, 2, \ldots, i-1$. The complexity is thus significantly lower.

- $ds(g_1(t_0))$ **and** $ds(g_2(t_0))$ The first two disclosure sets have some special properties. First of all, $ds(g_1(t_0) = per(g_1(t_0))$ is true. Intuitively, since any given table itself generally does not satisfy the privacy property, in computing ds_1, an adversary cannot exclude any table from per_1. More specifically, when $g_1(t_0)$ is disclosed, for all $t \in per(g_1(t_0))$, $path(t)$ must always end at ds_1, because $p(per(g_1(t))) = true$ follows from the fact that $per(g_1(t)) = per(g_1(t_0))$ (by the definition of permutation set) and $p(per(g_1(t_0))) = true$ (by the fact that $g_1(t_0)$ is disclosed). Therefore, $ds(g_1(t_0)) = per(g_1(t_0)) \backslash \{t : (1,1) \notin path(t)\}$ yields $ds(g_1(t_0) = per(g_1(t_0))$.

  Second, we show that $ds(g_2(t_0))$ is independent of the distance vector $\vec{k}$. That is, all algorithms in $a_{jump}(\vec{k})$ share the same $ds(g_2(t_0))$. By definition, $ds(g_2(t_0)) = per(g_2(t_0)) \setminus \{t : (2,1) \notin path(t)\}$. As $ds(g_1(t_0) = per(g_1(t_0))$ is true, the case $p(per(g_1(t_0))) = true \wedge p(ds(g_1(t_0))) = false$ is impossible, and consequently the jump from ds_1 is never to happen (which explains the missing edge in the lower chart of Figure 1). Therefore, the condition $(2,1) \notin path(t)$ does not depend on the distance vector $\vec{k}$.

- **Size of the Family** First, with n generalization functions, we can have roughly $(n-1)!$ different jump distance vectors since the i^{th} $(2 \leq i \leq n)$ iteration may jump to $(n-i+1)$ different destinations (that is, $i+1, i+2, \ldots, n+1$, where the $(n+1)^{th}$ iteration means disclosing nothing). Clearly, $(n-1)!$ is a very large number even for a reasonably large n. Moreover, the space of jump distance vectors will be further increased when we *reuse* generalization functions in a meaningful way, as will be shown in later sections. Therefore, we can now transform any given unsafe algorithm a_{naive} into a large family of safe algorithms. This fact lays a foundation for making secret choices of k-jump algorithm to prevent adversarial inferences.

  Note here the jump distance refers to possible ways an algorithm may jump at each iteration, which is not to be confused with the evaluation path of a specific table. For example, the vector $(n, n-1, \ldots, 1)$ yields a valid k-jump algorithm that always jumps to disclosing nothing, whereas any specific evaluation path can include at most one of such jumps. There is also another plausible but false perception related to this. That is, an algorithm with the jump distance k (note that here k denotes a vector whose elements are all equal to k) will only disclose a generalization under $g_i(.)$ where i is a multiplication of k. This perception may lead to false statements about data utility, for example, that the data utility for $k = 2$ is better than that for $k = 4$. In fact, regardless of the jump distance, an algorithm may potentially disclose a generalization under every $g_i(.)$. The reason is that each jump is only possible, but not mandatory for a specific table.

4. DATA UTILITY COMPARISON

In this section, we compare the data utility of different algorithms from the same family. Section 4.1 considers the family of k-jump algorithms. Section 4.2 studies the case when some generalization functions are reused in an algorithm. Section 4.3 addresses a_{safe}.

4.1 Data Utility of k-Jump Algorithms

Our main result is that the data utility of two k-jump algorithms $a_{jump}(\vec{k})$ and $a_{jump}(\vec{k'})$ from the same family is generally incomparable [3]. That is, the data utility cannot simply be ordered based on the jump distance of two algorithms.

We do not rely on specific utility measures. Instead, the generalization functions are assumed to be sorted in a non-increasing order of their data utility. Consequently, an algorithm a_1 is considered to have better or equal data utility compared to another algorithm a_2 (both algorithms are from the same family), if we can construct a table t for which a_1 returns $g_i(t)$ and a_2 returns $g_j(t)$, with $i < j$.

Such a construction is possible with two methods. First, we let $path(t)$ under a_2 to jump over the iteration in which a_1 terminates. Second, when the first method is not an option, we let $path(t)$ under a_2 to include a disclosure set that does not satisfy the privacy property $p(.)$, whereas $path(t)$ under a_1 to include one that does. We first consider the following two special cases.

- $a_{jump}(1)$ and $a_{jump}(i)$ $(i > 1)$ In this case, the evaluation path of $a_{jump}(1)$ can never jump over that of $a_{jump}(i)$ (in fact, a jump distance of 1 means no jump at all). Therefore, we apply the above second method, that is, to rely on different disclosure sets of the same disclosed generalization.
- $a_{jump}(i)$ and $a_{jump}(j)$ $(1 < i < j)$ For this case, we apply the above first method, that is, by constructing an evaluation path that jumps over the other.

From now on, we shall add superscripts to existing notations to denote the distance vector of different algorithms. For example, ds_1^k means the disclosure set ds_1 under the algorithm $a_{jump}(k)$. First, we need the following result.

LEMMA 1. *For any $a_{jump}(1)$ and $a_{jump}(i)$ $(i > 1)$ algorithms from the same family, we have $ds_3^i \subseteq ds_3^1$.*

PROOF. By definition, we have

$$ds_3^1(t_0) = per_3(t_0))/\{t|(t \in per_3(t_0)) \wedge (p(per_1(t)) = true \vee (p(per_2(t)) = true \wedge p(ds_2^1(t)) = true))\} \quad (1)$$

$$ds_3^i(t_0) = per_3(t_0)/\{t|(t \in per_3(t_0)) \wedge (p(per_1(t)) = true \vee p(per_2(t)) = true)\} \quad (2)$$

from which the result follows. □

From Lemma 1, we can have the following straightforward result needed for proving Theorem 1.

