A First Step towards Statistical Disclosure Control on Multiple Linked Tables

Kazuhiro Minami
(kminami@ism.ac.jp)
The Institute of Statistical Mathematics

Yutaka Abe
National Statistics Center

ABSTRACT

To perform statistical disclosure control (SDC) on multiple tables is a challenging task because sensitive information can be revealed from intersections of multiple tables involv-ing a common set of variables. This task is particularly dificult when each table contains a subset of the common variables because the intersection of those tables could form a subspace of any shape in the multi-dimensional domain space. To address this issue, we extend our SDC tool for solving a cell suppression problem of a two-dimensional table to support multi-dimensional ones. Our approach is to construct a single consolidated high-dimensional table from lower-dimensional multiple linked tables so that we can rep-resent the constraints of each input table in an integrated way. Our tool detects possible sensitive cells, which could be overlooked if each table is examined separately, by solving a cell suppression problem on the multi-dimensional table. In this paper, we describe an overview of the new SDC tool and show that we eliminate the risk of unintended informa-tion disclosure on sensitive cell values in our previous implementation based on a common decomposition technique for linear programming.

Keywords*: statistical disclosure control, cell suppression problem, linear programming, R.*

1. Introduction

The cell suppression problem (CSP) (Castro 2012) has been studied for many years to protect sensitive information in tabular data properly. The goal of CSP is to determine a set of suppressed cells that ensure sufficient uncertainty on the values of sensitive cells under the presence of linear relations concerning marginal sums in the table. However, many researchers (e.g., de Wolf and Hundepool (2010)) are concerned with the issue of differential attacks on multiple linked tables that are produced from the same microdata. Even if each table is verified to be safe, it is still possible for a malicious adversary to extract sensitive information by combining those multiple tables. We, therefore, need a rigorous way of verifying the safety of a set of multiple linked tables as a whole.

Previous research mainly addresses the issue of differential attacks with the modular techniques (Capobianchi and Franconi 2009; de Wolf 2002; de Wolf and Giessing 2008, 2009;

de Wolf and Hundepool 2010). The main idea of the modular approach is to decompose linked tables into smaller subtables in the hierarchy representing the inclusion relationships between spanning variables and to perform regular cell suppression on each table of a relatively small manageable size sequentially. To make the suppression patterns of the decomposed tables consistent, the suppression algorithm propagates or backtracks information on cell suppression patterns across tables. However, it is not clear that a set of locally suppressed linked tables are guaranteed to be safe as a whole because multiple cell variables in different tables could have an indirect relationship, which is derived from a chain of direct relationships among variables, restricting possible combinations of their cell values.

We, therefore, develop a rigorous method of safely suppressing linked tables extending our tool for statistical disclosure control (SDC) in *R* (Minami and Abe 2017). Our approach is to construct an integrated high-dimensional table, which includes the values of lower-dimensional tables in its marginal cells, and solve a CSP on that high-dimensional table considering its inner cells as primary suppressed cells. To realize this scheme, we make the following three contributions.

1. We extend the cell suppression algorithm to support a table of an arbitrary dimension; our previous tool only supports two-dimensional tables.

2. We relax the requirement on the availability of cell values to handle a table that contains cells whose values are missing. Since the previous algorithm requires that every cell in a table contains a value, it cannot handle a high-dimensional table produced from multiple lower dimensional tables due to the lack of inner cell values.

3. We modify the potentially unsafe constraints on the feasibility interval of a sensitive cell, which represents the range of possible cell values. Since our previous algorithm based on Castro (2012) uses the actual cell value as a reference point to enforce that constraint, an attacker knowing security parameters can infer a sensitive cell value more accurately than we intend.

The rest of the paper is organized as follows. Section 2 describes the formulation of the CSP introducing the notion of the feasibility interval, based on which we determine the safety of suppressed tabular data. We also discuss the issue of differential attacks on linked tables. In Section 3, we describe an overview of our approach for handling multiple linked tables. Section 4 shows that our new algorithm eliminates the vulnerability concerning the security constraints in our previous implementation. Section 6 discusses related work and Section 7 concludes.

