

## **Considerations to deal with the frozen cell problem in Tau-Argus Modular.**

Sarah Giessing (Destatis)

[Sarah.Giessing@destatis.de](mailto:Sarah.Giessing@destatis.de)

### ***Abstract***

The package  $\tau$  Argus is a widely used EU-funded Open Source tool for disclosure control in tabular data. For secondary cell suppression it offers several alternative algorithms, one of which, referred to as “Modular”, is recommended best practice for application to large, complex structured sets of hierarchical and linked tables. Meanwhile, several NSIs and Eurostat have invested into integration of the software into production environments. The issue we mainly address in this paper is an important use case still lacking a sound methodological approach: The usual production process for a statistic involves dissemination of a major set of pre-planned tables as soon as data are available for this production step. Typically, later on, there will be requests from users for specific additional tabulations, often linked to the tables of the already disseminated ones. When protecting such additional tables one has to take into account the suppression pattern of the disseminated tables: Suppressed cells must be suppressed in those new tables as well, and cells already disseminated must not be used as secondary suppression in the additional tables to avoid disclosure risks. However, the latter, sometimes referred to as “frozen cells”, may cause an infeasible problem for the cell suppression algorithm, creating a major obstacle for using  $\tau$ -Argus in this use case, especially in a widely automated production process. The paper will propose a work around for the infeasibility problem, and explore issues of respective enhancement of the Modular method.

# Considerations to deal with the frozen cell problem in $\tau$ -Argus Modular

Draft version of 20.11.2021

Sarah Giessing<sup>\*</sup>, Peter-Paul de Wolf<sup>\*\*</sup>, Michel Reiffert<sup>\*</sup>, Felix Geyer<sup>\*</sup>

<sup>\*</sup> Statistisches Bundesamt, Germany, [Sarah.Giessing@destatis.de](mailto:Sarah.Giessing@destatis.de), [michel.reiffert@destatis.de](mailto:michel.reiffert@destatis.de), [felix.geyer@destatis.de](mailto:felix.geyer@destatis.de)

<sup>\*\*</sup> Statistics Netherlands, The Netherlands, [pp.dewolf@cbs.nl](mailto:pp.dewolf@cbs.nl)

**Abstract:** The package  $\tau$ -Argus offers several alternative algorithms for secondary cell suppression, one of which, referred to as “Modular” is recommended best practice for application to large, complex structured sets of hierarchical and linked tables. The issue we mainly address in this paper is an important use case still lacking a sound methodological approach. When protecting tables in a late stage of the dissemination process of a statistics, one has to take into account the suppression pattern of disseminated tables released earlier: Suppressed cells must be suppressed in those new tables as well, and cells already disseminated must not be used as secondary suppression in the additional tables to avoid disclosure risks. The paper proposes a work around for the infeasibility problems frequently caused by such restrictions.

## 1 Introduction

The package  $\tau$ -Argus is a widely used EU-funded Open Source tool for disclosure control in tabular data. For secondary cell suppression it offers several alternative algorithms, one of which, referred to as “Modular”, is recommended best practice for application to large, complex structured sets of hierarchical and linked tables. Meanwhile, several NSIs and Eurostat have invested into integration of the software into production environments. The paper assumes some familiarity with the concepts of the Modular algorithm, like e.g. the concept of protecting a table as set of linked subtables and the concept of subtables of hierarchical tables itself, feasibility intervals and protection levels and the different risk-models for hierarchical and linked tables (c.f. Hundepool et al, 2012, chap. 4).