LEMMA 2. *The data utility of $a_{jump}(1)$ is always better than or equal to that of $a_{jump}(i)$ $(i > 1)$ when both algorithms are from the same family with a set-monotonic [4] privacy property $p(.)$ and $n = 3$.*

THEOREM 1. *For any $i > 1$, there always exist cases in which the data utility of the algorithm $a_{jump}(i)$ is better than that of $a_{jump}(1)$, and vice versa.*

PROOF. The key is to have different disclosure sets ds_3 under the two algorithms such that one satisfies $p(.)$ and the other fails. By Lemma 2, the case where the data utility of $a_{jump}(1)$ is better than or equal to that of $a_{jump}(i)$ $(i > 1)$ is trivial to construct and hence is omitted.

We only show the other case where $a_{jump}(i)$ has better data utility. Basically, we need to design a table to satisfy the following. First, per_1 and per_2 do not satisfy $p(.)$ while per_3 does. Second, $p(ds_3^i) = true$ and $p(ds_3^1) = false$ are both true.

Table 8 shows our construction for the proof. The privacy property $p(.)$ [5] is that the highest ratio of a sensitive value in a group must be no greater than $\frac{1}{2}$. We show that $a_{jump}(i)$ can disclose using g_3, whereas $a_{jump}(1)$ cannot.

1. For this special case, $ds_3^k(t_0)$ can be computed by first excluding any table t for which $p(per_1(t)) = true$. The tables in $ds_3^i(t_0)$ must belong to one of the following four disjoint sets.

   In the first case, I has sensitive value C_6. The number of tables in this case is $\binom{2}{1} \times \binom{2}{1} \times (\binom{4}{1} \times \binom{3}{1}) \times (\binom{6}{2} \times \binom{4}{2}) = 48 \times 90 = 4320$. Denote this set by S_1. In the other three cases, I does not have C_6 and both N and O have C_7, C_8, or C_9, denoted respectively by S_2, S_3, and S_4. Each of these includes $\binom{2}{1} \times \binom{2}{1} \times (\binom{4}{1} \times \binom{3}{1}) \times \binom{2}{1} \times (\binom{4}{1} \times \binom{3}{1}) = 48 \times 24 = 1152$ tables.

   Now consider generalizing these tables using g_2. All tables in the last three sets cannot be disclosed under g_2

[3] Here the comparison of data utility is independent of the given table, which explains why the notation $a_{jump}(\vec{k})$ does not indicate the given table.

[4] That is, $p(S) = true$ implies $\forall S' \supseteq S$ $p(S') = true$.

[5] Notice that here (and in the remainder of the paper) $p(.)$ is not necessarily set-monotonic.

since each of their permutation sets under g_2 fails the privacy property. For the same reason, tables in the first set in which both N and O have C_7, C_8, or C_9, which is denoted as S_1', cannot be disclosed under g_2, either. The cardinality of S_1' is $\binom{2}{1}\times\binom{2}{1}\times(\binom{4}{1}\times\binom{3}{1})\times\binom{4}{2}\times\binom{3}{1} = 48\times 18 = 864$.

For $a_{jump}(i)$, all the tables in $(S_1 \setminus S_1')$ will be excluded from $ds_3^i(t_0)$. The reason is the following. Each of their permutation sets under g_2 satisfies the privacy property, so $a_{jump}(i)$ will disclose them either under g_2 or after g_3. Therefore, $ds_3^i(t_0) = S_1' \cup S_2 \cup S_3 \cup S_4$. The highest ratio of sensitive value is that of A and B associated with C_0 or C_1, which is $\frac{1}{2}$. Since $ds_3^i(t_0)$ satisfies the privacy property, it can be disclosed using g_3 under $a_{jump}(i)$.

2. As to the case of $a_{jump}(1)$, the disclosure set of all the tables in $S_1 \setminus S_1'$ do not satisfy the privacy property and hence all of them cannot be removed from $ds_3^1(t_0)$. The reason is as follows. First, the permutation set of each such table under g_2 satisfies the privacy property. Next, consider their disclosure sets under g_2. The set $S_1 \setminus S_1'$ can be further divided into three disjoint subsets as follows.

    - Either N or O has C_7 and the other has C_8. This subset has $\binom{2}{1}\times\binom{2}{1}\times(\binom{4}{1}\times\binom{3}{1})\times\binom{1}{1}\times(\binom{4}{1}\times\binom{3}{1})\times\binom{2}{1} = 48\times 24 = 1152$ tables. Based on the sensitive value of H, this subset can be further divided into two disjoint subsets again.

        (a) H has C_6. This subset has $\binom{2}{1}\times\binom{2}{1}\times(\binom{3}{1}\times\binom{2}{1})\times\binom{1}{1}\times(\binom{4}{1}\times\binom{3}{1})\times\binom{2}{1} = 48\times 12 = 576$ tables. For each table in this subset, to obtain its disclosure set, we must exclude the tables that can be disclosed under g_1 from its permutation set following the same rule as above. The tables in its disclosure set must satisfy that both H and I have C_6. The ratio of both H and I being associated with C_6 is $1.0 > 0.5$. This clearly violates the privacy property.