2. Background

We give an overview of the cell suppression problem (CSP) introducing the notion of the feasibility interval, based on which the safety of a suppressed table is determined. For the details of the problem formulation of CSP, see Castro (2012), based on which we implement our SDC tool (Minami and Abe 2017) previously. We next describe the issue of differential attacks on multiple linked tables that are produced from the same microdata.

2.1. Cell suppression problem (CSP)

The objective of solving a CSP is to protect sensitive cell values in tabular data by determining a set of cells whose values must be suppressed. Figure 1 shows an overview of the CSP algorithm. The algorithm takes an original table and security parameters as inputs and

outputs a suppression pattern, which is a binary matrix for specifying a set of cells to be suppressed. We produce a suppressed table based on a suppression pattern, that is, if a binary value of an element in suppression pattern is 1, then the corresponding cell of the original table is suppressed; otherwise, that cell keeps the original value. The cell suppression algorithm solves a combinatorial optimization problem of minimizing information loss whose metrics is defined by an objective function (e.g., the number of suppressed cells).

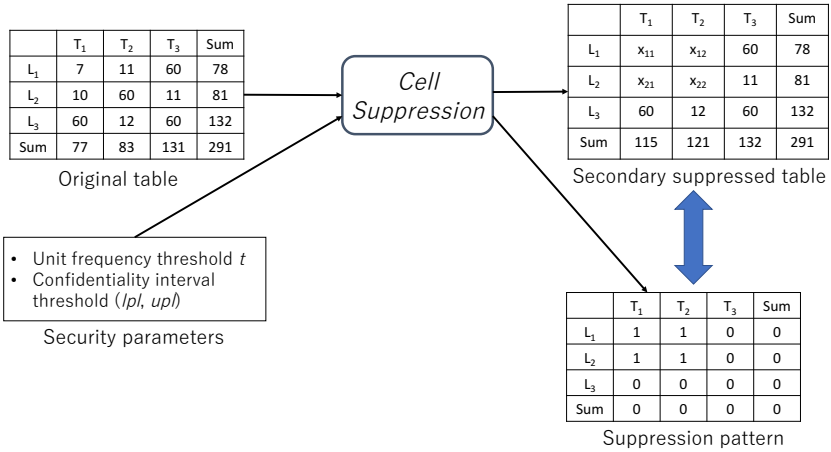

Figure 1: CSP algorithm. We represent suppressed cell values in the output table as variables x_{11}, x_{12}, x_{21}, and x_{22}

The procedure of cell suppression consists of two steps. First, we determines a set of *primary* suppressed cells by comparing each value v_i of cell i with a unit frequency threshold t. If $v_i < t$, then cell i is classified as a primary suppressed cell. However, since it is often trivial to restore the value of a primary suppressed cell when the table contains marginal cells, we need the additional procedure below. Second, we suppress non-sensitive cells additionally to protect sensitive primary suppressed cells. This process is called *secondary* suppression. When we determine a set of secondary suppressed cells, we must ensure that the value of each primary suppressed cell has enough uncertainty under the presence of linear relations concerning marginal sums.

We next introduce the notion of the *feasibility* interval to quantitatively evaluate the safety of a cell value. The feasibility interval for cell p is defined as an interval $(\underline{a_p}, \overline{a_p})$ where $\underline{a_p}$ and $\overline{a_p}$ are the minimum and maximum possible values that cell p can possibly take respectively. We compute those two values by solving two linear programming problems for the maximum and the minimum values of that cell variable with respect to the linear relations of marginal sums. Figure 2 shows the constraints on a feasibility interval of a primary suppressed cell p, which is defined by Castro (2012). Castro (2012) splits the interval into two segments at the actual value a_p and ensure that those two segments are wider than two threshold parameters lpl_p and upl_p respectively with the following conditions:

$$\underline{a_p} < a_p - lpl_p \tag{1}$$
$$a_p + upl_p < \overline{a_p} \tag{2}$$

Consequently, a feasibility interval is guaranteed to be wider than $lpl_p + upl_p$.