The issue we mainly address in this paper is an important use case still lacking a sound methodological approach: The usual production process for a statistic involves dissemination of a major set of pre-planned tables as soon as data are available for this production step. Typically, later on, there will be requests from users for specific additional tabulations, often linked to the tables of the already disseminated ones.<sup>1</sup> When protecting such additional tables sharing identical cells with already released ones, one has to take into account the suppression pattern of the disseminated tables: Suppressed cells must be suppressed in those new tables as well, and cells already disseminated

---

<sup>1</sup> Notably, even pre-planned tables of different statistics (on related subjects) can share certain totals or sub-tables, but one of the statistics will be released (and protected) earlier.

must not be used as secondary suppression in the additional tables to avoid disclosure risks. Sometimes, users try to give the disseminated cells extremely high costs, such that they will not be used as secondary suppression if possible. However, if needed to avoid infeasibility, Argus will still select those cells. Therefore, with this approach we can still end up with tables that have suppressed values published in another table, c.f. the example in sec. 2. This is why in the present paper we only consider the other option  $\tau$ -Argus offers, i.e. to give such cells a “protected” status, making them unavailable for secondary cell suppression. However, presence of such “frozen” cells, unavailable for secondary cell suppression, may cause an infeasible problem for the cell suppression algorithm, creating a major obstacle for using  $\tau$ -Argus in this use case, especially in a widely automated production process.

Section 2 provides some background to the problem we deal with in this paper. In section 3, the paper will propose on one hand ideas to identify/evaluate infeasibility issues due to the presence of frozen cells and ideas for mechanisms allowing users of Modular to tolerate certain underprotection risks in order to increase the amount of data that can be released “safely” in a relaxed sense of what is “safe”. On the other hand, we describe a work around for remaining, intolerable infeasibility problems in section 4. Section 5 discusses an illustrative example. The paper finishes with a brief summary.

## 2 Background

Statistical offices collect information on several properties that might be used for grouping respondents, like e.g. information about respondent economic activity (NACE) and geographic location. A naïve approach mentioned as one of three ways to deal with linked tables in (De Wolf and Hundepool, 2010) might be to create all potentially interesting cross-combinations of grouping variables, also at various levels of detail of those variables, and subject the resulting tables to primary and secondary cell suppression. The problem in practice is that those tables would often form a set of linked, huge, multidimensional, hierarchical tables involving many zero cells, and many cells with very few contributions on the lower levels of aggregation, which will turn out to be primary suppressions. However, the presence of many zero cells which usually cannot be used as secondary suppressions often requires secondary suppressions one or more levels of aggregation above the level of the primary suppression requiring the protection. Those higher level secondary suppressions themselves then require further secondary suppressions, and sometimes these again are only to be found on the next (or even higher) level of aggregation. Because of this, the naïve approach has a tendency to lead to rather many secondary suppressed high level aggregate cells on account of protecting much less relevant low level aggregate tables.

It is therefore good practise to split the dissemination process into two phases: In the first phase, the agency releases only those tables identified by the statistics experts as relevant for the users of a statistic in general. Of course, these tables should be protected

in a joint step, i.e. as set of linked tables, such that secondary suppressions are coordinated as offered by the Modular approach for linked tables in  $\tau$ -Argus.

In the second phase, users may put in requests for some extra results or tables left aside in the first phase. Before releasing them, protection has to be applied to these additional tables as well, in coordination with the suppression pattern of phase one: This means, if a new table is linked to the disseminated set of tables, all table cells which are logically identical must have the same suppression status, i.e. suppressed “*unsafe*”, or not suppressed, “*safe*” cells. The protection procedure has to be prepared in such a way that  $\tau$ -Argus treats phase 1 secondary suppressed cells as unsafe cells, and that phase 1 safe cells become not eligible for suppression. For the latter,  $\tau$ -Argus offers two alternative options: One is to assign high penalty costs to those cells. The other, more direct option is to mark them as “*protected*”, a.k.a. “*frozen*” cells.

However, there can be constellations, where under the constraints caused by “frozen” cells no feasible solution exists, as illustrated by the table in figure 1 below. For this instance, we assume only the margins of the table have been considered as relevant in phase 1, and have got released in that phase without suppressions. In phase 2, those cells must not be used as secondary suppressions – they must keep their status “safe”. In this instance, adding penalty costs will not prevent the algorithm from picking those “safe” cells as secondary suppression. The solution will be inconsistent (to phase 1), but this will be up to the user to check: since  $\tau$ -Argus is not aware of phase 1, it cannot issue a warning. As mentioned in the introduction, we therefore do not consider this approach in this paper.