        (b) H does not have sensitive value C_6, but has either C_4 or C_5. This subset has $\binom{2}{1}\times\binom{2}{1}\times\binom{3}{1}\times\binom{2}{1}\times\binom{1}{1}\times(\binom{4}{1}\times\binom{3}{1})\times\binom{2}{1} = 48\times 12 = 576$ tables. Similarly, the tables in the disclosure set must satisfy that two from the set $\{E, F, G\}$ have C_6. Moreover, one and only one of H and I has C_6. Therefore, the ratio of both E, F, and G being associated with C_6 is $\frac{2}{3} > 0.5$. This also violates the privacy property.

      In summary, the disclosure set of every table in this subset under function g_2 will violate the privacy property, and consequently these tables cannot be disclosed under g_2. Therefore, the algorithm $a_{jump}(1)$ must continue to evaluate these tables under g_3 whose permutation set satisfies the privacy property.

    - The other two cases are that N and O have C_7 and C_9, respectively, or C_8 and C_9, respectively. Similarly, each has 1152 tables, and for the same reason as above, the disclosure set of each table in each subset does not satisfy the privacy property, and hence cannot be disclosed under g_2.

   Consequently, all the tables in $S_1 \setminus S_1'$ cannot be removed from $ds_3^1(t_0)$. Therefore, $ds_3^1(t_0) = S_1 \cup S_2 \cup S_3 \cup S_4$. The ratio of I being associated with C_6 is $\frac{48\times 90}{48\times(90+24\times 3)} = 0.556 > 0.5$. This violates the privacy property. Therefore, the given table cannot be disclosed using g_3 under $a_{jump}(1)$.

□

QID	g_1	g_2	g_3	$\dots$
A	C_0	C_0	C_0	$\dots$
B	C_1	C_1	C_1	$\dots$
C	C_2	C_2	C_2	$\dots$
D	C_3	C_3	C_3	$\dots$
E	C_4	C_4	C_4	$\dots$
F	C_5	C_5	C_5	$\dots$
G	C_6	C_6	C_6	$\dots$
H	C_6	C_6	C_6	$\dots$
I	C_6	C_6	C_6	$\dots$
J	C_7	C_7	C_7	$\dots$
K	C_7	C_7	C_7	$\dots$
L	C_8	C_8	C_8	$\dots$
M	C_8	C_8	C_8	$\dots$
N	C_9	C_9	C_9	$\dots$
O	C_9	C_9	C_9	$\dots$

Table 8: The Case Where $a_{jump}(i)$ Has Better Utility Than $a_{jump}(1)$

Next, we prove the data utility of $a_{jump}(i)$ and $a_{jump}(j)$ to be incomparable by constructing non-overlapping evaluation paths.

THEOREM 2. *For any $j > i > 1$, there always exist cases where the data utility of the algorithm $a_{jump}(i)$ is better than that of $a_{jump}(j)$, and vice versa.*

PROOF SKETCH. Since both $a_{jump}(i)$ and $a_{jump}(j)$ can jump over iterations of the algorithm, we can easily construct evaluation paths for the proof. Figure 2 illustrates such constructed paths.

The case where $a_{jump}(i)$ has better utility than $a_{jump}(j)$ $(1 < i < j)$ is relatively easier to construct. We only show the construction for the other case. We basically need to construct a case satisfying the following conditions:

$$\begin{cases} (\text{if } \omega = 1), & p(per_\omega) = false; \\ (\text{if } \omega = 2), & p(per_\omega) = true \wedge p(ds_\omega^{i;j}) = false; \\ (\forall \omega \in [3, j]), & p(per_\omega) = false; \\ (\forall \omega \in [j+1, j+2]), & p(per_\omega) = true; \\ (\text{if } \omega = j+1), & p(ds_\omega^{i}) = false; \\ (\text{if } \omega = j+2), & p(ds_\omega^{j}) = true. \end{cases}$$

The above conditions imply that g_{j+2} will be used to disclose under $a_{jump}(j)$. On the other hand, when $a_{jump}(i)$ evaluates g_{i+2}, since its permutation set does not satisfy the privacy property, the algorithm will move to the the next function, and repeat this until it reaches g_{j+1}. Since $ds^{i}_{j+1}(t_0)$ does not satisfy the privacy property, the algorithm will jump to g_{j+1+i} and will disclose using a function beyond g_{j+2}.

Table 9 shows our construction where the privacy property is again that the highest ratio of a sensitive value is no greater than $\frac{1}{2}$. We assume the table has many others tuples not shown (the purpose of these additional tuples is only to ensure the data utility of the generalizations is in a non-increasing order). Although we shall omit the details due to space limitations, it can be verified that with this construction, both $a_{jump}(i)$ and $a_{jump}(j)$ will follow the desired evaluation paths as shown in Figure 2. □

g_1	g_2	g_3	$\ldots$	g_j	g_{j+1}	g_{j+2}	$\ldots$
C_0	C_0	C_0	$\ldots$	C_0	C_0	C_0	$\ldots$
C_1	C_1	C_1	$\ldots$	C_1	C_1	C_1	$\ldots$
C_2	C_2	C_2	$\ldots$	C_2	C_2	C_2	$\ldots$
C_3	C_3	C_3	$\ldots$	C_3	C_3	C_3	$\ldots$
C_4	C_4	C_4	$\ldots$	C_4	C_4	C_4	$\ldots$
S	S	S	$\ldots$	S	S	S	$\ldots$
S	S	S	$\ldots$	S	S	S	$\ldots$
C_5	C_5	C_5	$\ldots$	C_5	C_5	C_5	$\ldots$
C_6	C_6	C_6	$\ldots$	C_6	C_6	C_6	$\ldots$
C_7	C_7	C_7	$\ldots$	C_7	C_7	C_7	$\ldots$
C_8	C_8	C_8	$\ldots$	C_8	C_8	C_8	$\ldots$
C_9	C_9	C_9	$\ldots$	C_9	C_9	C_9	$\ldots$
$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$	$\ldots$

Table 9: The Case Where the Data Utility of $a_{jump}(j)$ is better than that of $a_{jump}(i)$ ($1 < i < j$)

Next, we extend the above results to the more general case in which the two algorithms $a_{jump}(\vec{k})$ and $a_{jump}(\vec{k'})$ both have an n-dimensional vector as their jump distances.