Example: Suppose that cell (L_1, T_1) is a primary suppressed cell and three cells (L_1, T_2), (L_2, T_1), and (L_2, T_2) are secondary suppressed cells in Figure 1. Then, the following four linear relations must hold among them.

$$x_{11} + x_{12} = 18 \quad (3)$$
$$x_{21} + x_{22} = 70 \quad (4)$$
$$x_{11} + x_{21} = 17 \quad (5)$$
$$x_{12} + x_{22} = 71 \quad (6)$$

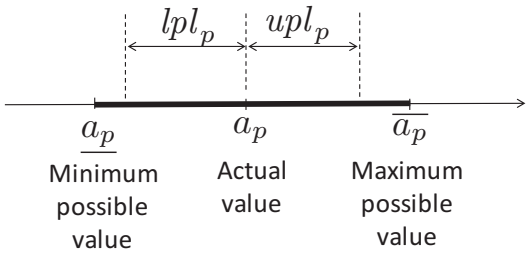

Figure 2: Constraints on a feasibility interval. We denote by variable a_p an actual value of a cell p. Two variables $\underline{a_p}$ and $\overline{a_p}$ refer to the minimum and maximum possible values for cell p, respectively. We say that an interval $(\underline{a_p}, \overline{a_p})$ is the feasibility interval of cell p.

Age	15〜19	20〜24	25〜29
#students	20	18	12

+

Age	15〜18	19〜22	23〜26
#students	18	12	12

Figure 3: An example of an differential attack on linked tables with the same type. Each table is derived from the same microdata by crossing the same categorical variables, *Age* and *the number of students*. Only the difference is that each table use the different value ranges for the variable *Age*.

if we choose *lpl* and *upl* be 5,

$$\underline{x_{11}} = 0 < 2 = x_{11} - lpl \quad (7)$$
$$x_{11} + upl = 12 < 17 = \overline{x_{11}} \quad (8)$$

satisfying the two conditions on the feasibility interval. We obtain the minimum and maximum values of variable x_{11} ($\underline{x_{11}} = 0$ and $\overline{x_{11}} = 17$) by solving two linear programming problems whose constraints are specified by equations (3)–(6).

2.2. Issue of differential attacks on linked tables

We call a set of multiple tables produced from the same microdata *linked* tables. Even if each table is suppressed properly, it is possible for an attacker to infer sensitive information by considering differences or intersections of them.

Figure 3 is a simple example of a differential attack on two linked frequency tables crossing the same categorical variables, *Age* and *the number of students*. Suppose that a threshold for the minimum frequency is 10. We then consider that each table is safe since there is no sensitive cell in the table. However, when we examine the set difference of those tables, we can easily find that the number of students at age 19 is only two, violating the minimum frequency rule. Although we can prevent such a differential attack on linked tables with the same set of categorical variables by determining the value ranges of each variable in advance. However, it is significantly more difficult to determine the safety of multiple linked tables, each of which chooses a different set of variables, as we show below.

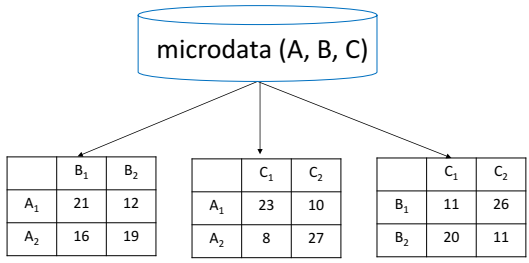

Figure 4: An example of linked tables with different sets of categorical variables.

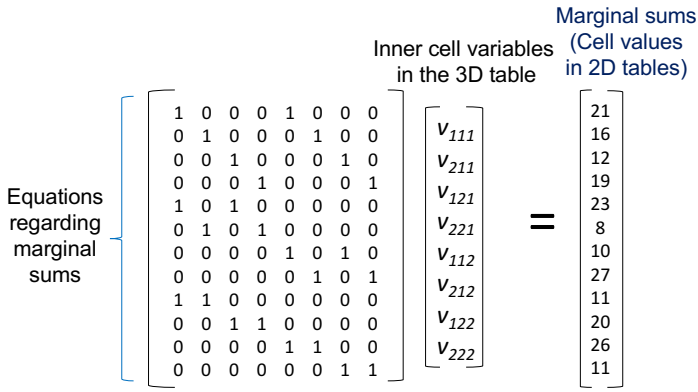

Figure 5: Linear equations of marginal sums in the three-dimensional table. Each variable v_{ijk} corresponds to a cell value at (A_i, B_j, C_k).

