

# Linking complementary cell suppression and the software R

Bernhard Meindl\*

\* Statistics Austria, e-mail: bernhard.meindl@statistik.gv.at

## Abstract

In this contribution we will present our work on the implementation of two important algorithms for statistical disclosure control within the free and open source statistical computing system R ([1]). We present a first version of the open source R-package `sdcTable`.

The paper is organised in the following way. We start by discussing the need for the development of new software for tabular data protection. Subsequently we present details of the implementation of two important algorithms for secondary cell suppression already available in *sdcTable*. Finally we present some computational results and conclude with an outlook of future developments and refer to existing problems in this new software package.

**Keywords:** Disclosure Control, Tabular Data, Cell Suppression.

## 1 Introduction

Statistical disclosure control is a key issue in data management for national statistical offices. Data which has been collected must be returned back to the public. However, due to legal aspects it is usually not possible to release authentic data. The reason is that single statistical units such as enterprises, persons or households may be identified by an attacker and thus the information available in the released data would be known for these units. Therefore, statistical offices apply different methods to protect the data in order to satisfy legal demands and the respondents' right for confidentiality.

In this work we deal with the protection of statistical tables. A frequency table for example can be constructed by recording the number of occurrences for all cross-classifications of a typically small number of dimensional variables. It is important to know that due to the possibly hierarchical structure of the dimensional variables, linear equations exist for each statistical table. The form of these equations is typically that a marginal cell can be computed as the sum of several internal cells. Thus, a statistical table can be modeled as a data vector of cell values satisfying a set of linear constraints.

A very popular method to protect tabular data is cell suppression. Cell suppression algorithms typically consist of two main steps. In a first step it is necessary to identify sensitive cells which are marked as primary suppressions in the dataset. In the second step, values of additional cells are suppressed in order to ensure confidentiality. The process of finding adequate additional cells - also addressed to as the *secondary cell suppression problem* - by minimizing a given loss function while assuring protection belongs to the class of NP-hard problems and is therefore difficult to solve. Even though optimal algorithms do exist ([2]), there is clearly a need for heuristic approaches since it may not possible to protect large, hierarchical, high-dimensional tables which often occur in practice using optimal algorithms in acceptable time.

In  $\tau$ -Argus ([3]) - the standard software for tabular disclosure control that is currently available in version 3.3.0 - several algorithms for secondary cell suppression are available. In addition to the optimal suppression algorithm, heuristics are also implemented. These algorithms may be seen as non-global approaches since the dataset is not processed as a whole but the procedures split the data set and solve occurring suppression problems in the resulting sub-tables by either the optimal algorithm (method *HITAS* [4]) or by using the HYPERCUBE-algorithm (method *HYPERCUBE* [5]).

Unfortunately, the source code of  $\tau$ -Argus is not available under a public license. Therefore, an implementation of both HYPERCUBE and HITAS algorithms was done using the open source software system R. In this contribution we describe the implementation, underlying concepts and discuss example results of the new R-package `sdcTable`. We also address the issue of implementation problems, missing pieces, the clear need for further improvements in our application as well as the advantages and flexibility of our implementation in this work.

## 2 Protecting sensitive cells using R

After defining the secondary cell suppression problem in (2.1), we show why it is important to standardize input data and how this standardization can be achieved. Finally we give details on the practical implementation of two important algorithms for data protection that are already available in `sdcTable`.

### 2.1 The secondary cell suppression problem

Cell suppression is an important method for protection sensitive information in tabular data. The procedure consists of two main steps:

- identify cells that need to be protected (primary suppression).

- suppress additional cells in an *optimal* way that primary suppressed cells are protected *adequately*.

The two important terms are obviously *optimal* and *adequately*. Optimality is often defined with regard to measures of information loss. This means that only the absolutely necessary number of additional cells guaranteeing the required protection level for primary suppressed cells should be selected as part of the suppression scheme.

In `sdcTable` two important (heuristic) algorithms to solve the secondary cell suppression problem are implemented. Details on the algorithms are given in section (2.3) and in section (2.4).