If the user marks those cells as “protected” on the other hand, Modular quits with an error message<sup>2</sup>, and provides no solution. For an unstructured, non-hierarchical table this may be appropriate. However, phase 2 tables may have complex hierarchical structures just as well as phase 1 tables do. And returning no solution for a large table when the infeasibility problem caused by cells marked “protected” affects only one particular subtable is very inconvenient and makes the method more or less impossible to use in phase 2 production scenarios.

Figure 2 provides some illustration: The table is 2-dimensional with hierarchical structure in the variable NACE. We assume all margins and submargins got released in phase 1, they must hence keep their status “safe”. This causes infeasibility of the subtable relating to parent node *A* of that hierarchy. Notably, the subtable is identical to the one of figure 1. In the two other subtables relating to nodes *B* and *C*, no infeasibility problem occurs. In this example the best and most practical solution would be for Modular to return a solution where all interior cells of the node *A* subtable are marked

---

<sup>2</sup> “Error in modular suppression procedure HiTaS: lower- and upper bound both equal infinity.”

as suppressed, including the zero cell. This is the concept worked out in some more detail in sec. 4.

Region x NACE			
	- A	A1	A2
- R1	18	10	8
1A	4	-	4
1B	14	10	4

**Fig. 1** Simple table, infeasible due to frozen cells

Region x NACE										
	- Total	- A	A1	A2	- B	B1	B2	- C	C1	C2
- R1	68	18	10	8	24	16	8	26	18	8
1A	26	4	-	4	10	6	4	12	8	4
1B	42	14	10	4	14	10	4	14	10	4

**Fig. 2** Hierarchical table, infeasible due to frozen cells

Especially with respect to that part of the paper, we limit the problem we address to the following case: A set of  $N$  linked phase 2 tables  $\{T_1, \dots, T_N\}$  that need to be protected simultaneously. For those tables using the same spanning variables we assume that they have hierarchies that can be covered in the sense of (de Wolf and Hundepool, 2010). This means, we assume a single hierarchy can be constructed such that all hierarchies of the same variable in the  $N$  tables are a sub hierarchy of the cover hierarchy. Under these circumstances, in the absence of any “protected” cells, we could process those tables with the modular approach for linked tables (de Wolf and Hundepool, 2010). This means that we use the Modular approach (de Wolf, 2002) on the cover table  $T_c$ , but only consider those subtables that are also subtables of at least one of the specified tables  $T_1, \dots, T_N$  and skip any other subtable that is not a subtable of any of the tables in the set  $\{T_1, \dots, T_N\}$ .

### 3 Infeasibility of a subtable with protected cells – checks and remedies

Within the modular method, a suppression pattern for a particular subtable is computed by an optimization routine, implementing the MILP approach of Fischetti and Salazar (2000). Especially in cases of infeasibility, a call to this routine sometimes finishes with a somewhat unclear result. In such a case we suggest to implement a check to find out, if “*complete suppression*” of the subtable subject to frozen cells, i.e. suppressing all cells of this subtable except for the “protected” ones and except for zero cells not eligible for suppression, might be considered a feasible solution for users willing to accept certain disclosure risks.

We imagine several alternatives for such a check at different degrees of computational complexity, and at different levels of tolerated underprotection risks. Notably, checks (3) and (4) should give identical results in theory, but might perform differently in practice. They are more rigorous compared to checks (1) and (2). Checks (2) to (4) should be implemented subject to eventually relaxed protection requirements regarding non-primary sensitive cells, see sec. 3.1.

Of course, subtables passing checks (3) or (4) will also pass check (2), and those passing check (2) always pass check (1) as well. The same holds the other way round, i.e. those failing the simple checks also fail the rigorous ones. Hence, which of the checks to actually implement is a matter of computational performance: heuristic checks should only be implemented, if the efficiency gain of not having to run the rigorous checks on those subtables already failing the heuristic ones is worth the effort of running several checks on those passing them.

