k-Jump: a Strategy to Design Publicly-Known Algorithms for Privacy Preserving Micro-Data Disclosure

Wen Ming Liu*[1], Lingyu Wang[1], Lei Zhang[2], and Shunzhi Zhu[3]

[1]Concordia University, Montreal, QC H3G 1M8, Canada

{l_wenmin,wang}@ciise.concordia.ca

[2] Google Inc., New York, NY 10011, USA

leo.zhang@gmail.com

[3] Xiamen University of Technology, Xiamen 361024, China

szzhu@xmut.edu.cn

Abstract

Data owners are expected to disclose micro-data for research, analysis, and various other purposes. 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 to an acceptable level. Typically, a disclosure algorithm evaluates potential generalization functions in a predetermined order, and then discloses the first generalization that satisfies the desired privacy property. Recent studies show that adversarial inferences using knowledge about such disclosure algorithms can usually render the algorithm unsafe. In this paper, we show that an existing unsafe algorithm can be transformed into a large family of safe algorithms, namely, k-jump algorithms. We then prove that the data utility of different k-jump algorithms is generally incomparable. The comparison of data utility is independent of utility measures and syntactic privacy models. Finally, we analyze the computational complexity of k-jump algorithms, and confirm the necessity of safe algorithms even when a secret choice is made among algorithms.

1 Introduction

The issue of preserving privacy in micro-data disclosure has attracted much attention lately [24]. Data owners, such as the Census Bureau, may need to disclose micro-data tables containing sensitive information 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 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 name, DoB, and condition are regarded as *identifier attribute*, *quasi-identifier attribute* and *sensitive attribute*, respectively. 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 is usually a combination of attributes, such as Age, Gender, and Zip Code). Nonetheless, we shall not include identifiers in the remainder of the paper for simplicity.

*Corresponding Author. Tel: +1 (514)848-2424-5642, Fax: +1 (514)848-3171, Email: l_wenmin@ciise.concordia.ca

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∼1999	flu
	cold
1960∼1979	cancer
	cancer
1940∼1959	headache
	toothache

Generalization $g_2(t_0)$

DoB	Condition
1970∼1999	flu
	cold
	cancer
1940∼1969	cancer
	headache
	toothache

Generalization $g_3(t_0)$

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

Table 1: A Micro-Data Table and Three Generalizations

To prevent such a *linking attack*, the micro-data table can be partitioned into *anonymized group* and then *generalized* to satisfy *k-anonymity* [42, 39]. 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 [36]. 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 [44, 50]. 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 attribute, such as Age in t_0, is public knowledge).We shall call this set of tables the *permutation set* with respect to the given generalization. The left-hand side of Table 2 shows two example tables in the permutation set (with the identifier Name deleted).

The permutation set would be the adversary's best guesses of the micro-data table, if the released generalization is his/her only knowledge. However, adversary may also know the generalization algorithm, and can simulate the algorithm to further exclude some invalid guesses from the permutation set. In other words, such knowledge may allow adversary to obtain a more accurate estimation of the private information than that can be obtained from the disclosed data alone. For example, assume that the adversary knows the generalization algorithm has considered $g_1(t_0)$ before it discloses $g_2(t_0)$. 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

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

multiset). From this set of tables, the adversary can infer that both Charlie and David, whose DoB are 1974 and 1962 respectively, are definitely associated with cancer. 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: The Disclosure Set of $g_2(t_0)$

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 may be different from the disclosure set of $g_2(t_0)$ shown in Table 3. The left-hand side of Table 4 shows such an example table t_3 in permutation set of $g_3(t_0)$. The disclosure set of $g_2(t_3)$ as shown in right-hand side of Table 4 is different from the disclosure set of $g_2(t_0)$. Clearly, such a recursive process is bound 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 discuss the computational complexity of such algorithms and prove that different algorithms under the k-jump strategy generally lead to incomparable data utility (which is also incomparable to that

t_3

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

Disclosure Set of $g_2(t_3)$

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

Table 4: A Table t_3 in the Permutation Set of $g_3(t_0)$ and its Corresponding Disclosure Set Under g_2

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. We also confirm that the choice of algorithms must be made among safe algorithms. Furthermore, the family of our algorithms is general and independent of the syntactic privacy property and the data utility measurement. Note that in this paper we focus on the syntactic privacy properties which has been evidenced as complementary and indispensable to the semantic notion of privacy, such as differential privacy [32, 9].

The preliminary version of this paper has previously appeared in [35]. In this paper, we have substantially improved and extended the previous version. The most significant extensions include the computational complexity analysis of k-jump algorithms by mathematical induction (Section 6), the confirmation on the necessity of safe algorithms by demonstrating that secret choice among unsafe algorithms cannot ensure the privacy (Section 7). In addition, we also further elaborate the preliminary results in [35], such as the proofs in Section 5.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 gives our model of two existing algorithms. Section 4 then introduces the k-jump strategy and discusses its properties. Section 5 presents our results on the data utility of k-jump algorithms. We analyze the computational complexity of k-jump algorithms in Section 6, and confirm that the secret choice must be made among safe algorithms such as the family of k-jump algorithms in Section 7. Section 8 concludes the paper.

2 Related Work

2.1 Privacy-Preserving Micro-Data Disclosure

The micro-data disclosure problem has received significant attention lately [18, 41, 23, 10, 11, 24, 43]. Various generalization techniques and models have been proposed to transform a micro-data table into a safe version that satisfies given privacy properties and retains enough data utility. In particular, data swapping [20, 16, 13] and cell suppression [12] 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 [38]. The authors in [15] address the problem ascribed to the independence assumption made in [38]. The important notion of k-anonymity has been proposed as a model of privacy requirement [39]. The main goal of k-anonymity is to anonymize the data such that each record owner in the resultant data is guaranteed to be indistinguishable from at least $k-1$ other record owner. Since the data owner modifies the data, some information is distorted. Therefore, it is desirable to find the modified table for k-anonymity with the minimum information loss. However, to achieve optimal k-anonymity with the most data utility is proved to be computationally

infeasible [37].

Since the introduction of k-anonymity, privacy-preserving data publishing has received tremendous interest in recent years. A model based on the intuition of *blending individuals in a crowd* is proposed in [7]. A personalized requirement for anonymity is studied in [46]. In [6], 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 [2, 3, 4, 39, 5, 30, 19], 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. For instance, l-diversity [36] requires that each equivalence class on the disclosed table should contain at least l well-represented sensitive values; t-closeness [31] requires that the distribution of a sensitive attribute in any equivalence class is close (roughly equal) to the distribution of the attribute in the whole table; (α, k)-anonymity [45] requires that the number of tuples in any equivalence class is at least k and the frequency (in fraction) of each sensitive value is at most α, where k and α are data publisher-specified thresholds. 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 [14].

2.2 The Case When Disclosure Algorithms is Publicly Known

While most existing work assume the disclosed generalization to be the only source of information available to an adversary, recent work [50] [44] 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 [50] [44], the authors discover the above problem and correspondingly introduce models and algorithms to address the issue. However, the method in [44] is still vulnerable to algorithm-based disclosure [25, 26], whereas the one in [50] is more general, but it also incurs a high complexity.

In [50], Zhang *et al.* presented a theoretical study on how an algorithm should be designed to prevent the adversary from inferring private information when the adversaries know the algorithm itself. The authors proved that it is NP-hard to compute a generalization which ensure privacy while maximizing data utility under such assumptions of adversaries' knowledge. The authors then investigate three special cases of the problem by imposing constraints on the functions and the privacy properties, and propose a polynomial-time algorithm that ensures entropy l-diversity.

Wong *et al.* in [44] showed that a minimality attack can compromise most existing generalization techniques with the aim of only a small amount of knowledge about the generalization algorithm. The authors assume that the adversaries only have one piece of knowledge that the algorithm discloses a generalization with best data utility. Under this assumption, minimality attacks can be prevented by simply disclosing sub-optimal generalizations. Unfortunately, the adversaries, equipped with knowledge of the algorithm, can still devise other types of attacks to compromise sub-optimal generalizations.

Since the problem is discovered, some work have been developed to tackle the problem in the case that $l - diversity$ is the desired privacy property [52, 47, 26, 34]. For example, [26] defines a new privacy model, namely, Algorithm-SAfe Publishing (ASAP), to capture and eliminate threats when the algorithm is publicly known. Global look-ahead and local look-ahead are then proposed to be integrated in the existing solutions to achieve ASAP. The authors also propose a post-process to enhance data utility.

To protect the individual privacy, the data owners may have different methods of anonymizing the original table. While there exist other techniques in the literature which are efficient for some privacy property, such as, anatomy [49] for l-diversity, in this paper, we focus on grouping-and-breaking, which partitions

the records into anonymized groups and break the linkage between the quasi-identifier value and sensitive value inside each group by generalization. Our proposed family of algorithms is general to handle different syntactic privacy properties and different measures of data utility. Closest to this work, a special case of the k-jump strategy is discussed in [51] where all jumps end at disclosing nothing. Our result in this paper is more general than those in [51]. It is also worth noting that we substantially extend our previous version [35] by elaborating on the proofs, analyzing the computational complexity, and confirming the necessity of safe algorithms even when making the choice secret among the algorithms.

2.3 Privacy-Preserving Macro-Data Disclosure

In contrast to micro-data disclosure, aggregation queries are addressed in statistical databases [1, 23, 40]. The main challenge is to answer aggregation queries without allowing inferences of secret individual values. The auditing methods in [8, 17] solve this problem by checking whether each new query can be safely answered based on a history of previously answered queries. The authors of [8, 29, 27] considered the same problem in more specific settings of offline auditing and online auditing, respectively.