## 2.2 Standardizing the input

The first important step of the implementation of heuristic algorithms for the second cell suppression problem is to define how hierarchical, tabular data needs to be organized in order to simplify further processing.

Since R is an object-oriented programming language, we can define that tabular data that should be used as an input object for protection algorithms need to be an object of class *fullData*. An object of class *fullData* is a list whose elements contain all the information that is needed by the protection algorithms to suppress complementary cells. Objects of class *fullData* can be generated in `sdcTable` using function `createFullData()`. In the following sections we show how to generate a suitable input object starting from a given data set and how to protect this table using R and package `sdcTable`.

First of all, we have a look at the example data containing all possible (sub)total listed in Table (1). Both variables that define the table are hierarchical. Figure 1 shows the structure of dimensional variable *REGION*.

One can clearly see that this variable consists of a total of 3 levels. The top level is the Total which consists of main regions A and B in the second level. The third level consists of sub-regions. Main Region A consists of 10 sub-regions while main region B consists of only two sub-regions. The values in parenthesis in Figure 1) show the characteristics of the standardized vector for dimensional variable *REGION*. Details on how to derive this standardized vector are given later on.

### position of dimensional variables:

An additional object needed as input object for function `createFullData()` is a vector specifying the position of the dimensional variables in the dataset.

SEX	REGION	VAL	SEX	REGION	VAL
Total	Total	160	Total	Main Region B	17
Male	Total	80	Male	Main Region B	5
Female	Total	80	Female	Main Region B	12
Total	Main Region A	124	Total	sub-Region B1	10
Male	Main Region A	64	Male	sub-Region B1	3
Female	Main Region A	60	Female	sub-Region B1	7
Total	sub-Region A1	11	Total	sub-Region B2	7
Male	sub-Region A1	4	Male	sub-Region B2	2
Female	sub-Region A1	7	Female	sub-Region B2	5
...	...	...			
Total	sub-Region A10	19			
Male	sub-Region A10	11			
Female	sub-Region A10	8			

Table 1: (parts) of the complete dataset.

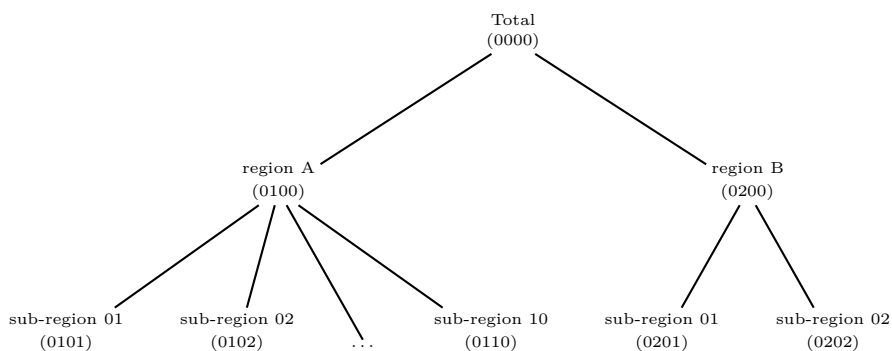


Figure 1: regional hierarchical dimensional variable.

Given the example, code-listing (1) shows the creation of object *indexDimVars* with the dimensional variables being located in column one and two.

#### hierarchical structure of dimensional variables:

The user also needs to specify the hierarchical structure of the dimensional variables by creating a list featuring as many elements as dimensional variables. Each list-element is a vector containing integer values that are the number of digits of the total number of characteristics within this level.

For example, the dimensional variable *REGION* consists of three levels, the grand total, main regions as well as sub-regions within the main regions. The next part is then to fill the elements of the hierarchical vector. For each dimensional variable the following steps need to be applied:

- count the number of levels of a dimensional variable.

- in each level  $i$ : count the maximum number of characteristics and specify the digits of this number.
- use this value as  $i^{\text{th}}$  element of the vector specifying the hierarchical structure of the current dimensional variable.