- (1) **Simple heuristic check.** The check only ensures that complete suppression yields for every subtable relation that either no, or at least two cells are suppressed, and that for every equation which contains more than one “*singleton*” (i.e. table cell based on only a single contribution) primary suppression, at least three cells are suppressed.
- (2) **Extended heuristic check.** In addition to the simple check, in the extended version we also check for any primary suppression within a subtable relation that the required protection level is afforded by the other suppressions in that equation.
- (3) **Check involving optimization routine.** This check passes the subtable once again to the optimization routine, after manipulating the input to reflect complete suppression of the subtable (except for the protected and zero cells), to see, if the routine returns a clear result for this modified version of the problem.
- (4) **Check by subtable audit.** The check computes a feasibility interval for every primary and secondary suppression subject to complete suppression of the subtable (except for protected and zero cells). A subtable would pass this check if all feasibility intervals satisfy the protection requirements of the respective cells with respect to the relaxed protection requirements regarding any non-primary sensitive cells, see sec. 3.1.

### 3.1 Tolerating underprotection risks as remedy of technical infeasibility

Feasibility of a cell suppression problem defined for a subtable with protected cells can depend specifically on protection requirements assigned to the cells in the subtable. While  $\tau$ -Argus derives protection requirements for primary unsafe cells based on fairly sound methodology (c.f. Hundepool et al., 2012, 4.3.2), for cells unsafe because they have been selected as secondary suppression in another subtable, Modular uses the minimum of a fixed percentage  $q$  of its cell value and the maximum protection level of all primary suppressions in this other subtable as heuristic approach. This ensures that in this other subtable  $ST_1$  which is of course linked to the original subtable  $ST_0$ , the suppression pattern will be computed in such a way that the upper and lower bounds of the feasibility interval for that cell will differ from its true value by at least  $q\%$  or by the largest protection level of suppressions in  $ST_0$ ,  $\max(\text{pl}(T_0))$ . However, as explained in Appendix A.1, in certain constellations the protection requirement of  $\max(\text{pl}(T_0))$  for a secondary suppression from  $ST_0$  in

subtable  $ST_1$  can be unnecessarily large, and this can have an undesirable effect, damaging especially in a situation with frozen cells:

Assume for example,  $ST_1$  to be at least two dimensional and the secondary suppression with value  $y$  from subtable  $ST_0$  to be a margin cell of an equation of  $ST_1$  with only two non-zero interior cells with cell values  $0 < z_1 < z_2$ , and  $pl(x) \leq z_1 < \max(pl(T_0)) < z_2$ . Assume further, the larger cell with value  $z_2$  to be the only non-zero interior cell in another equation and the margin cell of this equation (with identical cell value  $z_2$ ) to be protected. Then, at least regarding that table equation, the smaller cell with value  $z_1$  would provide enough protection for the primary unsafe cell of  $ST_0$  and this choice may lead to a feasible solution. But because  $z_1 < \max(pl(T_0))$  only the larger cell with value  $z_2$  will be eligible for the algorithm. Then, because of its protected margin,  $ST_1$  will prove infeasible for the secondary cell suppression algorithm. Sec. 5 presents an illustrative example for such a case where infeasibility is caused by an unnecessarily big protection requirement assigned by Modular to a secondary suppression.

De Wolf and Giessing (2009) drafts a method to determine protection levels for secondary suppressions in a theoretically sound way using “*partial suppression*” methodology (Salazar, 2005). However, this approach is not yet available in Argus. As a result, users of Modular have to put up with some underprotection risks due to too small protection levels, and at the same time with some secondary suppressions that are unnecessarily large, or located in subtable margins (eventually affecting further subtables), and may even cause subtable infeasibility in situations with frozen cells.