THEOREM 3. *For any $\vec{k_1}, \vec{k_2} \in [1, n]^n$, there always exist cases in which the data utility of the algorithm $a_{jump}(\vec{k_1})$ is better than that of $a_{jump}(\vec{k_2})$, and vice versa.*

PROOF SKETCH. Suppose the first different element of $\vec{k_1}$ and $\vec{k_2}$ is the i^{th} element. Without the loss of generality, assume that $\vec{k_1}[i] < \vec{k_2}[i]$. There are two cases as follows,

1. $\vec{k_1}[i] = 1$: Since $ds^{\vec{k_1}}_{l} = ds^{\vec{k_2}}_{l}$ for all $1 \leq l \leq i$, and $ds^{\vec{k_1}}_{i+1} \supseteq ds^{\vec{k_2}}_{i+1}$, we can construct in a similar way as in the proof of Theorem 1. Basically, we construct the following evaluation path: $per_1 \to per_2 \to \ldots \to per_i \to per_{i+1} \to ds_{i+1}$ so that in one case we have $p(ds^{\vec{k_1}}_{i+1}) = true \wedge p(ds^{\vec{k_2}}_{i+1}) = false$, whereas in the other case we have $p(ds^{\vec{k_1}}_{i+1}) = false \wedge p(ds^{\vec{k_2}}_{i+1}) = true$.

2. $\vec{k_1}[i] > 1$: In this case, we consider two sub-cases.

   (a) $(\exists j)((i + \vec{k_1}[i] \leq j < i + \vec{k_2}[i]) \wedge (j + \vec{k_1}[j] > i + \vec{k_2}[i]))$:

   In this sub-case, we can construct the following two evaluation paths.

   i. $a_{jump}(\vec{k_1}) : per_1 \to per_2 \to \ldots \to per_i \to ds^{\vec{k_1}}_{i} \to per_{i+\vec{k_1}[i]} \to \ldots \to per_j \to ds^{\vec{k_1}}_{j} \to per_{j+\vec{k_1}[j]} \to \ldots$
   $a_{jump}(\vec{k_2}) : per_1 \to per_2 \to \ldots \to per_i \to ds^{\vec{k_2}}_{i} \to per_{i+\vec{k_2}[i]} \to p(ds^{\vec{k_2}}_{i+\vec{k_2}[i]}) = true$

   ii. $a_{jump}(\vec{k_1}) : per_1 \to per_2 \to \ldots \to per_i \to ds^{\vec{k_1}}_{i} \to per_{i+\vec{k_1}[i]} \to p(ds^{\vec{k_1}}_{i+\vec{k_1}[i]}) = true$
   $a_{jump}(\vec{k_2}) : per_1 \to per_2 \to \ldots \to per_i \to ds^{\vec{k_2}}_{i} \to per_{i+\vec{k_2}[i]} \to \ldots$

   Since $j + \vec{k_1}[j] > i + \vec{k_2}[i]$, the data utility of $a_{jump}(\vec{k_1})$ in the first case is worse than that of $a_{jump}(\vec{k_2})$. Meanwhile, since $i+\vec{k_1}[i] < i+\vec{k_2}[i]$, we have the converse result in the second case.

   (b) $\neg(\exists j)((i + \vec{k_1}[i] \leq j < i + \vec{k_2}[i]) \wedge (j + \vec{k_1}[j] > i + \vec{k_2}[i]))$:
   In this sub-case, suppose that $ds^{\vec{k_1}}_{j} \neq ds^{\vec{k_2}}_{j}$ for certain $i < j \leq i + \vec{k_2}[i]$, we can construct an evaluation path in which all the permutation sets fail to satisfy the privacy property until the j^{th} iteration. Then, we construct a case where the disclosure set of g_j under $a_{jump}(\vec{k_1})$ satisfies the privacy property while it does not under $a_{jump}(\vec{k_2})$, and another case for the converse result. The detailed construction is omitted.

□

4.2 Reusing Generalization Functions

With the naive strategy, whether a generalization function satisfies the privacy property is independent of other functions. Therefore, it is meaningless to evaluate the same function more than once. However, we now show that with the k-jump strategy, it is meaningful to *reuse* a generalization function along the evaluation path. This will either increase the data utility of the original algorithm, or lead to new algorithms with incomparable data utility to enrich the the existing family of algorithms. That is, reusing generalization functions may benefit the optimization of data utility.

THEOREM 4. *Given the set of generalization functions, there always exist cases in which the data utility of the algorithm with reusing generalization functions is better than that of the algorithm without reusing, and vice versa.*