Code-Listing (1) shows the required code in R to specify object *structDimVars* with respect to the example.

```
# position of dimensional variables
indexDimVars <- c(1,2)

structDimVars <- list()           # hierarchical structure
structDimVars[[1]] <- c(1,1)     # SEX
structDimVars[[2]] <- c(1,1,2)   # REGION
```

Listing 1: position and structure of hierarchical variables

The hierarchical structure of a dimensional variable shows, how many digits are needed for each level when standardizing the characteristics of the variables. This procedure is now described in detail.

### Recoding of dimensional variables:

The next important step is to standardize the dimensional variables *SEX* and *REGION* by recoding them. The required length of the standardized vector for any characteristic of a dimensional variable is given by the the summing up the vectors in object *structDimVars*. Therefore, the length of the standardized vector for the dimensional variables *SEX* and *REGION* is 2 (1+1) and 4 (1+1+2), respectively.

```
SEX <- as.character(data$SEX)
SEX[SEX %in% "Total"] <- "00"
SEX[SEX %in% "Male"] <- "01"
SEX[SEX %in% "Female"] <- "02"; dat$SEX <- factor(SEX)

REGION <- as.character(data$REGION)
REGION[REGION %in% "Total"] <- "0000"
REGION[REGION %in% "Main Region A"] <- "0100"
for (i in 1:10) {
  ind <- which(REGION %in% paste("sub-Region A",i, sep=""))
  ifelse(nchar(i) == 1, REGION[ind] <- paste("010",i,sep=""),
  REGION[ind] <- paste("01",i,sep=""))
}
REGION[REGION %in% "Main Region B"] <- "0200"
REGION[REGION %in% "sub-Region B1"] <- "0201"
REGION[REGION %in% "sub-Region B2"] <- "0202"
dat$REGION <- factor(REGION)
```

Listing 2: standardizing dimensional variables by recoding.

The standardized dimensional vector for characteristics in the second level (eg. main regions A and B) is constructed by adding numbers starting from 1 to the total number of characteristics in this (sub)level after a leading zero which refers to the total value. Therefore, main-region A would be indexed by '0100' and main-region B by '0200'. To construct indices for characteristics of the third level (eg. sub-Region A1, . . . , sub-Region A10), the index of the top level (main region A, '01XX') would be used and extended by adding numbers from 1 to the total number of characteristics at the remaining 2 digits. Thus, sub-Region A1 would be indexed by '0101', the corresponding index for sub-region A1 would be '0110'. In a similar way, indices for sub-region B1 and sub-region B2 need to be constructed, leading to '0201' or rather '0202'. Unfortunately, user invention is needed here and this process cannot be done in an automated way.

#### **a minimal data set:**

The next step is about specifying a minimal input dataset by removing any (sub)totals in the data set that could be calculated from other cells. In a sense we are looking for all characteristics that are needed to calculate the complete table without losing any information.

Thus, considering dimensional variable *sex* it is obvious that characteristic "00" (the total) is not necessary and can be removed because values for males and females within a given regional characteristic sum up to the total value. Characteristics "0000", "0100" and "0200" for dimensional variable *REGION* are also redundant because they can be calculated from inner cells. Code listing (3) shows the necessary code to to remove all redundant cells from the original data set.

```
ind <- which(!SEX %in% "00" &
             !REGION %in% c("0000", "0100", "0200"))
minDat <- dat[ind,]
```

Listing 3: removing (sub)totals from dataset.

#### **creating an object of class *fullData*:**

Having all necessary parameters specified, an object of class *fullData* can now be calculated using function `createFullData()` which can finally be used as an input object for the protection procedure `protectTable()`.