Recently, a semantic privacy notation, differential privacy [21, 22], has been accepted as one of the strongest privacy models for answering statistic queries. Differential privacy aims to achieve the goal that the probability distribution of any disclosed information should be similar enough regardless of whether that disclosed information is obtained using the real database, or using a database without any one of the existing records. However, while the qualitative significance of the privacy parameter is well understood in the literature, the exact quantitative link between this value and the degree of privacy guarantee has received less attention. Furthermore, although differential privacy is extended to privacy preserving data publishing (PPDP) [32, 48], most existing approaches that ensure differential privacy are random noise-based and are suitable for specific types of statistical queries. It has also been evidenced that data analysis cannot replace PPDP in some situations, such as, large number of queries, diverse analysis tasks, and so on [32, 33, 9]. Moreover, Kifer *et al.* [28] disproved some popularized claims about differential privacy and showed that differential privacy cannot always guarantee the privacy in some cases. Due to these reasons, we focus on syntactic privacy properties in this paper and regard the differential privacy as future work.

3 The Model

We first introduce the basic model of micro-data table and generalization algorithm in Section 3.1. We then review two existing strategies and related concepts in Section 3.2. Table 5 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 5: The Notation Table

3.1 The Basic Model

A secret *micro-data table* (or simply a table) is 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 the identifier is not shown 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* $p(.) : 2^T \rightarrow \{true, false\}$ 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. Note that the discussion about Table 3 in Section 1 has explained why $p(.)$ should be evaluated on a set of, instead of one, tables, and we follow the widely accepted notion of generalization [39]. Given t_0 as the input to the algorithm a, either a generalization $g_i(t_0)$ will be the output and then disclosed, or $\emptyset$ will be the output indicating that nothing is disclosed. Note that returning $\emptyset$ in this paper means that the data owner decides to disclose nothing, and in this case the adversary cannot infer anything without knowing about the original table itself.

Note that in a real world generalization algorithm, a generalization function may take an implicit form, such as a cut of the taxonomy tree [44], illustrated as the dash-line in Figure 1. 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 [30], and the predetermined cuts of taxonomy trees as shown in Figure 1), 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.

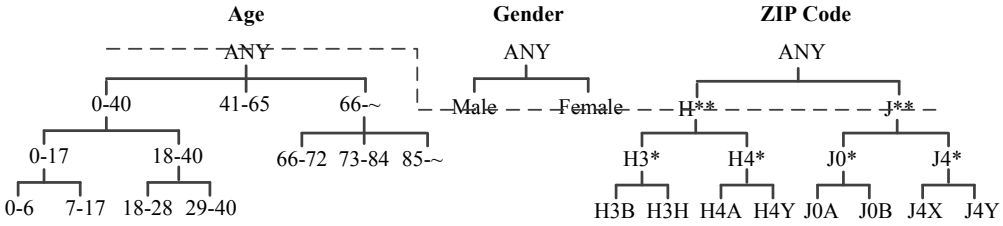

Figure 1: Taxonomy Trees Example: Age, Gender, and ZIP Code

3.2 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, 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. Note that our discussion does not depend on specific utility measures as long as the generalizations and their orders are determined based on quasi-identifiers.

Before giving the detail of *naive strategy*, we first formalizes the set of all tables in T whose generalizations, under a given function, are identical with that of a given table in Definition 1.

Definition 1 (Permutation Set)**.** *Given a micro-data table t_0, a generalization function $g_i(.)$, the* permutation

set *of* t_0 *under* $g_i(.)$ *is a function* $per(.) : G \rightarrow 2^T$*, defined by:*

$$per(g_i(t_0)) = \{t : g_i(t) = g_i(t_0)\}$$

Note that $per(g_i(t_0))$ is also written as per_i when both g_i and t_0 are clear from context. It is easily seen that, in the naive strategy, evaluating the privacy property $p(.)$ on a generalization $g_i(t_0)$ is equivalent to evaluating $p(.)$ on the permutation set $per(g_i(t_0))$.

Next we introduce the *evaluation path* in Definition 2. Informally, evaluation path represents the sequence of evaluated generalization functions.

Definition 2 (Evaluation Path)**.** *Given a micro-data table* t_0*, an algorithm composed of a sequence of generalization functions* $g_i(.)(1 \leq i \leq n)$*, the* evaluation path *of* t_0 *under the algorithm is a function* $path(.) : T \rightarrow 2^{[1,n]}$*, defined by:*

$$path(t_0) = \{i : (\textit{the algorithm will evaluate } t_0 \textit{ under } g_i) \wedge (1 \leq i \leq n)\}$$

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). With these two concepts, we can describe the above algorithm as a_{naive} shown in Table 6.

Input: Table t_0;
Output: Generalization g or $\emptyset$;
Method:
1. **Let** $path(t_0) = \emptyset$;
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** $\emptyset$;

Table 6: The Algorithm a_{naive}

Unfortunately, the naive strategy leads to an unsafe algorithm as illustrated in Section 1 (that is, an algorithm that fails to satisfy the desired privacy property). 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 $per(g_i(t_0))$. We call such a subset of $per(g_i(t_0))$ the *disclosure set*, as formally stated in Definition 3.

Definition 3 (Disclosure Set)**.** *Given a micro-data table* t_0*, an algorithm composed of a sequence of generalization functions* $s_g = (g_1, g_2, \ldots, g_n)$*, the* disclosure set *of* t_0 *under* $g_i(.)$ *is a function* $ds(.) : G \rightarrow 2^T$*, defined by:*

$$ds(t_0, g_i, s_g) = per(g_i(t_0)) \setminus \{t : i \notin path(t)\}$$

Note that $ps(t_0, g_i, s_g)$ is also written as $ps(g_i(t_0))$ or ps_i when g_i, s_g, and t_0 are clear from context. 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 modify 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 7.

Input: Table t_0;
Output: Generalization g or $\emptyset$;
Method:
1. **Let** $path(t_0) = \emptyset$;
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** $\emptyset$;

Table 7: The Algorithm a_{safe} (outline)

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.

It is worthy noting that, returning $\emptyset$ means to disclose nothing. In privacy-preserving data publishing (PPDP), we only release once by generalizing the original table. This is different from privacy-preserving data mining (PPDM), which may need to answer a sequence of queries, which may suffer from simulatable auditing. In Theorem 1, we prove that disclosure nothing is safe when the generalization algorithm is publicly-known. Basically, both disclosing nothing and adversaries' knowledge about the fact that an $\emptyset$ was produced are deemed as safe since the adversary cannot infer anything without knowing any form of the original table.

Theorem 1. *Returning $\emptyset$ is safe when generalization algorithm is publicly-known for privacy-preserving micro-data disclosure.*

Proof. First of all, we assume the given privacy property to be satisfied to an adversary who has no knowledge at all. For example, if the sensitive attribute has only two possible values in its domain (e.g., Gender), then 3-diversity can never be satisfied, even if the adversary has no knowledge at all about the original table.

We will focus on the worst case in which the adversary knows about following facts, and will discuss other cases later.

- The data owner was planning to publish data.
- The data owner did not publish anything, because the algorithm s_g returns an empty set when applied to the original micro-data table t_0.
- The adversary has a working copy of the algorithm s_g.
- The adversary does not know t_0, but knows its schema, the number of records, the quasi-identifier, the sensitive attribute, and the domain of every attribute.

In such a case, we show that the adversary's mental image about t_0 (the inferable set) will satisfy the given privacy property. The intuition behind the proof is the following. Knowing the algorithm returns an empty set, the adversary can simulate the algorithm to refine his/her mental image about t_0, since any table for which the algorithm does not return an empty set cannot be t_0. However, since the adversary does not see any released table in this case, he/she can only simulate the algorithm with every possible table (in accordance with his/her knowledge about the table schema, attribute domains, and number of records). The key is that, the final result obtained by the adversary (the inferable set) will be "symmetric" w.r.t. sensitive values (e.g. the fact that an empty set is returned by the algorithm would not give an adversary any reason to believe an identifier to be more likely associated with "cold" then "cancer", or any other sensitive value for that purpose). Therefore, the adversary's refined mental image will still have the same uniform distribution of sensitive values as initially when he/she has no knowledge at all, and hence both will satisfy the privacy property.

More formally, we first define a new concept, the symmetric set of a micro-data table, as follows. Given a micro-data table $t_0(QID, S)$, the symmetric set of t_0 is a set of tables, defined by,

$$S_{sym}(t_0) = \{\{(id, s') : (id, s) \in t_0 \wedge s' = \rho(s)\} : \rho \in \varrho(S)\},$$

where $\varrho(S)$ is the permutation set of the distinct version of S.

Then we have the following result: Given a generalization algorithm s_g, a micro-data table t_0, if $s_g(t_0) = \emptyset$, then $s_g(t') = \emptyset$ for all $t' \in S_{sym}(t_0)$. The reason is the following.

The generalization algorithm $s_g = (g_1, g_2, \ldots g_n)$ (the functions and their order) is identical for any tables in $S_{sym}(t_0)$, since the quasi-identifiers for these tables are identical. That is, given any function g_i, the quasi-identifiers in each anonymized group are same for all the tables in $S_{sym}(t_0)$, while the set of sensitive values may be different.

$\forall(t' \in S_{sym}(t_0)), path(t_0) = path(t')$. The reason is as follows.

Given any table $t_0(QID, S)$, for any $g_i(t_0) \in s_g$, any $g_{ij}(t_0) \in g_i$ (g_{ij} is an anonymized group of g_i), the corresponding $S_{ij}(t_0)$ (the multi-set of sensitive values in $g_{ij}(t_0)$), we have the following.

For any $t' \in S_{sym}(t_0)$, and the corresponding $g_{ij}(t') \in g_i(t')$ and $S_{ij}(t')$ (note that $S_{ij}(t') \subseteq S$ and $S_{ij}(t_0) \subseteq S$, but they are not necessary to be identical), there exists a one-to-one mapping $f : S_{ij}(t_0) \rightarrow S_{ij}(t')$, such that,

$$\forall(s_1, s_2 \in S_{ij}(t_0)), \begin{cases} f(s_1) = f(s_2) & \text{if } s_1 = s_2, \\ f(s_1) \neq f(s_2) & \text{if } s_1 \neq s_2, \end{cases}$$

Therefore, if the domain of a sensitive attribute has totally m different values, then the adversaries with aforementioned knowledge can only infer that each record owner in the table has $\frac{1}{m}$ probability to link to any sensitive value. In other words, the adversary has no knowledge to help him/her to distinguish between different sensitive values to be the one for each record owner, which means a uniform distribution over all values in the domain. Thus, knowing that an algorithm discloses nothing for a micro-data table will not assist adversaries in refining their mental images about the original table. In conclusion, releasing nothing is safe in this case.