While a sound methodology (based on methods such as partial suppression) is not yet available (i.e.: not yet implemented) in Modular, one might consider a simple alternative to define protection levels for secondary suppressions to be offered with Modular in general, or just when checking a subtable with an unclear status regarding its feasibility:

### **Secondary suppression protection levels avoiding only exact disclosure**

This option may lead to underprotection risks, but reduces cases of infeasible subtables to the best possible extent. According to the option, protection levels for secondary suppressions carried over from other subtables would be set to a fixed small amount, like the smallest non-zero value of all non-frozen cells in the table. Using this option in a scenario without frozen cells reduces cases of secondary suppression selected in subtable margins (hence leading to additional secondary suppressions in other linked subtables) to the best possible extent under the restriction of avoiding exact disclosure.

To reduce the underprotection risks associated to the approach, a variant could be, to execute the CSP algorithm twice for a subtable: In the first execution, the fixed small amount protection levels are used with the secondary suppressions. In the second run (which would be carried out independent of the first run), the standard protection

levels (e.g. small percentage of their cell value) are applied. If this second problem turns out infeasible, or if there are more subtable margin cells suppressed than by the first run (if the reduced protection levels should also apply in scenarios without frozen cells), the result of the second run would be discarded, otherwise we would discard the result of the first run.

#### 4 Skipping infeasible subtables as last resort

Assume now a scenario where the option of reducing protection levels does not apply, or does not solve the problem and a subtable with protected cells fails the checks of sec. 3. What we want to avoid is that Modular simply quits, providing not even feasible solutions for subtables eventually processed earlier.

The approach we propose here to deal with frozen cells in stage-2 tables is just a straightforward extension of the modular approach for linked tables (de Wolf and Hundepool, 2010) to skip certain subtables. The main difference is the following: in the “normal” linked tables approach, it is clear from the start, which subtables to skip, and which to consider. With the new method we suggest here, some of the subtables to skip are identified “on the fly”, at runtime. Like in the “classical” modular approach for linked tables we of course skip any subtable that is not a subtable of any of the tables in the set  $\{T_1, \dots, T_N\}$ . But now, additionally, when one of the other subtables is processed, and there are “protected” cells in that subtable, whenever this problem turns out to be infeasible, we skip this subtable for the remainder of the process, along with any of its “descendant”-subtables.

With the classical modular approach for linked tables, assigning a suppression status to cells of a skipped subtable is not an issue, because the skipped subtables do not belong to any of the set  $\{T_1, \dots, T_N\}$  anyway. But the skipped infeasible or descendant subtables do, so they need a suppression status. This should be a special status which should be assigned to *all interior cells including the zero cells* in the subtables, indicating that no cell with this status – zero or non-zero - must be published.

The general idea behind this approach is that releasing infeasible subtables, or their descendants, does not give away more information than the information already released at stage 1 plus the information in the feasible phase 2 subtables and should therefore be acceptable from a disclosure risk management point of view.

##### 4.1 What is a descendant-subtable?

A subtable of an  $n$ -dimensional hierarchical table is defined as cross combination of table relations:  $e_1 \times \dots \times e_n$ . The  $i$ th relation  $e_i$  defines the relation between one non-bottom category  $top(e_i)$  of the  $i$ th classification variable (for example “food production sector”) and those  $k(e_i)$  categories  $(bot(e_i)_j)_{j=1, \dots, k(e_i)}$  on the level immediately below belonging to this category (*here*: bakers, butchers, etc.). We refer



to  $top(e_i)$  as  $i$ th *parent node* of the sub-table, and to the  $bot(e_i)_j$  as *child nodes* of the  $i$ th parent node of the subtable. If one of the  $bot(e_i)_j$  nodes is the same as the  $top(e_i')$  node of another sub-table, i.e. if this child node is also a parent node of another sub-table, then we also refer to the child nodes  $bot(e_i)_j$  as descendant nodes of the higher level parent node  $top(e_i)$ . Notably, we also call the original child nodes  $bot(e_i)_j$  descendant nodes of  $top(e_i)$ .