It is also possible to demand primary suppression by simple frequency rules by setting additional options in function `createFullData()` as in code-listing (4) in which all cell values  $\leq 3$  would be marked as primary suppressed.

```

# creating the dataset needed for protection procedure
fullData <- createFullData (minDat, indexDimVars, structDimVars,
                           suppVals=TRUE, suppLimit=3)
res1 <- protectTable(fullData, method="HYPERCUBE"); summary(res1)
res2 <- protectTable(fullData, method="HITAS"); summary(res2)

```

Listing 4: creating a complete dataset.

Function `createFullData()` is applied to generate an object of class *fullData* which can be used as input parameter for function `protectTable()`. By specifying the method in `protectTable()` one can choose the protection algorithm which should be used to protect the tabular data.

### 2.3 Implementation of the HYPERCUBE algorithm:

The HYPERCUBE algorithm - proposed by [5] - is a heuristic procedure that makes use of multidimensional cuboids to protect primary suppressed values. In our implementation the complete dataset is divided into sub tables which consist of only two levels in each dimensional variable (e.g. a (sub)total and the corresponding characteristics from which the (sub)total can be constructed). Within these sub-tables, for each primary suppressed value an *optimal* cuboid that protects the primary suppressed value is found and all values of the optimal cuboid that are not already suppressed are marked as additional suppressions. An *optimal* cuboid is chosen in a way that either the total number of additional suppressions or the sum of additionally suppressed values (or their logs) is minimized which can be specified by the user. Since due to the (possible) hierarchical structure of the complete dataset secondary suppressions may also exist in other sub-tables, additional runs of the algorithms are needed to ensure that it is not possible to recalculate secondary suppressed cells exactly. Therefore, all additional suppressions are marked and checked in the following runs. The algorithm runs in a loop until no further additional suppressions need to be added.

In our implementation, computationally time-consuming functions have been implemented in standard C-programming language to reduce running time of the algorithm. We have implemented interval protection for primary suppressed values as well. The magnitude of the required interval protection may be chosen by the user when calling the protection procedure.

We also took special care to protect cells that only contain one respondent. During the process of finding an optimal suppression scheme, schemes that do not feature single cells are preferred. If however only schemes that possess single cells exist that satisfy the required interval protection for a suppressed value, an additional cuboid is selected in order to protect the single cell

value. Furthermore it is possible for the user to specify if cells containing zeros should be allowed to be part of a possible suppression scheme.

## 2.4 Implementation of the HITAS algorithm:

The heuristic algorithm HITAS for secondary cell suppression in hierarchical data structures was proposed by [4]. The key idea is to split the entire data structure into sub tables in a specific way using a top-down approach and to protect primary suppressed values within the resulting sub tables using the optimal suppression algorithm proposed by [2].

The key idea of the optimal suppression algorithm is to formulate a (mixed) integer linear problem which for each sensible cell finds a suppression scheme that protects the sensible cell adequately according to the chosen protection levels by minimizing the loss of information. The loss of information is taken into account by the objective function of the problem. The chosen suppression minimizes the objective function which is of form  $\sum_{i=1}^n w_i \cdot x_i$ .  $x_i$  are binary variables that are 1 only if cell with index number  $i$  is part of the suppression scheme and  $w_i$  are costs which occur if the corresponding cell is suppressed.

In our implementation the Mixed Integer Linear Programming (MILP) solver *lpsolve* [6] is used for solving the resulting mixed linear integer optimization problems. Important reasons for this choice were that the solver is free and open source software as well as the fact that the solver can be conveniently called inside R.

## 2.5 Protecting sensitive cells in a statistical table:

Protecting a table is usually done by calling function `protectTable()` using two mandatory parameters and several optional parameters which are used to alter the default values for the protection method chosen. As input parameters one needs to specify an object of class *fullData* and a method that should be used to protect primary suppressed values in the dataset by suppression additional cells. Currently only *HYPERCUBE* and *HITAS* can be chosen, however due to the design and the standardization of input objects it should be possible to add further suppression algorithms in future.

A safe table can finally be generated by calling `protectTable()`. The resulting output object of this procedure is an object of class *safeTable*. When no additional parameters are specified, the default values for the chosen protection procedure are used while protecting the dataset.