Next, it is easy to see that other cases in which an adversary has less knowledge than in the above case will also be safe. For example, assume the adversary does not know that the data owner was planning to publish a table. In this case, when the data owner does not release anything, it may happen in two scenarios, which the adversary cannot distinguish:

A.) The data owner did not plan to publish anything (the result of the algorithm may or may not be an empty set),

B.) The data owner plans to publish, but the algorithm returns empty set.

In case A.), it is obvious that the adversary would not gain any additional knowledge about the original table. For case B.), we have provided the proof above. Hence, when adversary cannot distinguish between the two cases, it is easy to show that his/her mental image will still satisfy the privacy property by following similar reasoning.

For another example, if the adversary does not know the number of records in t_0, then he/she must simulate the algorithm with infinitely many tables (all possible tables with different sizes), which gives him/her an infinitely large inferable set that can be easily shown to also satisfy the privacy property. □

As discussed above, we need to evaluate the privacy property on the disclosure set (the adversaries' mental image about the original table). Informally, we call an algorithm is safe if, when that algorithm disclose a generalization for any original table, the disclosure set of that disclosed generalization with regard to the original table satisfies the desired privacy property, as formally stated in Definition 4.

Definition 4 (Safe Algorithm)**.** *Given a generalization algorithm G composed of a sequence of generalization functions $g_i(.)$ $(1 \leq i \leq n)$, its output range X composed of a subset of generalizations, the desired privacy property $p(.)$, we say G is safe if*

$$\forall(x \in X)\forall(t \in G^{-1}(x)), p(ds(t)) = true;$$

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 7, 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, which can be justified by the pseudo-code in Table 8.

Algorithm $a_{safe}(t_0, s_g, p)$	Algorithm $ds(t_0, i, s_g, p)$
Input: an original table t_0, sequence of functions $s_g = (g_1, g_2, \ldots, g_n)$, and a privacy property $p(.)$; **Output:** a generalization $g_i(1 \leq i \leq n)$ or $\emptyset$; 1: **for** $(i = 1; i <= n; i++)$ **do** 2: **if** $(p(ds(t_0, i, s_g, p)) = true)$ **then** 3: **return** $g_i(t_0)$; 4: **end if** 5: **end for** 6: **return** $\emptyset$;	**Input:** a table t_0, function i (to calculate t_0's disclosure set), sequence of functions $s_g = (g_1, g_2, \ldots, g_n)$, and a privacy property $p(.)$; **Output:** the disclosure set $ds_i(t_0)$; 1: $ds_i \leftarrow per(g_i(t_0))$; 2: **for all** $(t \in ds_i)$ **do** 3: **for** $(j = 1; j \leq i - 1; j++)$ **do** 4: **if** $(p(ds(t, j, s_g, p)) = true)$ **then** 5: $ds_i \leftarrow ds_i / \{t\}$; 6: break; 7: **end if** 8: **end for** 9: **end for** 10: **return** ds_i;

Table 8: The Algorithm a_{safe} (Detail)

On the other hand, we can see that for computing $ds(g_i(t_0))$, we must compute the truth value of the condition $i \notin path(t)$ 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 $j = 1, 2, \ldots, i - 1$. Clearly, this is an expensive process. In next section, we shall investigate a novel family of algorithms for reducing the cost.

4 k-jump Strategy

In this section, we first introduce the k-jump strategy in Section 4.1, and then discuss its properties in Section 4.2.

4.1 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 2, 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.

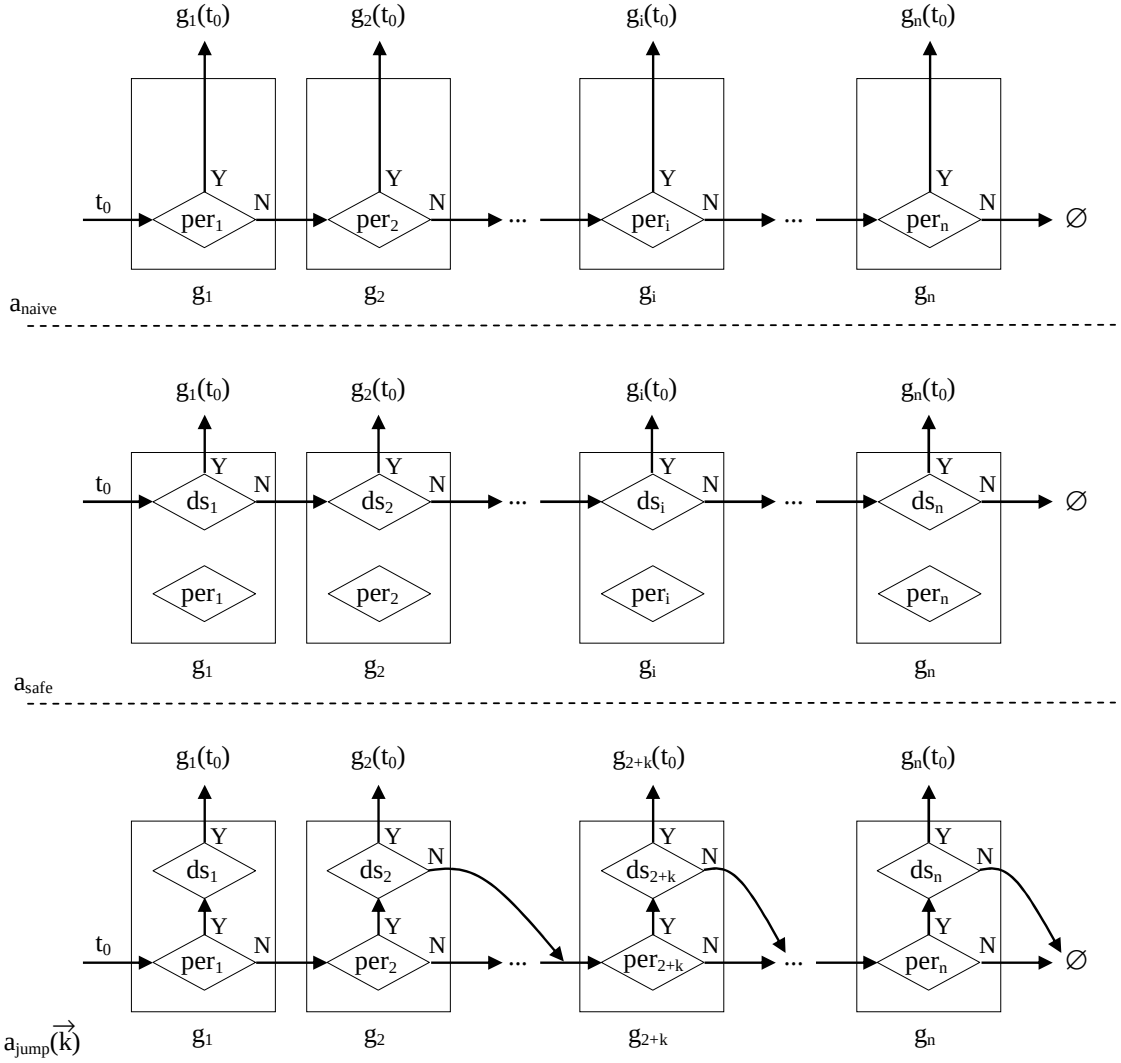

Figure 2: The Decision Process of Different Strategies

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):

1. 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.
2. If $p(per_i) = p(ds_i) = true$, both algorithms will disclose $g_i(t_0)$ and terminates.
3. The case of $p(per_i) = false \wedge p(ds_i) = true$ does not exist when privacy property is set-monotonic (which $p(S) = true$ implies $\forall S' \supseteq S\ p(S') = true$).
4. 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 2. More formally, the family of algorithms under the k-jump strategy is shown in Table 9.

Input: Table t_0, vector $\vec{k} \in [1, n]^n$;
Output: Generalization g or $\emptyset$;
Method:
1. **Let** $path(t_0) = \emptyset$;
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. **Else**
12. **Let** $i = i + 1$;
13. **Return** $\emptyset$;

Table 9: The Algorithm Family $a_{jump}(\vec{k})$

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 $path(.) : T \rightarrow 2^{[1,n]\times\{0,1\}}$ 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 13 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 $\vec{k}$.

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.

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

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

4.2.1 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 9, we can then observe that this value can be decided in line 6, right before $ds(g_i(t))$ is needed (in line 7).

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 $j = 1, 2, \ldots, i-1$. In contrast, this expensive recursive process is not always necessary under $a_{jump}(\vec{k})$.

Referring to the lower chart in Figure 2, 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 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 $\emptyset$ (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.

4.2.2 $ds(g_1(t_0))$ 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, $ds(g_1(t_0)) = per(g_1(t_0)) \setminus \{t : (1, 1) \notin path(t)\}$ 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 2). Therefore, the condition $(2, 1) \notin path(t)$ does not depend on the distance vector $\vec{k}$.

4.2.3 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 , 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 multiple 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.

5 Data Utility Comparison

In this section, we compare the data utility of different algorithms. Section 5.1 considers the family of k-jump algorithms. Section 5.2 studies the case when some generalization functions are reused in an algorithm. Section 5.3 addresses a_{safe}.

5.1 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. That is, the data utility cannot simply be ordered based on the jump distance of two algorithms. Note that, deterministically the data utility cannot be improved without the given table, and the data utility among algorithms is only comparable for the given table. In other words, here the comparison of data utility is independent of the given table, accordingly, the notation $a_{jump}(\vec{k})$ does not indicate the given table.

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 $a_{jump}(k)$.