Figure 4 is an example of linked tables with different sets of categorical variables. Each table chooses a different pair of two categorical variables from variables A, B, and C in the microdata. Suppose that a threshold for the minimum unit frequency is 5 this time. Then, each table is determined to be safe because every cell in the table has more than five units.

However, if we reconstruct a three-dimensional table from those two-dimensional tables, we can determine the exact values of the cells in the three-dimensional underlying table by solving the linear equation in a matrix form in Figure 5. This linear equation represents a set of linear sum relations concerning marginal sums in the three dimensional table. We refer to each cell value at a position (A_i, B_j, C_k) as v_{ijk} in the equation. The vector v on the right holds the values of the marginal sums, which are populated from the two-dimensional tables in Figure 4, and the matrix A on the left indicates which three-dimensional cell should contribute to the marginal sums on the right. By solving this equation, we obtains the solution for $v^t = [11, 0, 12, 8, 10, 16, 0, 11]$. This example shows that, even if a set of low-dimensional tables are safe independently, there is a risk of revealing the value of a sensitive cell in the consolidated high-dimensional table.

3. Our approach

We need a systematic way of verifying the safety of multiple linked tables, as we discuss in Section 2.2. To address this issue, we take an approach of constructing a high-dimensional table from a list of lower-dimensional linked tables and of performing cell suppression on that

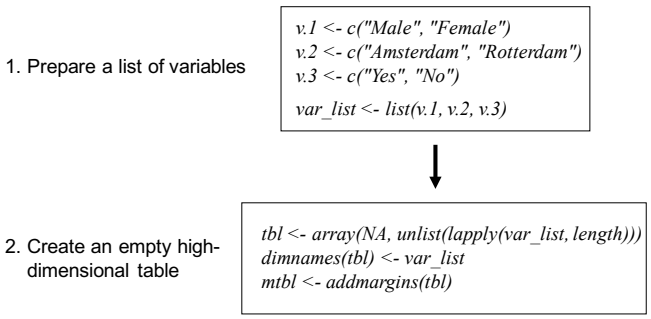

Figure 6: Snippet of R program for constructing an initial high-dimensional table.

high-dimensional table treating its inner cells as user-defined primary suppressed cells.

3.1. Construction of a high-dimensional table

We first construct an empty high-dimensional table from a list of categorical variables that appear in lower-dimensional linked tables. We assume that each linked table is labeled with variable names with the function *dimnames* in R so that we can obtain the list of variables for the high-dimensional table. We also assume that the set of values that each categorical variable can possibly take is consistently defined across different linked tables. If not, we divide the ranges of a categorical variable into smaller segments to make the set of values consistent to all linked tables. Note that we currently do not support tables with nested hierarchies or multiple levels.

Figure 6 shows a snippet of R program for constructing an initial high-dimensional table. We first make a list of variables, each of which contains a list of values. We next construct a multi-dimensional table based on the list of variables such that the number of dimensions is equal to that of the variables and the size of each dimension is equal to the number of values for that variable. We set the initial value of every cell to be NA (i.e., not available).

We next copy the values of low-dimensional linked tables into marginal cells in the high-dimensional table as shown in Figure 7. For each linked table, we take the following steps. First, we make the list of all possible combination of values. We consider each row of the list as a *local* address pointing to a particular cell of the table. We use this list to scan all the cells of the linked table. Second, we convert each local address to the *global* address pointing to the marginal cell in the high-dimensional table. We perform the address conversion as follows:

1. Copy the values in a local address to the corresponding fields of a global address.
2. For each remaining field in the global address, set the number of values for the corresponding variable plus one, which is the index for the marginal sum.