We call an entire sub-table  $e_1' \times \dots \times e_n'$  a descendant subtable of a subtable  $e_1 \times \dots \times e_n$ , if for all dimensions  $i = 1, \dots, n$  either  $e_i' = e_i$ , or  $top(e_i')$  is a descendant of  $top(e_i)$ .

## 5 An Illustrative Example

For illustration of the propositions presented above, we use the structure of the instance of a single 2-dimensional table used in (de Wolf, 2002), but with a modified setting regarding cell values and primary suppressions, c.f. fig. 4.

R x BC										
	-BC	-I	LI	MI	SI	-A	LA	SA	O	
-R	1645	90	80	5	5	1000	995	5	555	
P1	550	50	50	-	-	-	-	-	500	
-P2	1080	30	20	5	5	1000	995	5	50	
-C21	995	-	-	-	-	995	995	-	-	
D211	105	-	-	-	-	105	105	-	-	
D212	890	-	-	-	-	890	890	-	-	
C22	85	30	20	5	5	5	-	5	50	
-P3	15	10	10	-	-	-	-	-	5	
C31	5	-	-	-	-	-	-	-	5	
C32	10	10	10	-	-	-	-	-	-	

**Fig. 4** Suppression pattern for table  $R \times BC$ , cell values of primary unsafe cells displayed in red, secondary suppressions obtained by “Optimal”, in blue

R x BC										
	-BC	-I	LI	MI	SI	-A	LA	SA	O	
-R	1645	90	80	5	5	1000	995	5	555	
P1	550	50	50	-	-	-	-	-	500	
-P2	1080	30	20	5	5	1000	995	5	50	
-C21	995									
D211	105									
D212	890									
C22	85									
-P3	15	10	10	-	-	-	-	-	5	
C31	5	-	-	-	-	-	-	-	5	
C32	10	10	10	-	-	-	-	-	-	

**Fig. 5** Table  $R \times BC$ , shaded bars covering interior cells of six skipped subtables, c.f. sec. 5.2.

The instance assumes cell (R,A) to be primary unsafe with upper and lower protection levels of 4, and (P1,O) unsafe with protection level of 20. We also assume that the margins of variable  $BC$  have been published already at the higher levels (0, 1, 2) of variable  $R$ , thus the status of these cells has to be frozen. As all those margin cells are unsuppressed in this instance, they are not allowed as secondary suppressions. We set them to protected when setting up the instance for  $\tau$ -Argus.

We further assume Modular to protect the subtables in a sequence defined by crossings of hierarchy levels  $(a_1, a_2)$  of variables  $R$  and  $B$ , here in the sequence (1,1), (2,1), (1,2), (2,2), (3,1) and (3,2) as suggested in (de Wolf and Loeve, 2004), with eventual backtracking when subtable margins of level 2 onwards get suppressed in a subtable. Throughout this instance we denote subtables by  $(a_1, a_2)|(R_i, BC_j)$ , i.e. by the crossing of hierarchy levels they relate to, together with categories  $R_i$  and  $BC_j$  of the grand total cell of the respective subtable.

When protecting the first subtable  $(1,1)|(R,BC)$ , four cells will be suppressed. In this simple case, it is obvious that three of them  $\{(P2,O), (P2,I), (P1,I)\}$  are selected to protect cell  $(P1,O)$ , while the fourth,  $(P2,A)$ , is secondary suppression to  $(R,A)$ . When processing in the next step the subtables relating to the crossings of levels  $(2,1)$  and  $(1,2)$ , those cells will be handled (temporarily) as primary suppressions. In appendix A.2 we explain why the protection level of  $(P2,A)$  in subtable  $(2,1)|(P2,BC)$  is likely to be larger than 5, and why this leads to infeasibility of subtable  $(2,1)|(P2,BC)$ . In fact, current versions of Modular (Version 4.1.7, for example) abort when attempting to protect this instance.