5.1.1 $a_{jump}(1)$ vs. $a_{jump}(i)$ ($i > 1$)

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, $ds(g_3(t_0)) = per(g_3(t_0)) \setminus \{t : (3,1) \notin path(t)\}$. Obviously, for $a_{jump}(1)$, the disclosure set $ds_3^1(t_0)$ is derived from the permutation set of $g_3(t_0)$ by excluding those are disclosed under g_1 and g_2, while for $a_{jump}(i)$ ($i > 1$), the disclosure set $ds_3^i(t_0)$ is derived from the permutation set of $g_3(t_0)$ by excluding those permutation set are safe under g_1 or g_2. In other words, to remove a table t from $per(g_3(t_0))$, not only the permutation set but also disclosure set of $g_2(t)$ must satisfy the privacy property for $a_{jump}(1)$; while only permutation set of $g_2(t)$ must satisfy the privacy property for $a_{jump}(i)$ ($i > 1$), since in this case, no matter whether the disclosure set satisfies or not, $(3,1) \notin path(t)$. Formally,

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

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

from which the result follows. □

From Lemma 1, we can have the following straightforward result for the case that privacy property is set-monotonic. This result is needed for proving Theorem 2.

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 privacy property* $p(.)$ *and* $n = 3$.

Proof. As shown in Section 4.2.2, $per_1(t_0)$, $per_2(t_0)$, and $ds_2(t_0)$ are identical once the sequence of generalization functions are given. Therefore, either t_0 can be released by g_1 (or g_2) in both $a_{jump}(1)$ and $a_{jump}(i)$ ($i > 1$), or it cannot be in both of them.

For g_3, based on Lemma 1 and the definition of set-monotonic, $ds_3^i(t_0)$ satisfies privacy property only if $ds_3^1(t_0)$ satisfies. The proof is complete. □

Theorem 2. *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. Figure 3 illustrates such evaluation paths.

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 10 shows our construction for the proof. The privacy property $p(.)$ is that the highest ratio of a sensitive value in a group must be no greater than $\frac{1}{2}$. Notice that here (and in the remainder of the paper)

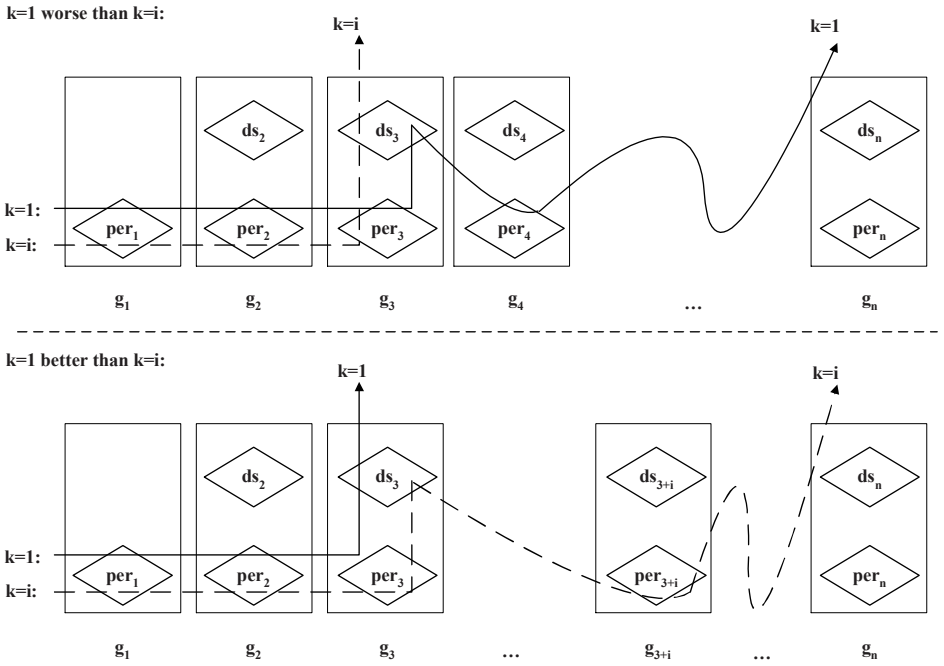

Figure 3: The Construction for $a_{jump}(1)$ and $a_{jump}(i)$ $(1 < i)$

$p(.)$ is not necessarily set-monotonic. 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 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	...
A	C_0	C_0	C_0	...
B	C_1	C_1	C_1	...
C	C_2	C_2	C_2	...
D	C_3	C_3	C_3	...
E	C_4	C_4	C_4	...
F	C_5	C_5	C_5	...
G	C_6	C_6	C_6	...
H	C_6	C_6	C_6	...
I	C_6	C_6	C_6	...
J	C_7	C_7	C_7	...
K	C_7	C_7	C_7	...
L	C_8	C_8	C_8	...
M	C_8	C_8	C_8	...
N	C_9	C_9	C_9	...
O	C_9	C_9	C_9	...

Table 10: The Case Where $a_{jump}(i)$ Has Better Utility Than $a_{jump}(1)$

5.1.2 $a_{jump}(i)$ vs. $a_{jump}(j)$ $(1 < i < j)$

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

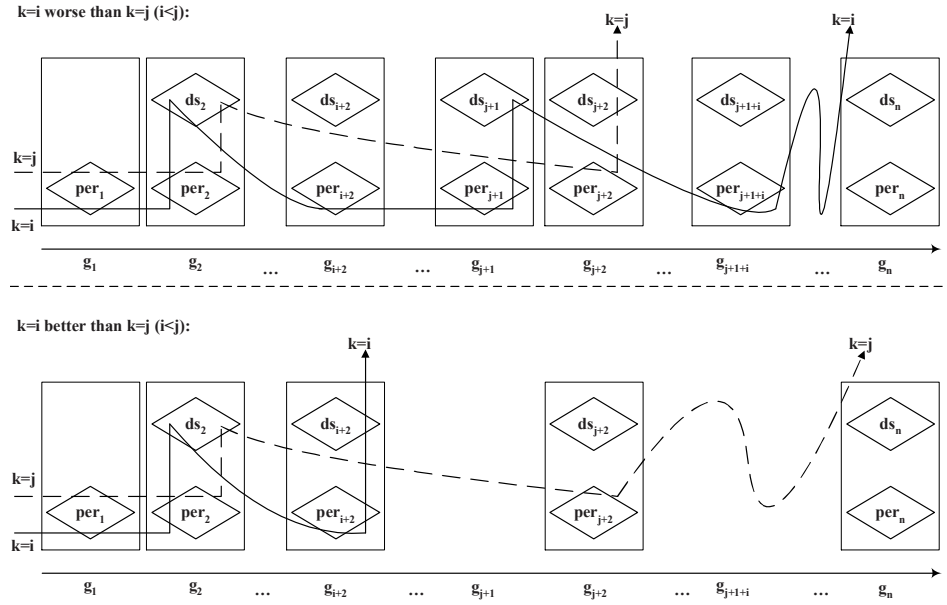

Figure 4: The Construction for $a_{jump}(i)$ and $a_{jump}(j)$ $(1 < i < j)$

Theorem 3. *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. 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 4 illustrates such constructed paths.

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

$$\begin{cases} (\text{if } \omega = 1), & p(per_\omega) = false; \\ (\text{if } \omega = 2), & p(per_\omega) = true \wedge p(ds_\omega) = false; \\ (\text{if } \omega = i+2), & p(per_\omega) = true \wedge p(ds^i_\omega) = true. \end{cases}$$

The above conditions imply that g_{i+2} will be used to disclose under $a_{jump}(i)$, while the algorithm $a_{jump}(j)$ will jump over the $(i+2)^{th}$ function to disclose under or after g_{j+2} since permutation set of g_2 satisfies privacy property while disclosure set of g_2 does not.

Secondly, we show the construction for the other case where $a_{jump}(i)$ has worse utility than $a_{jump}(j)$ $(1 < i < j)$. 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 next function, and repeat this until it reaches g_{j+1}. Since $ds_{j+1}^i(t_0)$ 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 11 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). The left (right) side of Table 11 shows the case where the data utility of $a_{jump}(i)$ is better (worse) than that of $a_{jump}(j)$ $(1 < i < j)$. Without loss of generality, we discuss the first 12 tuples in these two tables.

Firstly, we discuss the left side of Table 11. The given table, denoted by t_0, cannot be disclosed under g_1 since $p(per_1) = false$. For g_2, we have $p(per_2) = true$. The tables in ds_2 (note that $ds_2^i \equiv ds_2^j$ as shown in Section 4.2.2) satisfy that E, F, G, and H have the sensitive value C_4, S, S, and C_5, respectively. Clearly, $p(ds_2) = false$, and $g_2(t_0)$ cannot be disclosed, either. Then $a_{jump}(i)$ and $a_{jump}(j)$ will jump to evaluate under g_{i+2} and g_{j+2}, respectively.

Now we show that $a_{jump}(i)$ can be disclosed using g_{i+2}. The ds_{i+2}^i can be computed first by excluding the tables $\{t : p(per_1(t) = true\}$. The tables in ds_{i+2}^i must belong to one of the following three disjoint sets.

1. Two of A, B, and C have S. This subset has $\binom{3}{1} \times \binom{5}{1} \times 4! \times 5! = 3 \times 5! \times 5!$ tables.
2. Both D and E have S. This subset has $5! \times 5!$ tables.
3. Both F and G have S. This subset also has $5! \times 5!$ tables.

Next, $a_{jump}(i)$ will evaluate these tables using g_2. Clearly, the permutation set of each of these tables satisfies privacy property. The $a_{jump}(i)$ will further evaluate their ds_2. As discussed above, all the tables in last set cannot be disclosed under g_2. Similarly, those in second set cannot either. For the first set, all the tables which D has S are safe under g_1. In other words, the ds_2 for each table in this set satisfies that two of A, B, and C have S, which violates the privacy property. Summarily, all these tables are in $ds_{i+2}^i(t_0)$. The ratio of A, B, and C being associated with S are $\frac{2}{5}$, which is the highest ratio. Thus, $g_{i+2}^i(t_0)$ can be safely released. Besides, $a_{jump}(j)$ must disclose table t_0 under or after g_{j+2}, therefore, in this case, $a_{jump}(i)$ has better data utility than $a_{jump}(j)$.