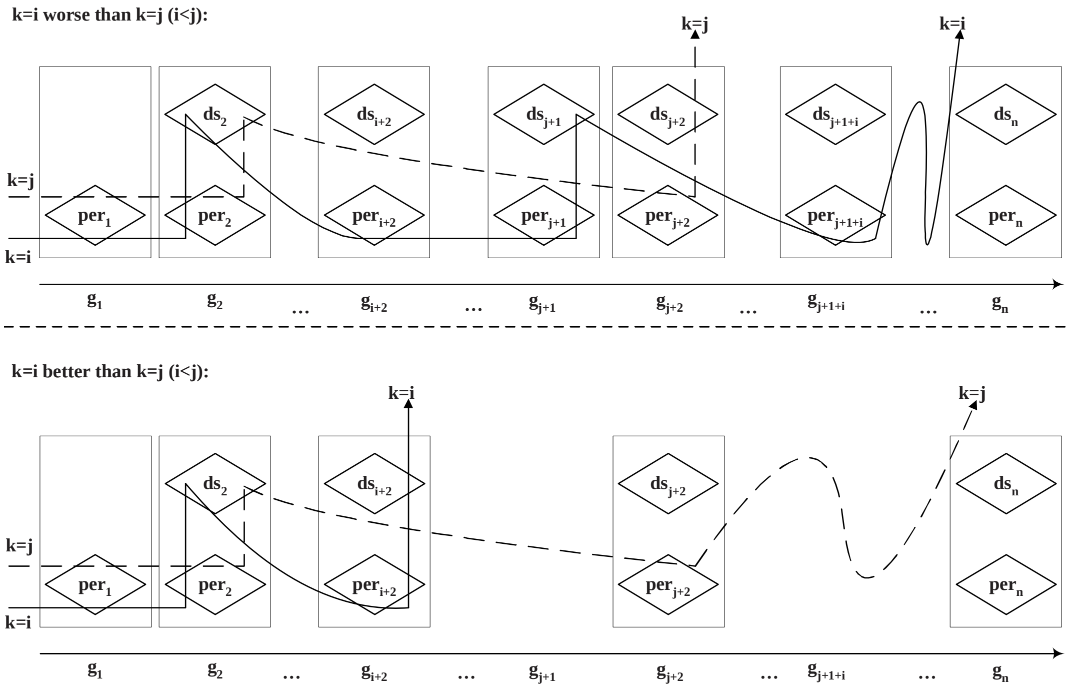

Figure 2: The Construction for $a_{jump}(i)$ and $a_{jump}(j)$ ($1 < i < j$)

PROOF. Consider two algorithms a_1 and a_2 that define the functions $g_1, g_2, g_3, g_4, \ldots$ and $g_1, g_2, g_3, g_{2'}, g_4, \ldots$, respectively, where $g_{2'}(.)$ and $g_2()$ are identical. Suppose both algorithms has the same jump distance $k = 1$, and the privacy property is not set-monotonic. We can construct the following two evaluation paths.

1. $a_1(t_0) : per_1(t_0) \rightarrow per_2(t_0) \rightarrow ds_2^1(t_0) \rightarrow per_3(t_0) \rightarrow per_4(t_0) \ldots$
   $a_2(t_0) : per_1(t_0) \rightarrow per_2(t_0) \rightarrow ds_2^1(t_0) \rightarrow per_3(t_0) \rightarrow per_{2'}(t_0) \rightarrow ds_{2'}^1(t_0) \rightarrow p(ds_{2'}^1(t_0)) = true$
2. $a_1(t_0) : per_1(t_0) \rightarrow per_2(t_0) \rightarrow per_3(t_0) \rightarrow per_4(t_0) \rightarrow ds_4^1(t_0) \rightarrow p(ds_4^1(t_0)) = true$
   $a_2(t_0) : per_1(t_0) \rightarrow per_2(t_0) \rightarrow per_3(t_0) \rightarrow per_{2'}(t_0) \rightarrow per_4(t_0) \rightarrow ds_4^1(t_0) \rightarrow p(ds_4^1(t_0)) = false$

Clearly, the data utility of a_1 in the first case is worse than that of a_2, while in the second case it is better. □

It is worth noting that although the same generalization function is repetitively evaluated, its disclosure set will depend on the functions that appear before it in the evaluation path. Take the identical functions g_2 and g_2' above as an example, the disclosure set of g_2 is computed by excluding from its permutation set the tables which can be disclosed under g_1; however, the disclosure set of g_2' needs to further exclude tables which can be disclosed under g_3. Therefore, $ds_{2'} \subseteq ds_2$. This leads to the following.

PROPOSITION 1. *With a set-monotonic privacy property, reusing generalization functions in a k-jump algorithm does not affect the data utility under $a_{jump}(1)$.*

PROOF SKETCH. Suppose $g_i(.)$ is reused as $g_{i'}(.)$ in a later iteration of the algorithm. $p(ds_{i'}(t_0)) = true$ implies $p(ds_i(t_0)) = true$ so that the algorithm will disclose under $g_i(.)$; if $p(ds_{i'}(t_0)) = false$ then the algorithm will continue to the next iteration. In both cases, $g_{i'}(.)$ does not affect the data utility. □

On the other hand, when generalization functions are reused at the end of the original sequence of functions, some tables which will lead to disclosing nothing under the original sequence of functions may have a chance to be disclosed under the reused functions, which will improve the data utility.

PROPOSITION 2. *Reusing a generalization function after the last iteration of an existing k-jump algorithm may improve the data utility when $p(.)$ is not set-monotonic.*

PROOF SKETCH. We construct a case in which reusing a function will improve the data utility. Consider two algorithms a_1 and a_2 that define the functions g_1, g_2, g_3 and $g_1, g_2, g_3, g_{2'}$, respectively, where $g_{2'}(.)$ and $g_2()$ are identical. Suppose both algorithms have the same jump distance $k = 1$, and the privacy property is not set-monotonic. We need to construct the following two evaluation paths by which a_1 will disclose nothing, while a_2 will disclose using $g_{2'}$.

1. $a_1(x) : per_1(x) \rightarrow per_2(x) \rightarrow ds_2^1(x) \rightarrow p(per_3(x)) = false$
2. $a_2(x) : per_1(x) \rightarrow per_2(x) \rightarrow ds_2^1(x) \rightarrow per_3(x) \rightarrow per_{2'}(x) \rightarrow ds_{2'}^1(x) \rightarrow p(ds_{2'}^1(x)) = true$

Table 10 shows our construction. The table will lead to disclosing nothing without reusing g_2, whereas reusing g_2 will lead to a successful disclosure. In this example, the jump distance is 1, and the privacy property is that the highest ratio of any sensitive value is no greater than $\frac{1}{2}$.