### 5.1 Solution based on the risk toleration approach





In this particular instance, the problem is clearly due to how Modular assigns the protection level to the temporary primary suppressions, because a protection level of 4 would indeed be sufficient for  $(P2,A)$  to protect in return  $(R,A)$ . Actually, the  $\tau$ -Argus full optimization algorithm (“Optimal”) returns a feasible solution for the instance, cf. fig. 4. However, while restoring to Optimal is no problem for this tiny instance, for our typical large 3- and more dimensional hierarchical tables it is not usually an option because of the enormous computational burden of the full optimization method.

In order to get a solution from Modular, we now follow our proposal from sec. 3.1 to rather tolerate some underprotection risks, if this way we can obtain a suppression pattern that at least avoids exact disclosure. As suggested in sec. 3.1, we use as protection level for temporary primaries the smallest non-zero value of all non-frozen cells, which is 5 in our instance and leads to selection of  $(C22,A)$  to protect  $(P2,A)$  in subtable  $(2,1)|(P2,BC)$ .

### 5.2 Solution based on the skipping of infeasible subtables

Alternatively, we assume now that reducing protection levels for temporary primary suppressions is not an option. Then, when we process subtable  $(2,1)|(P2,BC)$ , the optimization routine cannot provide a feasible result, because  $(C21,A)$  will have to be secondary suppression for  $(P2,A)$  in that column of the subtable. But, as pointed out above, it is the only non-zero interior cell in its row of this subtable, and the row margin,  $(C21,R)$  is a protected cell. We assume now the optimization-routine to deliver a result with all cells suppressed, except for the protected and empty cells. For this pattern, even the simple heuristic check (1) suggested in sec. 3 will fail:  $(C21,A)$  is the only suppressed cell in the row relation corresponding to Region  $C21$  in this subtable. Hence, obviously the value of  $(C21,A)$  must be identical to the (assumed to be published) value of  $(C21,R)$ .

According to our suggestion of sec. 4, Modular should then skip the subtable  $(2,1)|(P2,BC)$  for the remainder of the process, along with any of its “descendant”-subtables. This means it would have to skip the following subtables as illustrated by fig. 5 using differently shaded bars covering the respective interior cells:

  $(2,1)|(P2,BC)$ , 
   $(2,2)|(P2,I)$ ,  $(2,2)|(P2,A)$ ,  
  $(3,1)|(C2I,BC)$    $(3,2)|(C2I,I)$ ,  $(3,2)|(C2I,A)$ .

The remaining subtables, i.e.  $(1,1)|(R,BC)$ ,  $(1,2)|(R,I)$ ,  $(1,2)|(R,A)$ ,  $(2,1)|(P3,BC)$ ,  $(2,2)|(P3,I)$  and  $(2,2)|(P3,A)$  will be processed.

## 6 Summary and Final Remarks

The paper has addressed a relevant use case for software implementing secondary cell suppression, i.e. the case where a user submits tables linked to other, already published tables. A suppression pattern involving as secondary suppressions cells already disseminated in an earlier publication computed on the basis of the same dataset is of course unsafe. The protection it provides to sensitive cells in the new tables can be easily undone (often leading to exact disclosure of sensitive cells) by users of the new tables when taking into account the already disseminated tables. Therefore, productive systems implementing automated cell suppression for this use case must not ignore such risks.

At the same time, such a system must be able to provide pragmatic solutions for hierarchically structured tables. Meaning that it must be able to identify those subtables that can actually be released safely, or eventually at the expense of accepting certain residual disclosure risks. I.e. accepting in particular those risk issues<sup>3</sup> that might already exist (in a hidden way), when disseminating – in the earlier publication phase – results which are margins of subtables considered now, but not taken into account in the SDC process of the earlier phase. We also suggest making the system tolerate certain risks of inferential disclosure occurring only when an intruder considers more than one subtable at a time. After all, this would be for the sake of being able to offer a practical alternative to what might be practitioner’s risky practices now, e.g. ignoring – now and then - risks of exact disclosure due to suppression of already published cells.