Secondly, we discuss the right side of Table 11. Similarly, $a_{jump}(i)$ will jump to evaluate g_{i+2} while $a_{jump}(j)$ will jump to g_{j+2}. For $a_{jump}(j)$, since $p(per_{j+2}) = true$ and $p(ds_{j+2}^j) = true$ (The ratio of A, B, C, J, K, and L being associates with S is $\frac{2}{9}$ which is highest ratio), therefore, $a_{jump}(j)$ will disclose g_{j+2}. For $a_{jump}(i)$, since $p(per_{i+2}) = false$, it will move to evaluate g_{i+3} and repeat until g_{j+1} due to the same reason. Obviously, the tables in ds_{j+1}^i satisfy that both F and G have sensitive value S, which violates

the privacy property. Therefore, the algorithm $a_{jump}(i)$ jumps beyond g_{j+2} since $j+2 < j+1+i$. Clearly, with these constructions, both $a_{jump}(i)$ and $a_{jump}(j)$ will follow the desired evaluation paths as shown in Figure 4. □

(a). $a_{jump}(i)$ better than $a_{jump}(j)$

QID	g_1	g_2	...	g_{i+2}	...	g_{j+2}	...
A	C_0	C_0	...	C_0	...	C_0	...
B	C_1	C_1	...	C_1	...	C_1	...
C	C_2	C_2	...	C_2	...	C_2	...
D	C_3	C_3	...	C_3	...	C_3	...
E	C_4	C_4	...	C_4	...	C_4	...
F	S	S	...	S	...	S	...
G	S	S	...	S	...	S	...
H	C_5	C_5	...	C_5	...	C_5	...
I	C_6	C_6	...	C_6	...	C_6	...
J	C_7	C_7	...	C_7	...	C_7	...
K	C_8	C_8	...	C_8	...	C_8	...
L	C_9	C_9	...	C_9	...	C_9	...
...	...	...	...	...	...	...	...

(b). $a_{jump}(i)$ worse than $a_{jump}(j)$

QID	g_1	g_2	g_3	...	g_j	g_{j+1}	g_{j+2}	...
A	C_0	C_0	C_0	...	C_0	C_0	C_0	...
B	C_1	C_1	C_1	...	C_1	C_1	C_1	...
C	C_2	C_2	C_2	...	C_2	C_2	C_2	...
D	C_3	C_3	C_3	...	C_3	C_3	C_3	...
E	C_4	C_4	C_4	...	C_4	C_4	C_4	...
F	S	S	S	...	S	S	S	...
G	S	S	S	...	S	S	S	...
H	C_5	C_5	C_5	...	C_5	C_5	C_5	...
I	C_6	C_6	C_6	...	C_6	C_6	C_6	...
J	C_7	C_7	C_7	...	C_7	C_7	C_7	...
K	C_8	C_8	C_8	...	C_8	C_8	C_8	...
L	C_9	C_9	C_9	...	C_9	C_9	C_9	...
...	...	...	...	...	...	...	...	...

Table 11: The Data Utility Comparison Between $a_{jump}(j)$ and $a_{jump}(i)$ $(1 < i < j)$

5.1.3 $a_{jump}(\vec{k_1})$ vs. $a_{jump}(\vec{k_2})$ $(\vec{k_1} \neq \vec{k_2})$

Next, we extend the above results to the more general case in which the two algorithms $a_{jump}(\vec{k_1})$ and $a_{jump}(\vec{k_2})$ both have an n-dimensional vector as their jump distances.

Theorem 4. *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. Suppose the first element with different jump distance of $\vec{k_1}$ and $\vec{k_2}$ is the i^{th} element. Without the loss of generality, assume that $\vec{k_1}[i] < \vec{k_2}[i]$. Figure 5 illustrates such constructed paths. There are two cases as follows,

First, $\vec{k_1}[i] = 1$: Since $ds_l^{\vec{k_1}} = ds_l^{\vec{k_2}}$ for all $1 \leq l \leq i$, and $ds_{i+1}^{\vec{k_1}} \supseteq ds_{i+1}^{\vec{k_2}}$, we can construct in a similar way as in the proof of Theorem 2. Basically, we construct the following evaluation path: $per_1 \rightarrow per_2 \rightarrow \ldots \rightarrow per_i \rightarrow per_{i+1} \rightarrow ds_{i+1}$ so that in one case we have $p(ds_{i+1}^{\vec{k_1}}) = true \wedge p(ds_{i+1}^{\vec{k_2}}) = false$, whereas in the other case we have $p(ds_{i+1}^{\vec{k_1}}) = false \wedge p(ds_{i+1}^{\vec{k_2}}) = true$.

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

1. $(\exists j)((i + \vec{k_1}[i] \leq j < i + \vec{k_2}[i]) \wedge (j + \vec{k_1}[j] > i + \vec{k_2}[i]))$:
   In this sub-case, we can construct the following two evaluation paths.

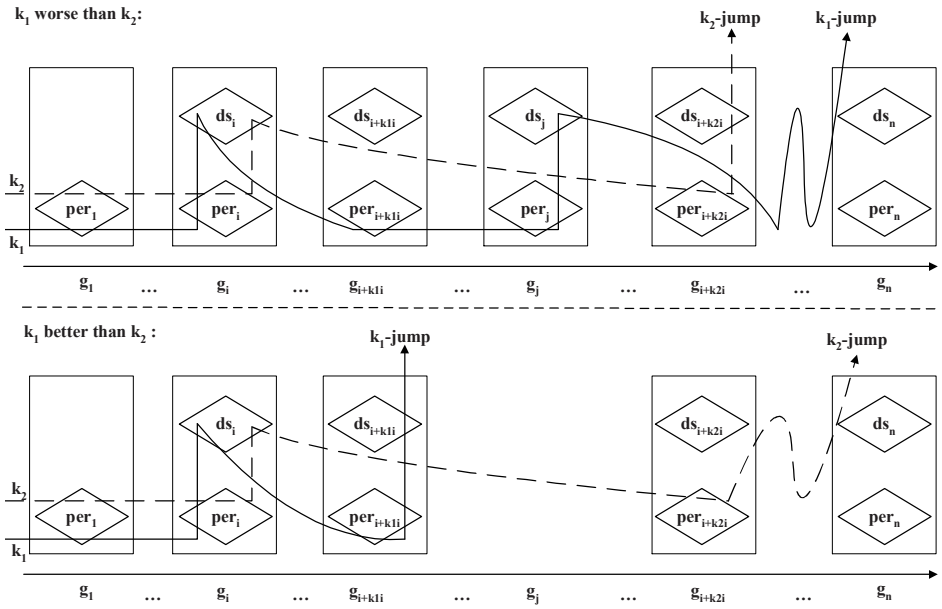

Figure 5: The Construction for $a_{jump}(\vec{k_1})$ and $a_{jump}(\vec{k_2})$ ($\vec{k_1} \neq \vec{k_2}$)

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

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

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.

2. $\neg(\exists j)((i + \vec{k_1}[i] \leq j < i + \vec{k_2}[i]) \wedge (j + \vec{k_1}[j] > i + \vec{k_2}[i]))$:
In this sub-case, $ds^{\vec{k_1}}_{i+\vec{k_2}[i]} \subseteq ds^{\vec{k_2}}_{i+\vec{k_2}[i]}$. We can reason as follows. The disclosure set of $g_{i+\vec{k_2}[i]}$ under $a_{jump}(\vec{k_2})$ is computed by excluding from its permutation set the tables which can be disclosed using g_1 and those which $p(per_2(t)) = true$; however, the disclosure set under $a_{jump}(\vec{k_1})$ needs to further exclude the tables which can be disclosed under some function g_j and $(j, 0)$ is in the evaluation path, where $(i + \vec{k_1}[i] \leq j \leq i + \vec{k_2}[i] - 1)$. Based on this result, we can construct the following evaluation paths.

(a) $a_{jump}(\vec{k_1}) : per_1 \rightarrow per_2 \rightarrow \ldots \rightarrow per_i \rightarrow ds_i^{\vec{k_1}} \rightarrow per_{i+\vec{k_1}[i]} \rightarrow \ldots \rightarrow per_{i+\vec{k_2}[i]} \rightarrow p(ds^{\vec{k_1}}_{i+\vec{k_2}[i]}) = false$
$a_{jump}(\vec{k_2}) : per_1 \rightarrow per_2 \rightarrow \ldots \rightarrow per_i \rightarrow ds_i^{\vec{k_2}} \rightarrow per_{i+\vec{k_2}[i]} \rightarrow p(ds^{\vec{k_2}}_{i+\vec{k_2}[i]}) = true$

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

$$a_{jump}(\vec{k_2}) : per_1 \rightarrow per_2 \rightarrow \ldots \rightarrow per_i \rightarrow ds_i^{\vec{k_2}} \rightarrow per_{i+\vec{k_2}[i]} \rightarrow p(ds^{\vec{k_2}}_{i+\vec{k_2}[i]}) = false$$

Clearly, the data utility of $a_{jump}(\vec{k_1})$ in the first (second) case is worse (better) than that of $a_{jump}(\vec{k_2})$.

□

5.2 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 5. *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.*

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, $ds_{2'} \subseteq ds_2$. Generally, $ds_{i'} \subseteq ds_i$ when $g_i(.)$ is reused as $g_{i'}(.)$ in a later iteration. 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. Suppose $g_i(.)$ is reused as $g_{i'}(.)$ in a later iteration of the algorithm. For any table t, since $ds_{i'}(t) \subseteq ds_i(t)$, $p(ds_{i'}(t)) = true$ implies $p(ds_i(t)) = true$ for any set-monotonic privacy property $p(.)$. Therefore, if $p(ds_{i'}(t)) = true$, the algorithm will disclose under $g_i(.)$; if $p(ds_{i'}(t)) = false$ then the algorithm will continue to the next iteration. In both cases, $g_{i'}(.)$ cannot exclude the tables from permutation set other than $g_i(.)$ can do, therefore, $g_{i'}(.)$ 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. 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. $a_1(t_0) : per_1(t_0) \rightarrow per_2(t_0) \rightarrow ds_2^1(t_0) \rightarrow p(per_3(t_0)) = false$
2. $a_2(t_0) : per_1(t_0) \rightarrow per_2(t_0) \rightarrow ds_2^1(t_0) \rightarrow per_3(t_0) \rightarrow per_{2'}(t_0) \rightarrow ds_{2'}^1(t_0) \rightarrow p(ds_{2'}^1(t_0)) = true$

Table 12 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 it by S_2.
3. C does not have C_3, and both F and G have C_3. Similarly, 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 $g_{2'}$, the tables which can be disclosed under g_1, g_2, and g_3 must be excluded from $ds_{2'}$. After excluding the tables which can be disclosed under g_1, 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. We further evaluate whether these tables can be disclosed using g_3. 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. We can reason as follows. Consider each table t in S_{1_1} under g_3. Since the tables which can be disclosed under g_1 must be excluded from $ds_3(t)$, the remaining tables in $ds_3(t)$ must be in one of the following two disjoint sets.