Analyzing the results of the protection procedure can be done by applying

a summary method to objects of class *safeTable*. Using the available summary function one is presented with some statistics about the suppression process such as the algorithm that was used to protect the data set, the total running time of the procedure, the total number of additional, secondary suppressions as well as the necessary runs of the algorithm to protect secondary suppressions as well. Furthermore, the protected data set is returned to the user too.

### 3 Computational results:

To further illustrate the process of protecting hierarchical data using our implementation in R, we give the following example. Consider a data set that consists of 3 hierarchical variables that consist of a total of 4, 3 and 3 hierarchies, respectively. The number of characteristics without (sub)-totals is 47 for the first, 19 for the second and 15 for the third dimensional variable. Generating the complete data structure resulted in a dataset consisting of a total of 30096 cells when these hierarchical variables are used. We populated the cells using random values drawn from a poisson distribution with parameter  $\lambda=7$ .

The next step is to identify primary suppressed cells. We chose a simple minimal frequency rule and therefore marked all non-zero cells as primary suppressed that had a cell value less or equal to 3. This resulted in a total of 293 primary suppressions.

When the data set was protected using the heuristic *HYPERCUBE* procedure a total of 1323 cells needed to be marked as secondary suppressions (4.4%). The total running time of the algorithm was 595 seconds on a standard personal personal computer.

Applying the *HITAS* algorithm to protect the data set resulted in a total of 1104 (3.67%) additional suppressions to protect the sensitive cells. However, it took about an hour for *HITAS* to protect the hierarchical table which is much longer than the running time of *HYPERCUBE*. This means of course that work still needs to be done to reduce the running time of the *HITAS* algorithm in our implementation.

### 4 Conclusion:

In this work we presented an early implementation of some methods for disclosure control for tabular data available in the new developed R-package *sdTable*. Even though two important algorithms are already implemented, there is still a lot of work left in the future to improve the code and to make

`sdcTable` useable for real world problems. Especially it will be necessary to review and to optimize the code and the implementation of the algorithms for possible optimizations.

However, it should also be noted that it will be important to implement more algorithms to protect tabular data in the future. In addition to the two algorithms for secondary cell suppression, algorithms for simple, random and controlled rounding as well as a function to perform controlled tabular adjustment of tabular data have been implemented in `sdcTable`. However, the implementation of these algorithms is working for specific (sub)tables but it is not yet possible to deal with hierarchical tables in a heuristic way by splitting the tabular structure into subtables and individually protecting them. It should also be noted that the implementation of additional protection methods can make use of existing and already implemented data structures.

`sdcTable` is the first attempt to link statistical disclosure control for tabular data and the open source statistical software system R. Due to the fact that the source code is freely available and everyone is not only allowed to but also encouraged to modify and to improve the code base as well as to implement new methods, `sdcTable` may in future become a standard tool for protecting tabular data in real life applications.

## References

- [1] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.
- [2] Matteo Fischetti and Juan José Salazar. Solving the cell suppression problem on tabular data with linear constraints. *Manage. Sci.*, 47(7):1008–1027, 2001.
- [3] A. Hundepool, R. Ramaswamy, de Wolf P-P., L. Franconi, S. Giessing, D. Repsilber, J.J. Salazar, C. Castro, G. Merola, and P. Lowthian, 2008.
- [4] Peter-Paul de Wolf. Hitas: A heuristic approach to cell suppression in hierarchical tables. In *Inference Control in Statistical Databases, From Theory to Practice*, pages 74–82, London, UK, 2002. Springer-Verlag.
- [5] Dietz Repsilber. Sicherung persönlicher Angaben in Tabellendaten. In *Statistische Analysen und Studien Nordrhein-Westfalen*. Landesamt für Datenverarbeitung und Statistik NRW, 2002.
- [6] `lp_solve`, a Mixed Integer Linear Programming (MILP) solver. Available online at <http://lpsolve.sourceforge.net/>, last accessed: 12/16/08.