More specifically, the given table, denoted by t_0, cannot be disclosed under $g_1(.)$ or $g_3(.)$ since $p(per_1) = p(per_3) = false$. For g_2, we have $p(per_2) = true$. The tables in ds_2 must be in one of the following three disjoint sets.

1. C has the sensitive value C_3. The number of such tables is $\binom{2}{1} \times (\binom{4}{1} \times \binom{3}{1}) = 24$. Denote this set by S_1.
2. C does not have C_3, and both D and E have C_3. There are $\binom{2}{1} \times \binom{2}{1} \times \binom{2}{1} = 8$ such tables. Denote this set by S_2.
3. C does not have C_3, and both F and G have C_3. There are 8 such tables. Denote this set by S_3.

We then have $ds_2 = S_1 \cup S_2 \cup S_3$. The ratio of C being associated with C_3 is $\frac{24}{24+8+8} = 0.6 > 0.5$, so $g_2(t_0)$ cannot be disclosed, either.

Now, consider the case that g_2 is reused as $g_{2'}$. To calculate the disclosure set of $ds_{2'}$, the tables which can be disclosed under g_1, g_2, and g_3 must be excluded from per_2. We have that the remaining tables in $ds_{2'}$ are the same as above, that is, $S_1 \cup S_2 \cup S_3$. These tables cannot be disclosed under g_2 as mentioned above. S_1 can be further divided into three disjoint subsets as follows.

1. One and only one of D and E has C_3, so does F and G. This subset has $\binom{2}{1} \times \binom{2}{1} \times \binom{2}{1} \times \binom{2}{1} = 16$ tables, and is denoted by S_{1_1}.
2. Both D and E have C_3. This subset has $\binom{2}{1} \times \binom{2}{1} = 4$ tables, and is denoted by S_{1_2}.
3. Both F and G have C_3. This subset also has 4 tables, and is denoted by S_{1_3}.

All the tables in S_{1_2}, S_{1_3}, S_2, and S_3 cannot be disclosed under g_3 since their permutation sets under g_3 do not satisfy the privacy property (the highest ratios of a sensitive value are respectively 0.6, 1.0, 0.6, and 1.0). On the other hand, the tables in S_{1_1} can be disclosed under g_3 (the highest ratio is 0.5, which is the ratio of F and G being associated with C_3 and C_4 (or C_5)). The disclosure set under the reused function $g_{2'}$ is $ds_{2'} = S_{1_2} \cup S_{1_3} \cup S_2 \cup S_3$. The ratio of A and B being associated with C_1 or C_2 are 0.5, which is the highest ratio. Therefore, $g_{2'}(t_0)$ can be safely disclosed. □

4.3 a_{safe} and $a_{jump}(1)$

We show that the algorithm a_{safe} is equivalent to $a_{jump}(1)$ when the privacy property is either set-monotonic, or based on the highest ratio of sensitive values.

Given a group EC_i in the disclosed generalization, let nr_i be the number of tuples and ns_i be the number of unique sensitive values. Denote the sensitive values within EC_i by $\{s_{i.1}, s_{i.2}, \ldots, s_{i.ns_i}\}$. Denote by $n_{s_{i.j}}$ the number of tuples associated with $s_{i.j}$.

QID	g_1	g_2	g_3	$g_{2'}$
A	C_1	C_1	C_1	C_1
B	C_2	C_2	C_2	C_2
C	C_3	C_3	C_3	C_3
D	C_4	C_4	C_4	C_4
E	C_5	C_5	C_5	C_5
F	C_3	C_3	C_3	C_3
G	C_3	C_3	C_3	C_3

Table 10: The Case Where Reusing Generalization Functions Improves Data Utility

LEMMA 3. *If the privacy property is either set-monotonic or based on the highest ratio of sensitive values, then a permutation set not satisfying the privacy property will imply that any of its subsets does not, either.*

PROOF SKETCH. The result is obvious if the privacy property is set-monotonic. Now consider a privacy property based on the highest ratio of sensitive values, which is supposed to be no greater than a given δ. Suppose that EC_i is a group that does not satisfy the privacy property, and in particular, $s_{i.j}$ is a sensitive value that leads to the violation. First, we have that $\frac{n_{s_{i.j}}}{nr_i} > \delta$. Let nt be the cardinality of any subset of the permutation set. Since all tables in this subset have the same permutation set, each such table has totally $n_{s_{i.j}}$ appearances of $s_{i.j}$. Therefore, among these tables, the total number of appearances of $s_{i.j}$ is $n_{s_{i.j}} \times nt$. On the other hand, assume that one subset of the permutation set with totally nt tables actually satisfies the privacy property. Then, the number of each sensitive value associated with a tuple should satisfy $|s_{i.j}| \leq \delta \times nt$. Therefore, the total number of sensitive values for all identities is:

$$nr_i \times |s_{i.j}| \leq nr_i \times (\delta \times nt) < nr_i \times \frac{n_{s_{i.j}}}{nr_i} \times nt = n_{s_{i.j}} \times nt. \tag{3}$$

Therefore, we have $n_{s_{i.j}} \times nt < n_{s_{i.j}} \times nt$, a contradiction. Consequently, the initial assumption that there exists a subset of the permutation set satisfying the privacy property must be false. □

Since the disclosure set is computed by excluding tables from the corresponding permutation set, we immediately have the following.