1. Both A and B have C_3. This subset has $3! \times 2! = 12$ tables, and is denoted by $S_{1_{1_1}}$.

2. Two of C, D and E have C_3. This subset has $(\binom{3}{1} \times \binom{2}{1}) \times \binom{3}{1} \times \binom{2}{1} = 36$ tables, and is denoted by $S_{1_{1_2}}$.

We must exclude from $ds_3(t)$ the tables which can be disclosed using g_2. The tables in $S_{1_{1_1}}$ cannot be disclosed under g_2 since their permutation sets under g_2 do not satisfy the privacy property. Furthermore, the tables in $S_{1_{1_2}}$ can be further divided into two disjoint subsets based on whether C has C_3. The tables in the case that C has C_3 cannot be disclosed using g_2 because of the same reason as those in $S_{1_{1_1}}$, while the tables in the case that C does not have C_3 cannot be disclosed using g_2 because of the similar reason as $g_2(t_0)$. In a word, all the tables in $S_{1_{1_1}}$ and $S_{1_{1_2}}$ cannot be disclosed using g_2, accordingly, these tables cannot be excluded from $ds_3(t)$. Thus, $ds_3(t) = S_{1_{1_1}} \cup S_{1_{1_2}}$. The ratio of C, D, E, F or G being associated with C_3 in $ds_3(t)$ is $\frac{1}{2}$ which is the highest ratio, accordingly, the tables in S_{1_1} can be disclosed under g_3.

Therefore, the disclosure set under the reused function $g_{2'}$ must exclude the tables in S_{1_1}, consequently, $ds_{2'} = S_{1_2} \cup S_{1_3} \cup S_2 \cup S_3$. The ratio of F and G being associated with C_3 are 0.5, which is the highest ratio. Therefore, $g_{2'}(t_0)$ can be safely disclosed. □

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 12: The Case Where Reusing Generalization Functions Improves Data Utility

5.3 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 $s_{i.j}$.

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. 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, obviously 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 $s_{i.j}$ in EC_i. Therefore, among these tables, the total number of appearances of $s_{i.j}$ in EC_i 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 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}).

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

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

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

Theorem 6. *The data utility of a_{safe} and $a_{jump}(1)$ is generally incomparable.*

Proof. 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. □

6 Computational Complexity of k-Jump Algorithms

In this section, we analyze the computational complexity of k-jump algorithms. Given a micro-data table t_0 and one of its k-jump algorithm a, let n_r be the cardinality of t_0, and n_p and n_d be the number of tables in its permutation set and disclosure set under function g_i, respectively. In the worst case, $n_p = n_r!$ and $n_d \approx n_p$, in which there is only one anonymized group and all the sensitive values are distinct.

Due to the definition of permutation set, we directly have the following result: *Given a micro-data table t_0 under a generalization function, the distribution of sensitive values corresponding to each identity in the*

permutation set is coincident with the distribution of the multiset of sensitive values in the anonymized group the identity belongs to.

Based on such coincidence, the privacy can be operated on table (when ignoring the order (identifier) of records inside each anonymized group), which is equivalent to evaluating on the permutation set (which is a set of tables). Based on these results, the running time of evaluating permutation set against privacy property reduces from $O(n_p \times n_r)$ to $O(n_r)$ for most existing privacy models, such as k-anonymity, l-diversity, and so on. Given a table t_0, let $e_p(t_0)$ and $e_d(t_0)$ be the running time of evaluating permutation set and disclosure set under a function g_i, respectively. Since generally the disclosure set does not satisfy the coincidence in the distribution, the running time of evaluating disclosure set is $O(n_d \times n_r)$. Nevertheless, for simplicity, we will consider that $O(e_p(t)) = O(1)$ and $O(e_d(t)) = O(1)$ in the following discussion. To facilitate the analysis, we elaborate the family of k-jump algorithms as shown in Table 13.

Algorithm $a_{jump}(t_0, s_g, \vec{k})$

Input: an original table t_0,
sequence of functions $s_g = (g_1, g_2, \ldots, g_n)$,
vector of jump distance $\vec{k}$,
and a privacy property $p(.)$;
Output: a generalization $g_i (1 \leq i \leq n)$ or $\emptyset$;

1: $i \leftarrow 1$;
2: **while** $(i \leq n)$ **do**
3: **if** $(p(per(g_i(t_0))_{=}true)$ **then**
4: **if** $(p(ds(t_0, i, s_g, \vec{k})) = true)$ **then**
5: **return** $g_i(t_0)$;
6: **else**
7: $i \leftarrow i + \vec{k}[i]$;
8: **end if**
9: **else**
10: $i \leftarrow i + 1$;
11: **end if**
12: **end while**
13: **return** $\emptyset$;

Algorithm $ds(t_0, i, s_g, \vec{k})$

Input: a table t_0,
function i (to calculate t_0's disclosure set),
sequence of functions $s_g = (g_1, g_2, \ldots, g_n)$,
vector of jump distances $\vec{k}$,
and a privacy property $p(.)$;
Output: the disclosure set $ds_i(t_0)$;

1: $ds_i \leftarrow per(g_i(t_0))$;
2: **for all** $(t \in ds_i)$ **do**
3: $j \leftarrow 1$;
4: **while** $(j \leq i - 1)$ **do**
5: **if** $(p(per(g_j(t))) = true)$ **then**
6: **if** $(p(ds(t, j, s_g, \vec{k})) = true)$ **then**
7: $ds_i \leftarrow ds_i / \{t\}$;
8: break;
9: **else**
10: $j \leftarrow j + \vec{k}[i]$;
11: **end if**
12: **else**
13: $j \leftarrow j + 1$;
14: **end if**
15: **end while**
16: **if** $(j > i)$ **then**
17: $ds_i \leftarrow ds_i / \{t\}$;
18: **end if**
19: **end for**
20: **return** ds_i;

Table 13: Algorithms: $a_{jump}(\vec{k})$ and $ds_i^{\vec{k}}$ With Any Given Privacy Property $p(.)$

Basically, an $a_{jump}(\vec{k})$ algorithm checks the original table t_0 against privacy property $p(.)$ under each generalization function in the given order and discloses the first generalization g_i in the sequence whose permutation set per_i and disclosure set ds_i both satisfy the desired privacy property. To determine whether a table can be disclosed under certain generalization function g_i in the algorithm, its permutation set per_i is evaluated first. If the permutation set does not satisfy the privacy property, the table will not be disclosed under this function and the algorithm moves to evaluate under next function, otherwise, its disclosure set ds_i is evaluated. If the disclosure set satisfies the privacy property, the table can be disclosed under this function g_i; otherwise, the algorithm will check the $(i + \vec{k}[i])^{th}$ generalization function in a similar way. This procedure will continue until the table is successfully disclosed under a function g_i $(1 \leq i \leq n)$ or fails

to satisfy the privacy property for all functions and nothing is disclosed.

To compute the disclosure set ds_i of t_0 under generalization function g_i, we first enumerate all possible tables by permuting each group in the generalization $g_i(t_0)$. Then, by following the algorithm, for each table t in the permutation set $per_i(t_0)$, we first assume it is the original table t_0, check under the generalization functions in sequence following the paths of the generalization algorithm, then determine whether it will not be disclosed under generalization function g_i. Such tables may fall into two different cases. First, the table can be disclosured under certain generalization function $g_j(j < i)$ before g_i; Second, the table will not be checked by the generalization function g_i, even it cannot be disclosed before g_i, which has been discussed in Section 4.2.1.

Based on the above detailed analysis of the algorithm, it can be shown that the running time of evaluating whether a given disclosure set satisfies privacy property is different from the time of deriving that disclosure set. On one hand, we consider $O(e_d(t)) = O(1)$. On the other hand, to derive $ds_i(t_0)$, we must separately evaluate each table t in $per_i(t_0)$ to determine whether it is a valid guess.

With the aforementioned discussions, we can analyze the time complexity of k-jump algorithms as follows.

Theorem 7. *Given a micro-data table t_0, a generalization algorithm of k-jump strategy that considers the sequence of generalization functions $g_1, g_2, \ldots, g_n$ in the given order and the jump-distance k, the computational complexity of such k-jump strategy is $O((max_p)^{\frac{n}{k}})$ where max_p is the maximal cardinality of possible tables in the permutation set among the functions (in the worse case, $max_p = (|t_0|)!$.*

Proof. Given a jump-vector, we prove the result by mathematical induction on n. Note that, the number of generalization functions is related to the domain size (the larger the domain size is, the more possible of generalization functions based on the methods in the literature). For simplicity, we assume the jump-vector to be jump-distance k, where k is a constant.

The Inductive Hypothesis: To compute the disclosure set of micro-data table t_0 under generalization function i in k-jump strategy, its computational complexity is $O((max_p)^{\frac{i}{k}})$.

The Base Case: When $i = 1$, it is clear that we only need to evaluate whether the permutation set satisfies the privacy property, whose running time is $e_p(t)$.

For $i = 2, 3, \ldots, 1 + k$, as mentioned before, the tables for which $p(per_j) = true$ for any $j < i$ will be removed from g_i's disclosure set. Therefore, the worst case is to evaluate all the permutation sets under each $j < i$ and evaluate both permutation set and disclosure set under function i. Thus, the running time is $O((i-1) \times max_p \times e_p(t) + e_p(t) + e_d(t)) = O((k \times max_p + 1) \times e_p(t))$, which is $O((max_p)^1)$.