Third, we copy the value pointed by the local address in the linked table into the marginal cell of the high-dimensional table pointed by the converted global address. We repeat this task of copying a cell value for each local address in the list of possible value combinations of the linked table.

3.2. Support for a high-dimensional table with missing cell values

We need to perform cell suppression on a high-dimensional table whose inner cells do not have a value as described in Section 3.1. Our goal is to consider inner cells without a value as user-defined primary suppressed cells and to make sure that no value of inner cells are disclosed

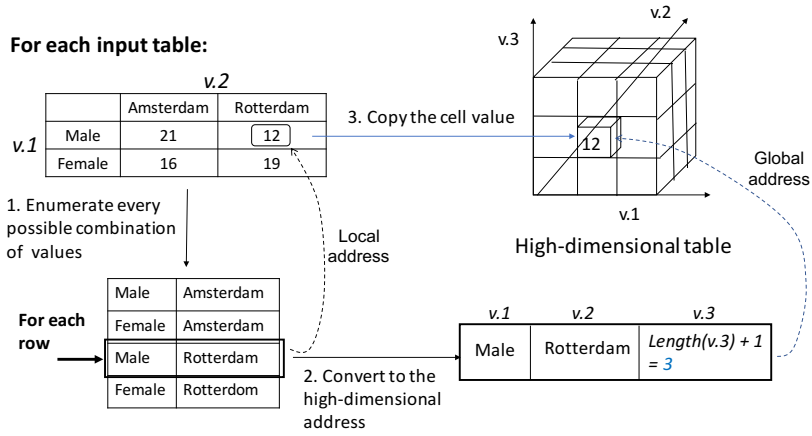

Figure 7: Procedure for populating marginal cells in a high-dimensional table.

unintentionally. However, since our previous implementation based on Castro (2012) needs all the cell values of an input table to check its safety, it cannot perform cell suppression on a high-dimensional table produced from lower-dimensional linked tables. Fortunately, the constraints on the feasibility interval by Castro (2012) in Section 2.1 is stronger than necessary. We only need to ensure that the width of a feasibility interval is greater than a given threshold δ; that is, for the feasibility interval $(\underline{a_p}, \overline{a_p})$ of cell p, where $\underline{a_p}$ and $\overline{a_p}$ are the minimum and maximum possible values that cell p can possibly take respectively, the constraint on the feasibility interval is given as follows.

$$\overline{a_p} - \underline{a_p} > \delta. \tag{9}$$

Castro (2012) achieves this property by putting constraints on the two segments divided at the actual cell value in equations (1) and (2) to efficiently find the optimal suppression pattern using the technique of the Benders decomposition (Benders 1962).

We thus implement a brute-force algorithm of examining every suppression pattern in Algorithm 1. Although this algorithm runs in exponential time, it does not require the actual values of suppressed cells to check the widths of their feasibility intervals. The algorithm takes a primary suppressed table *tbl* and a threshold width *width* for a feasibility interval as inputs and outputs a suppression pattern *sol* with the minimum number of secondary suppressed cells. Line 4 initializes a variable *sol* for the optimal suppression pattern. Line 5 computes all possible suppression patterns of length l by calling the function *enumerateBitPatterns* in Algorithm 2, which recursively appends 0 and 1 to all bit patterns of length $l-1$. Algorithm 2 maintains a list of bit strings with the data structure of a *list* in *R*.

Lines 6–21 examine every suppression pattern for a table *tbl*. For each suppression pattern *sp*, lines 7–11 compute the minimum and maximum values for each primary suppressed cell, which is done with the lpSolve package in our implementation in *R*, and lines 12–14 check whether their confidentiality intervals are greater than a threshold *width*. In lines 16–20, if the current suppression pattern *sp* is safe, then if the number of suppressed cells are fewer than the current candidate pattern *sol*, *sp* becomes the new candidate. After we examine all suppression patterns, the function returns the optimal suppression pattern *sol*.