COROLLARY 1. *When the privacy property is either set-monotonic or or based on the highest ratio of sensitive values, the algorithm a_{safe} has the same data utility as $a_{jump}(1)$.*

For other kinds of privacy properties, we prove that the data utility is again incomparable between a_{safe} and $a_{jump}(1)$. First, we compare their disclosure set under the 3^{rd} generalization function.

LEMMA 4. *The ds_3 under a_{safe} is a subset of that under $a_{jump}(1)$.*

PROOF. By definition, we have the following (where the superscript 0 denotes a_{safe}).

$$\begin{aligned} ds_3^1(t_0) =& per_3(t_0))/\{t|(t \in per_3(t_0)) \wedge (p(per_1(t)) \\ & \vee (p(per_2(t)) \wedge p(ds_2^1(t))))\} \end{aligned} \tag{4}$$

$$\begin{aligned} ds_3^0(t_0) &= per_3(t_0)/\{t|(t \in per_3(t_0)) \wedge (p(ds_1^0(t)) \\ & \vee p(ds_2^0(t)))\} \\ &= per_3(t_0)/\{t|(t \in per_3(t_0)) \wedge (p(per_1(t)) \\ & \vee p(ds_2^1(t)))\} \end{aligned} \tag{5}$$

Therefore, we have $ds_3^1(t_0) \supseteq ds_3^0(t_0)$. □

THEOREM 5. *The data utility of a_{safe} and $a_{jump}(1)$ is generally incomparable.*

PROOF SKETCH. Based on Lemma 4, we can construct the following two evaluation paths.

1. $a_{jump}(1) : per_1 \rightarrow per_2 \rightarrow per_3 \rightarrow p(ds_3^1) = true$
   $a_{safe} : ds_1^0(per_1) \rightarrow ds_2^0 \rightarrow p(ds_3^0) = false$
2. $a_{jump}(1) : per_1 \rightarrow per_2 \rightarrow per_3 \ldots$
   $a_{safe} : ds_1^0 \rightarrow p(ds_2^0) = true$

Clearly, the data utility of $a_{jump}(1)$ in the first case is better than that of a_{safe}, while in the second case it is worse. □

5. RELATED WORK

The micro-data disclosure problem has received significant attention lately [1, 4, 13, 19, 20]. In particular, data swapping [12, 26, 30] and cell suppression [21] both aim to protect micro-data released in census tables, but those earlier approaches cannot effectively quantify the degree of privacy. A measurement of information disclosed through tables based on the perfect secrecy notion by Shannon is given in [11]. The authors in [6] address the problem ascribed to the independence assumption made in [11]. The important notion of k-anonymity has been proposed as a model of privacy requirement [28]. It has received tremendous interest in recent years. To achieve optimal k-anonymity with the most data utility is proved to be computationally infeasible [24].

A model based on the intuition of *blending individuals in a crowd* is proposed in [29]. A personalized requirement for anonymity is studied in [33]. In [14], the authors approach the issue from a different perspective, that is, the privacy property is based on generalization of the protected data and could be customized by users. Much efforts have been made around developing efficient k-anonymity algorithms [10, 2, 3, 28, 27, 18, 8], whereas the safety of the algorithms is generally assumed. Many more advanced models are proposed to address limitations of k-anonymity. Many of these focus on the deficiency of allowing insecure groups with a small number of sensitive values, such as l-diversity [23], t-closeness [22], $alpha$-k-anonymity [32], and so on. In addition, a generic model called GBP was proposed to unify the perspective of privacy guarantees in both generalization-based publishing and view-based publishing [5].

While most existing work assume the disclosed generalization to be the only source of information available to an adversary, recent work [34] [31] shows the limitation of such an assumption. In addition to such information, the adversary may also know about the disclosure algorithm. With such extra knowledge, the adversary may deduce more information and finally compromise the privacy property. In the work of [34] [31], the authors discover the above problem and correspondingly introduce models and algorithms to address the issue. However, the method in [31] depends on a specific privacy property, whereas the one in [34] is more general, but it also incurs a high complexity. Closest to this work, a special case of the k-jump strategy is discussed in [35] where all jumps end at disclosing nothing. Our result in this paper is more general than those in [35].

In contrast to micro-data disclosure, aggregation queries are addressed in statistical databases [25, 13, 16]. The main challenge is to answer aggregation queries without allowing inferences of secret individual values. The auditing methods in [9, 7] solve this problem by checking whether each new query can be safely answered based on a history of previously answered queries. The authors of [9, 15, 17] considered the same problem in more specific settings of offline auditing and online auditing, respectively. Closest to our work, the authors of [17] considered knowledge about the decision algorithm itself. However, the solution in [17] only applies to a limited case of aggregation queries and it ignores the current state of the database in determining the safety of a query.

6. CONCLUSION

In this paper, we have proposed a novel k-jump strategy for micro-data disclosure. We have shown how a given generalization algorithm can be transformed into a large number of safe algorithms. By constructing counter-examples, we have shown that the data utility of such algorithms is generally incomparable. The practical impact of this result is that we can make a secret choice from a large family of k-jump algorithms, which is analogous to choosing a cryptographic key from a large key space, to optimize data utility based on a given table while preventing adversarial inferences (due to space limitations, more details about such optimization will be given in an extended version of this paper). It can be shown that the computational complexity of a k-jump algorithm with n generalization functions is exponential in $\frac{n}{k}$ (more details about this result and its proof will be given in the extended version), which indicates a reduction in the complexity due to k. Although the complexity is still exponential, we believe such a reduction may be meaningful considering that micro-data disclosure is typically an offline application.

Further studies will be conducted in the following directions. First, we will study other, more efficient algorithms using the same strategy of making a secret choice of public algorithms. Second, we will employ statistical methods to investigate the average-case data utility provided by different k-jump algorithms. Third, we will further investigate the issue of reusing generalization functions in an existing algorithm, which has only received limited study in the current work.