The Inductive Assumption: Suppose the inductive hypothesis hold for any $j > 0$, the running time for $i \in [2 + j \times k, 1 + k + j \times k]$ is $O((max_p)^{j+1})$.

The Inductive Step: Now we show the hypothesis also holds for $j + 1$, and equivalently, for $i = 2 + (j + 1) \times k, 3 + (j+1) \times k, \ldots, 1 + k + (j+1) \times k$. Based on the assumption above, the most-time-consuming case is that for each table t in permutation set $per_i(t_0)$, there exists an evaluation of disclosure set $ps_m(t)$ where $m \in [2+j \times k, 1+k+j \times k]$. Therefore, the running time is $O(e_p(t) + \ldots + max_p \times O((max_p)^{j+1}) + e_d(t)) = O((max_p)^{j+2})$. Therefore, the assumption holds for any $j > 0$, and equivalently, for any $i \geq 2$. This concludes the proof. □

Summarily, it is shown that the computational complexity of the family of algorithms is exponential in $\frac{n}{k}$ and $|t_0|$. Similarly, k-jump algorithm requires $O(max_p)^{\frac{n}{k}})$ storage for maintaining the disclosure sets. The main reason is as follows: to compute disclosure set of an original data under a generalization function, the solution must first compute the disclosure set for each instance in each permutation set, under all the previous

generalization functions in the sequence. In turn, to compute disclosure set under any previous function for any instance, the solution must compute the disclosure set by repeating aforementioned process.

Although the worse case complexity is still exponential, this is, to the best of our knowledge, one of the first algorithms that allow users to ensure the privacy property and optimize the data utility given that the adversaries know the algorithms. Furthermore, unlike the safe algorithms discussed in [26, 47] which only work with l-diversity, the family of our algorithms $a_{jump}(\vec{k})$ is more general and independent of the privacy property and the measure of data utility.

7 Making Secret Choices of Publicly-Known Algorithms

In this section, we discuss the feasibility of protecting privacy by making a secret choice among publicly-known algorithms. The main objective of discussing secret choices in this section is not to advocate using this solution to 'get more security', but to demonstrate the fallacy of such a seemingly attractive solution, by showing that making secret choices among unsafe algorithms cannot ensure the privacy (in order to confirm the necessity of safe publicly-known algorithms, such as k-jump algorithms). Recall that we say an algorithm is safe if it can ensure the privacy property for any micro-data in the case that the adversary knows the algorithm itself, otherwise, we say it is unsafe. K-jump algorithms are safe since they evaluate the privacy property on the adversaries' mental image instead of the disclosed generalization (data), which is simutable.

7.1 Secret-Choice Strategy

From previous discussions, we know that the family of algorithms a_{jump} share two properties, namely, a large cardinality and incomparable data utility. The practical significance of this result is that we can now draw an analogy between a_{jump} and a cryptographic algorithm, with the jump distance $\vec{k}$ regarded as a cryptographic key. Instead of relying on the secrecy of an algorithm (which is security by obscurity), we can rely on the secret choice of $\vec{k}$ for protecting privacy. Table 14 shows the analogy between the secret choice among k-jump algorithm and cryptography. The main difference is how to utilize the output. For cryptography, it is to decrypt the cipher-text for the plain-text. For privacy-preserving data publishing, it is to achieve relatively statistical analysis on the disclosed data (the result should be as close to the analysis on the original data as possible) and prevent from recovering the original data.

Secret Choice	**Cryptography**
the k-jump strategy	the cipher algorithm
the jump parameter ($\vec{k}$)	the secret key
to generalize the original data	to encrypt the plain-text
the disclosed data (generalization)	the cipher-text
to analyze the disclosed data	to decrypt the cipher-text

Table 14: The Analogy Between Secret Choice Among k-Jump Algorithms and Cryptography

On the other hand, as discussed in previous sections, a safe algorithm (e.g., a_{safe} or a_{jump}) usually incur a high computational complexity, therefore, one may suggest that we can make the secret choice among unsafe but more efficient publicly-known algorithms instead of safe algorithms to reduce the computational complexity. We first formulate the secret-choice strategy.

The secret-choice strategy among a set of algorithms can take the following three stages. Given a table t_0 and the set of generalization functions $g_i(.)(1 \leq i \leq n)$, the strategy first defines a large set of generalization

algorithms (either safe or unsafe) based on the set of functions, then randomly and secretly selects one of these algorithms, and finally executes the selected algorithm to disclose the micro-data. We can thus describe the above strategy as a_{secret} shown in Table 15.

Input: Table t_0, a set of functions $g_i(.)(1 \leq i \leq n)$;
Output: Generalization g or $\emptyset$;
Method:
1. **Define** a large set of generalization algorithms $A = \{a_1, a_2, \ldots, a_m\}$ based on $g_i(i \in [1, n])$;
2. **Select** an $j \in [1, m]$ randomly for representing one of the above algorithms a_j;
3. **Return** (**Call** a_j);

Table 15: The Secret-Choice Strategy a_{secret}

There certainly exist many approaches to defining the sets of algorithms (the first stage of a_{secret}). We demonstrate the abundant possibilities through the following two examples.

First, each generalization function is slightly revised to be a generalization algorithm. That is, instead of only evaluating whether the permutation set of a micro-data table under the function satisfies the desired privacy property, such generalization algorithm further discloses the generalization or nothing. To complete the random selection, the a_{secret} will randomly select one of such algorithms and then discloses its corresponding generalization if it satisfies privacy property or nothing otherwise. Intuitively, this approach may be safe as long as the cardinality of the set of functions is sufficient large. However, such randomness will generally lead to worse data utility since usually the number of functions under which the permutation sets of a given micro-data satisfy privacy property is relatively low compared to the total number of functions. Consequently, such algorithm will disclose nothing for the micro-data with considerably high probability. Therefore, in the following discussion, without loss of generality, the randomness refers to the selection of algorithms which is not to be confused with the selection of functions in an algorithm. In other words, we assume that the algorithms sort the functions in a predetermined non-decreasing order of the data utility.

The k-jump strategy is another possible approach to defining the set of algorithms based on a given set of generalization functions. In k-jump, k is the secret choice, while all the functions appear in each algorithm and are sorted based on data utility. Given the set of functions, the one and only difference among k-jump algorithms is the jump-distance (k). As discussed above, k-jump algorithms are safe and the adversaries can at most refine their mental image to the disclosure set no matter whether they know the k. In other words, it is not necessary to hide the k among the family of k-jump algorithms. Similarly, we do not need to make a secret choice among other categories of safe algorithms. Therefore, in the remainder of this section, we will restrict the discussions on the case of secret choice among the unsafe algorithms based on predetermined order of the generalization functions. We show that secret choice among such unsafe algorithms cannot guarantee the privacy through a family of unsafe algorithms.

7.2 Subset Approach

To facilitate our discussion, we design a straightforward *subset approach* to define the set of unsafe algorithms for the first stage of a_{secret}. Given a set of generalization functions $G = \{g_1, g_2, \ldots, g_n\}$, the *subset approach* first construct all the subsets S_G of G which includes at least 2 functions. Then the naive strategy discussed in Section 3.2 is adapted on each of such subsets to embody an algorithm. That is, the functions in a subset is sorted in the non-increasing order of the data utility, and then the first function under which the permutation set of given micro-data satisfies the privacy property is disclosed; otherwise, $\emptyset$ will be the output and nothing is disclosed as shown in Table 16. We assume that the adversaries know the set of functions G since they know the released micro-data and in most cases the generalization is based on the quasi-identifier.

We also call the *secret-choice strategy* built upon subset approach *subset-choice strategy*.

Input: Set of function $G = \{g_1, g_2, \ldots, g_n\}$;
Output: Set of algorithms S_A
Method:
1. **Let** $S_A = \emptyset$;
2. **Let** $S_G = 2^G / \{\emptyset \cup \{g_i : 1 \leq i \leq n\}\}$;
3. **For** each element S_f in S_G
4. **Create** in S_A an algorithm by applying naive strategy on S_f;
5. **Return** S_A;

Table 16: The Subset Approach For Designing the Set of Unsafe Algorithms

From the adversaries' point of view, when they know the disclosed data, the subset-choice strategy (that is, the secret-choice strategy with the subset approach as its first stage), the privacy property, and the set of functions G, they may be able to validate their guesses and refine their mental image about the original data. With the knowledge about G, the adversary can know there are $\binom{|G|}{2} + \binom{|G|}{3} + \ldots + \binom{|G|}{|G|} = 2^{|G|} - |G| - 1$ possible different secret choices; With the knowledge of the disclosed data, the adversary can further know the following two facts. First, the original micro-data is in the permutation set of the disclosed generalization. Second, the generalization function corresponding to the disclosed data should be a function in the selected algorithms, and consequently the number of possible secret choices in his/her mental image is reduced to be $2^{|G|-1} - 1$. Each secret choice corresponding to an algorithm is equally likely selected. For each of these refined secret choices, the adversary first assumes that it is the true secret choice, then deduces the disclosure set for given disclosed data and corresponding naive algorithm in a similar way discussed in Section 1. Finally, the adversary refines his/her mental image to be $(2^{|G|-1} - 1)$ disclosure sets.

Based on such a mental image, the adversary may refine his knowledge about an individual's sensitive information. For example, for entropy l-diversity, the adversary can calculate the ratio of an individual being associated with a sensitive value in each disclosure set, and then average the ratio among all disclosure sets. Whenever the average ratio among the disclosure sets of an individual being associated with a sensitive value is larger than $\frac{1}{l}$, the privacy of that individual is violated. Taking k-anonymity as another example, the adversary can simply count the number of sensitive values that an individual possibly being associated with among all disclosure sets. If the resultant number for any individual is less than k, the privacy of that individual is violated.

7.3 The Safety of Subset-Choice Strategy

In the following, we show that subset-choice strategy cannot ensure the privacy property by constructing a counter-example.