4. Security analysis

Algorithm 1 Function $bruteForceCSP(tbl, width)$

```
1: % Input: a table: tbl, a threshold width: δ
2: % Output: the optimal suppression pattern: sol
3:
4: sol ← NULL % the optimal suppression pattern
5: all ← enumerateBitPatterns(length(tbl))
6: for each suppression pattern sp ∈ all do
7:    (A, b) ← genConstraints(tbl, sp) % where Ax = b
8:    safe ← true
9:    for each primary suppressed cell i do
10:      min ← computeMin(i, A, b)
11:      max ← computeMax(i, A, b)
12:      if max − min < δ then
13:         safe ← false
14:      end if
15:   end for
16:   if safe = true then
17:      if sum(sp) < sum(sol) then
18:         sol ← sp
19:      end if
20:   end if
21: end for
22: return sol
```

Algorithm 2 Function $enumerateBitPatterns(s)$

```
% Input: the length of a bit string: l
% Output: a set of all possible bit patterns: a

a ← list()
if l = 1 then
    a ← c(a, 0)
    a ← c(a, 1)
    return a
else
    res ← enumerateBitPatterns(l − 1)
    N ← length(res)
    p ← res
    q ← res
    for i in 1 : N do
        p[[i]] ← append(0, p[[i]])
        q[[i]] ← append(1, q[[i]])
    end for
    a ← c(a, p)
    a ← c(a, q)
end if
return a
```

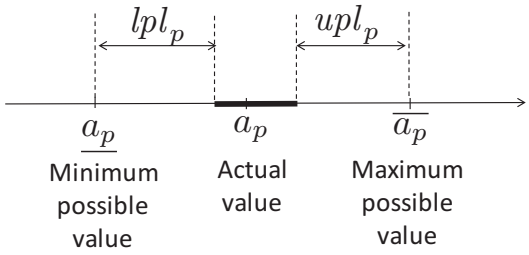

Figure 8: Effective feasibility interval. An attacker with knowledge of two security parameters (lpl_p, upl_p) can infer that the actual cell value a_p is within the interval $(\underline{a_p} + lpl_p, \overline{a_p} - upl_p)$, which is shown with the think line.

In section 3.2, we modify the constraints on a feasibility interval defined by Castro (2012) to support a high-dimensional table whose inner cells do not have assigned values. In this section, we discuss that the brute-force algorithm, which directly support the constraint in equation (9), is more secure than the algorithm of Castro (2012) because the latter uses an actual cell value as a reference point to define constraints in equations (1) and (2). We can easily derive the following inequality equation specifying the range of an actual cell value a_p.

$$\underline{a_p} + lpl_p < a_p < \overline{a_p} - upl_p \tag{10}$$

In Figure 2, we expect that the width of a feasibility interval for cell p is greater than $\delta = lpl_p + upl_p$. However, if an attacker has knowledge about two security parameters (lpl_p, upl_p) of our algorithm, he can infer that the actual width of the interval

$$(\overline{a_p} - upl_p) - (\underline{a_p} + lpl_p) = \overline{a_p} - \underline{a_p} - (upl_p - lpl_p) < \overline{a_p} - \underline{a_p}, \tag{11}$$

could be shorter than $\delta = lpl_p + upl_p$, as shown in Figure 8.

On the other hand, since the brute-force algorithm in Section 3.2 does not refer to an actual cell value a_p in its constraints on a feasibility interval, an attacker does not have any clue about narrowing down the range of possible cell values.

5. Illustrative example

We implement the cell suppression scheme in Section 3 as a set of functions in R. We show an illustrative example of handling three two-dimensional linked frequency tables below. The function *buildTable* takes three two-dimensional tables and a list of variables as inputs and outputs a consolidated three-dimensional tables. Cell values in each input table are copied to the corresponding marginal cells in the three dimensional table. Note that the values of the inner cells are not specified in the output table.