Acknowledgment

The authors thank the anonymous reviewers for their valuable comments. This material is based upon work partially supported by Natural Sciences and Engineering Research Council of Canada under Discovery Grant N01035, and by Fonds de recherche sur la nature et les technologies.

7. REFERENCES

[1] A.Dobra and S.E.Feinberg. Bounding entries in multi-way contingency tables given a set of marginal totals. In *Foundations of Statistical Inference: Proceedings of the Shoresh Conference 2000*. Springer Verlag, 2003.

[2] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Anonymizing tables. In *ICDT'05*, pages 246–258, 2005.

[3] G. Aggarwal, T. Feder, K. Kenthapadi, R. Motwani, R. Panigrahy, D. Thomas, and A. Zhu. Approximation algorithms for k-anonymity. *Journal of Privacy Technology*, November 2005.

[4] A.Slavkovic and S.E.Feinberg. Bounds for cell entries in two-way tables given conditional relative frequencies. *Privacy in Statistical Databases*, 2004.

[5] A. Deutsch. Privacy in database publishing: a bayesian perspective. In Michael Gertz and Sushil Jajodia, editors, *Handbook of Database Security: Applications and Trends*, pages 464–490. Springer, 2007.

[6] A. Deutsch and Y. Papakonstantinou. Privacy in database publishing. In *ICDT*, pages 230–245, 2005.

[7] D.P.Dobkin, A.K.Jones, and R.J.Lipton. Secure databases: Protection against user influence. *ACM TODS*, 4(1):76–96, 1979.

[8] Y. Du, T. Xia, Y. Tao, D. Zhang, and F. Zhu. On multidimensional k-anonymity with local recoding generalization. In *ICDE*, pages 1422–1424, 2007.

[9] F.Chin. Security problems on inference control for sum, max, and min queries. *J.ACM*, 33(3):451–464, 1986.

[10] G.Aggarwal, T.Feder, K.Kenthapadi, R.Motwani, R.Panigrahy, D.Thomas, and A.Zhu. k-anonymity: Algorithms and hardness. *Technical report, Stanford University*, 2004.

[11] G.Miklau and D.Suciu. A formal analysis of information disclosure in data exchange. In *SIGMOD*, pages 575–586, 2004.

[12] G.T.Duncan and S.E.Feinberg. Obtaining information while preserving privacy: A markov perturbation method for tabular data. In *Joint Statistical Meetings*. Anaheim,CA, 1997.

[13] I.P.Fellegi. On the question of statistical confidentiality. *Journal of the American Statistical Association*, 67(337):7–18, 1993.

[14] J.Byun and E.Bertino. Micro-views, or on how to protect privacy while enhancing data usability: concepts and challenges. *SIGMOD Record*, 35(1):9–13, 2006.

[15] J.Kleinberg, C.Papadimitriou, and P.Raghavan. Auditing boolean attributes. In *PODS*, pages 86–91, 2000.

[16] J.Schlorer. Identification and retrieval of personal records from a statistical bank. In *Methods Info. Med.*, pages 7–13, 1975.

[17] K.Kenthapadi, N.Mishra, and K.Nissim. Simulatable auditing. In *PODS*, pages 118–127, 2005.

[18] K.LeFevre, D.DeWitt, and R.Ramakrishnan. Incognito: Efficient fulldomain k-anonymity. In *SIGMOD*, pages 49–60, 2005.

[19] L.H.Cox. Solving confidentiality protection problems in tabulations using network optimization: A network model for cell suppression in the u.s. economic censuses. In *Proceedings of the Internatinal Seminar on Statistical Confidentiality*, 1982.

[20] L.H.Cox. New results in disclosure avoidance for tabulations. In *International Statistical Institute Proceedings*, pages 83–84, 1987.

[21] L.H.Cox. Suppression, methodology and statistical disclosure control. *J. of the American Statistical Association*, pages 377– 385, 1995.

[22] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115, 2007.

[23] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3, 2007.

[24] A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *ACM PODS*, pages 223–228, 2004.

[25] N.R.Adam and J.C.Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Comput. Surv.*, 21(4):515–556, 1989.

[26] P.Diaconis and B.Sturmfels. Algebraic algorithms for sampling from conditional distributions. *Annals of Statistics*, 26:363–397, 1995.

[27] R.J.Bayardo and R.Agrawal. Data privacy through optimal k-anonymization. In *ICDE*, pages 217–228, 2005.

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

[29] S.Chawla, C.Dwork, F.McSherry, A.Smith, and H.Wee. Toward privacy in public databases. In *Theory of Cryptography Conference*, 2005.

[30] T.Dalenius and S.Reiss. Data swapping: A technique for disclosure control. *Journal of Statistical Planning and Inference*, 6:73–85, 1982.

[31] R.C. Wong, A.W. Fu, K. Wang, and J. Pei. Minimality attack in privacy preserving data publishing. In *VLDB*, pages 543–554, 2007.

[32] R.C. Wong, J. Li, A. Fu, and K. Wang. alpha-k-anonymity: An enhanced k-anonymity model for privacy-preserving data publishing. In *KDD*, pages 754–759, 2006.

[33] X.Xiao and Y.Tao. Personalized privacy preservation. In *SIGMOD*, pages 229–240, 2006.

[34] L. Zhang, S. Jajodia, and A. Brodsky. Information disclosure under realistic assumptions: privacy versus optimality. In *CCS*, pages 573–583, 2007.

[35] L. Zhang, L. Wang, S. Jajodia, and A. Brodsky. Exclusive strategy for generalization algorithms in micro-data disclosure. In *Proceeedings of the 22nd annual IFIP WG 11.3 working conference on Data and Applications Security*, pages 190–204, 2008.