Theorem 8. *Given a subset-choice strategy, there exist cases that the strategy discloses an unsafe generalization.*

Proof. One counter example, that an algorithm taking subset-choice strategy discloses a generalization while the privacy is actually violated, is sufficient to prove the theorem. Table 17 shows our construction for the proof. The left tabular shows the micro-data table t_0 whose identifiers are removed. The right tabular shows the five generalization functions in G. For clarification purposes, we intentionally keep the original value of QID. In other words, we only focus on the anonymized groups as illustrated by the horizontal lines while omitting the modification of quasi-identifiers. For example, by g_1, we partition t_0 into two

(a). The Micro-Data Table t_0

QID	S
A	C_0
B	C_0
C	C_0
D	C_1
E	C_2
F	C_3
G	C_4
H	C_5
I	C_6

(b). The set G of generalization functions for t_0

g_1		g_2		g_3		g_4		g_5	
QID	S	QID	S	QID	S	QID	S	QID	S
A	C_0	A	C_0	B	C_0	A	C_0	A	C_0
B	C_0	C	C_0	C	C_0	B	C_0	B	C_0
C	C_0	B	C_0	A	C_0	C	C_0	C	C_0
D	C_1	D	C_1	D	C_1	D	C_1	D	C_1
E	C_2	E	C_2	E	C_2	E	C_2	E	C_2
F	C_3	F	C_3	F	C_3	F	C_3	F	C_3
G	C_4	G	C_4	G	C_4	G	C_4	G	C_4
H	C_5	H	C_5	H	C_5	H	C_5	H	C_5
I	C_6	I	C_6	I	C_6	I	C_6	I	C_6

Table 17: The Counter Example for Secret Choice among Unsafe Algorithms

Possible S_G	**Probability**			**Possible** S_G	**Probability**		
	A	B	C		A	B	C
$\{g_1, g_5\}$	1	1	$\frac{1}{7}$	$\{g_1, g_2, g_5\}$	1	1	1
$\{g_2, g_5\}$	1	$\frac{1}{7}$	1	$\{g_1, g_3, g_5\}$	1	1	1
$\{g_3, g_5\}$	$\frac{1}{7}$	1	1	$\{g_1, g_4, g_5\}$	1	1	$\frac{1}{7}$
$\{g_4, g_5\}$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{2}{3}$	$\{g_2, g_3, g_5\}$	1	1	1
$\{g_1, g_2, g_3, g_5\}$	1	1	1	$\{g_2, g_4, g_5\}$	1	$\frac{1}{7}$	1
$\{g_1, g_2, g_4, g_5\}$	1	1	1	$\{g_3, g_4, g_5\}$	$\frac{1}{7}$	1	1
$\{g_1, g_3, g_4, g_5\}$	1	1	1	$\{g_1, g_2, g_3, g_4, g_5\}$	1	1	1
$\{g_2, g_3, g_4, g_5\}$	1	1	1				

Table 18: The Possible Subsets of Functions and the Corresponding Probability of A,B, and C Being Associated With C_0

anonymized groups: A and B form one anonymized group, while the others ($C - I$) form another group. In this construction, the privacy property is 2-diversity and the data utility is measured by discernibility measure (DM).

Suppose that the algorithm select subset S_G of generalization functions to be $S_G = \{g_4, g_5\}$. Obviously, the permutation set of t_0 under function g_4 does not satisfy 2-diversity, while it does so under g_5. Therefore, based on the subset-choice strategy, the algorithm discloses $g_5(t_0)$.

Unfortunately, the knowledge of G and disclosed table will enable the adversary to refine his mental image about the original micro-data, and finally violate the privacy property since the adversary can infer that the ratio of A, B and C being associated with C_0 is $\frac{272}{315} > \frac{1}{2}$.

The adversary can reason as follows. There are totally $\binom{5}{2} + \binom{5}{3} + \binom{5}{4} + \binom{5}{5} = 26$ possible secret choices of S_G. By observing the disclosed data, the adversary knows that $g_5 \in S_G$ and then refines the number of possible choices to be $\binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 15$. That is, one, two, three or all of g_1, g_2, g_3 and g_4 together with g_5 form S_G. Note that these 15 possible subsets are equally likely to be S_G. The possible subsets of functions are shown in the *possible* S_G column of Table 18.

By the data utility measurement DM, g_1, g_2, and g_3 have the same data utility which is better than that of g_4, and g_4 has better data utility than g_5. From the adversary's point of view, since g_5 is disclosed, the micro-data t_0 under any other functions in the selected S_G should violate the 2-diversity (otherwise, other

generalization should be disclosed based on the subset-choice algorithm).

Based on the disclosed data g_5, the adversary knows that only three individuals can share the same sensitive value (C_0). Therefore, the anonymized group $\{C - I\}$ in g_1, whose cardinality is 7, cannot violate 2-diversity, neither do groups $\{B, D - I\}$ in g_2, $\{A, D - I\}$ in g_3, and $\{D - I\}$ in g_4. In other words, the reason that subset approach does not disclose t_0 using function g_1, g_2, g_3 or g_4 is that the group $\{A, B\}$, $\{A, C\}$, $\{B, C\}$ or $\{A, B, C\}$ respectively does not satisfy 2-diversity. For example, suppose that $S_G = \{g_1, g_5\}$ and g_5 is disclosed, then g_1 must violate 2-diversity, therefore, both A and B should be associated with C_0, while C can be associated with any sensitive value in set $\{C_i : i \in [0, 6]\}$. The similar analysis can be applied to other possible subsets S_G and the probability of A, B, and C being associated with C_0 are shown in Table 18 when corresponding subset S_G of G is selected. Since each S_G is equally likely selected, the ratio of A being associated with C_0 is $\frac{12\times1+2\times\frac{1}{7}+1\times\frac{2}{3}}{15} = \frac{272}{315} > \frac{1}{2}$, so do B and C. In other words, once the adversary knows G, the subset-choice algorithm, subset approach, and the disclosed data g_5, he/she can infer that A, B, and C is associated with C_0 with ratio higher than $\frac{1}{2}$ even in the case that she/he does not know the secret choice (the adversary does not know which subset of G is selected). This clearly violates the privacy property. Thus we have proved the theorem. □

The counter example in the above proof is sufficient to demonstrate that secret choices made among unsafe algorithms does not always guarantee the privacy property. Therefore, safe algorithms are still necessary for preserving the privacy property.

8 Conclusion

In this paper, we have proposed a novel k-jump strategy for preserving privacy in micro-data disclosure using public algorithms. We have shown how a given unsafe 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. It has been shown that the computational complexity of a k-jump algorithm with n generalization functions is exponential in $\frac{n}{k}$ which indicates a reduction in the complexity due to k. We have also shown that making a secret choice among unsafe algorithms cannot ensure the desired privacy property which embodies the need of safe algorithms from another standpoint. Besides theoretical value, this result has the following positive impact: it identifies the root cause of high complexity in developing safe public generalization algorithms, which motivates further efforts on developing alternative methods to avoid the naturally recursive process under the popular idea in the literature.

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. Authors with Concordia University are partially supported by Natural Sciences and Engineering Research Council of Canada under Discovery Grant N01035, and Canada Graduate Scholarship. Shunzhi's research is supported in part by the National Natural Science Foundation of China under Grants No. 61373147.

References

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

[2] 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.

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

[4] 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.

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

[6] 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.

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

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

[9] C. Clifton and T. Tassa. On syntactic anonymity and differential privacy. In *ICDEW '13*, pages 88–93, 2013.

[10] 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.

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

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

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

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

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

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

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

[18] 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.

[19] 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.

[20] 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.

[21] C. Dwork. Differential privacy. In *ICALP (2)*, pages 1–12, 2006.

[22] C. Dwork, F. Mcsherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC '06*, pages 265–284, 2006.

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

[24] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys*, 42(4):14:1–14:53, June 2010.

[25] X. Jin, N. Zhang, and G. Das. Algorithm-safe privacy-preserving data publishing. In *EDBT '10*, pages 633–644, 2010.

[26] X. Jin, N. Zhang, and G. Das. Asap: Eliminating algorithm-based disclosure in privacy-preserving data publishing. *Inf. Syst.*, 36:859–880, July 2011.

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

[28] D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *SIGMOD '11*, pages 193–204, 2011.

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

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

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

[32] N. Li, W. H. Qardaji, and D. Su. Provably private data anonymization: Or, k-anonymity meets differential privacy. *CoRR*, abs/1101.2604, 2011.

[33] Ninghui Li, Wahbeh Qardaji, and Dong Su. On sampling, anonymization, and differential privacy or, k-anonymization meets differential privacy. In *ASIACCS '12*, pages 32–33, 2012.

[34] W. M. Liu and L. Wang. Privacy streamliner: a two-stage approach to improving algorithm efficiency. In *CODASPY*, pages 193–204, 2012.

[35] W. M. Liu, L. Wang, and L. Zhang. k-jump strategy for preserving privacy in micro-data disclosure. In *ICDT '10*, pages 104–115, 2010.

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

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

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

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

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

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

[42] L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.

[43] R. C. Wong and A. W. Fu. *Privacy-Preserving Data Publishing: An Overview*. Morgan and Claypool Publishers, 2010.

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

[45] 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.

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

[47] X. Xiao, Y. Tao, and N. Koudas. Transparent anonymization: Thwarting adversaries who know the algorithm. *ACM Trans. Database Syst.*, 35(2):1–48, 2010.

[48] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE '10*, pages 225–236, 2010.

[49] Xiaokui Xiao and Yufei Tao. Anatomy: Simple and effective privacy preservation. In *VLDB '06*, pages 139–150, 2006.

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

[51] L. Zhang, L. Wang, S. Jajodia, and A. Brodsky. Exclusive strategy for generalization algorithms in micro-data disclosure. In *Data and Applications Security XXII*, volume 5094 of *Lecture Notes in Computer Science*, pages 190–204. 2008.

[52] L. Zhang, L. Wang, S. Jajodia, and A. Brodsky. L-cover: Preserving diversity by anonymity. In *SDM '09*, pages 158–171, 2009.