Next, we perform cell suppression on the three-dimensional frequency table with the function *suppressFT* shown in Figure 9. The function *suppressFT* takes the consolidated three-dimensional table as an input and outputs a secondary suppressed table as well as the corresponding suppression pattern. The function treats all cells with NA as primary suppressed cells and performs secondary suppression on the marginal cells. The additionally suppressed cells contain NA with a bold font in Figure 9. Notice that the four marginal cells are additionally suppressed to protect sensitive inner cells.

Our current implementation of the brute-force algorithm in Section 3 takes a few minutes to perform secondary suppression on the table in Figure 10, which is significantly slower than the algorithm based on Castro (2012) by two orders of magnitude. We, therefore, plan to explore

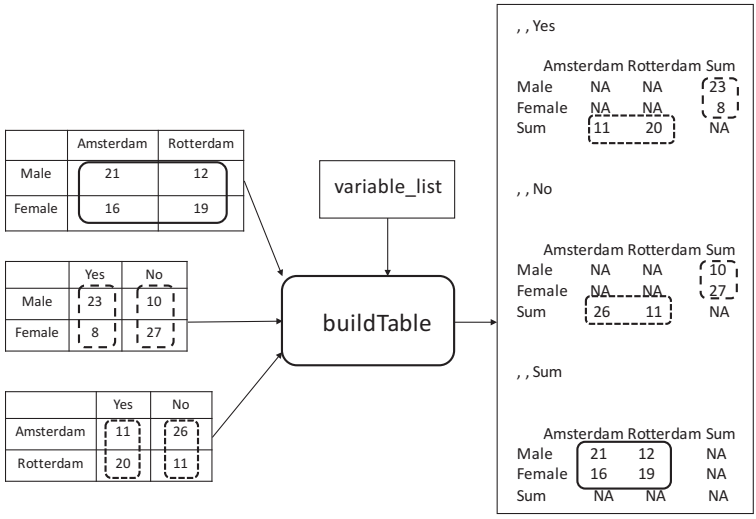

Figure 9: The function *buildTable* for building a consolidated multi-dimensional table. The rounded rectangles of different line types show to where cell values are copied from the input tables to the output table.

parallel execution of the brute-force algorithm to make our algorithm scalable to tables of larger sizes.

6. Related work

Previous research addresses the issue of differential attacks with the modular techniques (Capobianchi and Franconi 2009; de Wolf 2002; de Wolf and Giessing 2008, 2009; de Wolf and Hundepool 2010) targeting linked tables whose variables form a hierarchical structure. Statistical offices in many countries often publish such hierarchical tables. The main idea of the modular approach is to decompose linked tables involving hierarchical structures of categorical variables into a set of smaller non-hierarchical subtables so that we can perform regular cell suppression on each table of a relatively small manageable size sequentially. To make the suppression patterns of the decomposed tables consistent, the suppression algorithm propagates or backtracks information on cell suppression patterns across tables.

de Wolf (2002) proposes a top-down approach of performing secondary cell suppression on a set of sub tables that represent the original hierarchy in spanning variables from the top to the bottom sequentially to reduce the number of backtracking operations upwards. de Wolf (2007) further suggests discarding certain subtables that will not be published to reduce significant information loss due to excessive suppression. However, de Wolf (2007) also points out that there is a risk of disclosing sensitive cell values in the underlying high-dimensional table, as we point out in Section 2.2. de Wolf and Giessing (2008, 2009) additionally introduce two alternative approaches of changing the granularity of subtables to be suppressed. The linked subtables modular approach processes subtables at the same level of a hierarchy at a time while the traditional approach performs cell suppression on a set of original linked tables sequentially without decomposing them into subtables and propagates information on suppression patterns across those linked tables. Giessing and Repsilber (2002) takes a table-to-table approach, which is similar the traditional approach in (de Wolf and Giessing 2008),

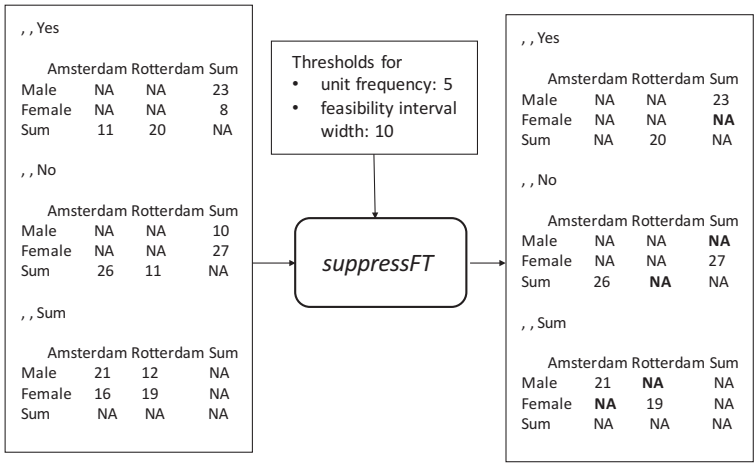

Figure 10: The function *suppressFT* for performing secondary suppression on a high-dimensional table. We show secondary suppressed cells with the value 'NA' with a bold font.

to perform suppression on overlapping tables. However, it is not clear that a set of locally suppressed linked tables are guaranteed to be safe as a whole although the modular method can reduce expensive computational cost of solving CSPs for large tables.

7. Conclusion

In this paper, we describe our new algorithm for performing cell suppression on multiple linked tables. Our approach is to construct a single consolidated high-dimensional table from lower-dimensional multiple linked tables so that we can represent the constraints of each input table in an integrated way. Since inner cell values in a consolidated high-dimensional table are not available in general, we adopt a brute-and-force algorithm for performing optimal secondary cell suppression. As the side effect of our approach, we eliminate the vulnerability of revealing sensitive cell values due to inappropriately defined constraints on a feasibility interval cell in our previous implementation. Our future work includes making the current algorithm scalable to tabular data of a realistic size by parallelizing the brute-and-force search in *R*. Also, we plan to release our SDC tool as a R package on the comprehensive R archive Network (CRAN) in the near future.

References

Benders JF (1962). "Partitioning procedures for solving mixed-variables programming problems." *Numerische Mathematik*, **4**(1), 238–252. ISSN 0945-3245. doi:10.1007/BF01386316. URL https://doi.org/10.1007/BF01386316.

Capobianchi A, Franconi L (2009). "Cell suppression in linked tables from structural business statistics using Tau-Argus 3.3.0: a conceptual framework." In *Proceedings of the New Techniques and Technologies for Statistics(NTTS)*.

Castro J (2012). "Recent advances in optimization techniques for statistical tabular data protection." *European Journal of Operational Research*, **216**(2), 257 – 269. ISSN 0377-2217.

de Wolf PP (2002). *HiTaS: A Heuristic Approach to Cell Suppression in Hierarchical Tables*, pp. 74–82. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-47804-1.

de Wolf PP (2007). "Cell Suppression in a Special Class of Linked Tables." In *Proceedings of the Joint UNECE/Eurostat Work Session on Statistical Data Confidentiality*.

de Wolf PP, Giessing S (2008). "How to Make the τ-ARGUS Modular Method Applicable to Linked Tables." In J Domingo-Ferrer, Y Saygın (eds.), *Privacy in Statistical Databases*, pp. 37–49. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-87471-3.

de Wolf PP, Giessing S (2009). "Adjusting the τ-ARGUS Modular Approach to Deal with Linked Tables." *Data Knowl. Eng.*, **68**(11), 1160–1174. ISSN 0169-023X. doi:10.1016/j.datak.2009.06.005. URL http://dx.doi.org/10.1016/j.datak.2009.06.005.

de Wolf PP, Hundepool A (2010). "Three Ways to Deal with a Set of Linked SBS Tables Using τ-argus." In J Domingo-Ferrer, E Magkos (eds.), *Privacy in Statistical Databases*, pp. 66–73. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-15838-4.

Giessing S, Repsilber D (2002). *Tools and Strategies to Protect Multiple Tables with the GHQUAR Cell Suppression Engine*, pp. 181–192. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-540-47804-1.

Minami K, Abe Y (2017). "Statistical Disclosure Control for Tabular Data in R." *Romanian Statistical Review*, (4), 67–76.