While the paper has outlined our ideas and concepts, implementing a proof of concept useable to investigate their practical efficiency is beyond the scope of this paper, but hopefully subject of our future work.

---

<sup>3</sup> I.e. risks typically due to the existence and location of zero or singleton cells that might be known to intruders with special insider knowledge.

## References

- De Wolf, P.P. (2002), ‘*HiTaS: A Heuristic Approach to Cell Suppression in Hierarchical Tables*’, In: ‘Inference Control in Statistical Databases’ Domingo-Ferrer (Ed.), Springer (Lecture notes in computer science; Vol. 2316)
- De Wolf, P.P., Loeve, A. (2004), ‘Reducing the set of tables  $\tau$ -ARGUS considers in a Hierarchical Setting’, *Privacy in Statistical Databases*, J. Domingo-Ferrer and V. Torra (Eds.), Springer 2004, LNCS 3050 pp. 99-109
- De Wolf, P.P. and S. Giessing (2009), *Adjusting the  $\tau$ -ARGUS modular approach to deal with linked tables*, Data & Knowledge Engineering, Volume 68, Issue 11, pp. 1160-1174.
- De Wolf, P.P. and A. Hundepool (2010), *Three ways to deal with a set of linked SBS tables using  $\tau$ -ARGUS*, *Privacy in Statistical Databases*, J. Domingo-Ferrer and E. Magkos (Eds.), Springer 2010, LNCS 6344 pp. 66-74.
- De Wolf, P.P. de, Hundepool, A., Giessing, S., Salazar, J.J., and Castro, J. (2014), ‘ *$\tau$ -ARGUS User's manual*’, Statistics Netherlands, The Hague.
- Fischetti, M, Salazar Gonzales, J.J. (2000), *Models and Algorithms for Optimizing Cell Suppression Problem in Tabular Data with Linear Constraints*, in Journal of the American Statistical Association, Vol. 95, pp 916
- Fischetti, M., Salazar Gonzales, J.J. (2005), ‘*A Unified Mathematical Programming Framework for different Statistical Disclosure Limitation Methods*’, in Operations Research 53/5 pp. 819-829
- Hundepool, A., Domingo-Ferrer, J., Franconi, L., Giessing, S., Schulte Nordholt, E., Spicer, K., and Wolf, P.P. de (2012), *Statistical Disclosure Control*, Wiley, Chichester, United Kingdom.

## Appendix

### A.1 A constellation in which a protection requirement of $\max(\text{pl}(T_0))$ for a secondary suppression from $ST_0$ in subtable $ST_1$ will be unnecessary large

As observed in sec. 3.1, the way Modular derives protection requirements for cells unsafe in a subtable  $ST_1$  because they were selected as secondary suppression in another subtable  $ST_0$ , ensures that the suppression pattern for  $ST_1$  will be computed in such a way that the upper and lower bounds of the feasibility interval for that cell will differ from its true value by at least  $q\%$  or by the largest protection level of suppressions in  $ST_0$ ,  $\max(\text{pl}(T_0))$ .

The worst case still possible then, is that users of the protected subtable  $ST_1$  can derive a bound  $\hat{y}$  for a true value  $y$  of a secondary suppressions where  $|y - \hat{y}| =$

$\min(\frac{q}{100}y, \max(\text{pl}(T_0)))$ , for example lower bounds  $\hat{y} = y - q y/100$  or  $\hat{y} = y - \max(\text{pl}(T_0))$ .

If in the subtable  $ST_0$ , where the cell was originally selected, this secondary and the primary unsafe cell it protects, with, say, cell value  $x$ , are the only suppressed cells of a table equation (and both are interior cells), the new bound  $\hat{y}$  can be used to derive an upper bound for  $x$  of (1)  $x + y - (y - q \frac{y}{100}) = x + q \frac{y}{100}$  or (2)  $x + y - (y - \max(\text{pl}(T_0))) = x + \max(\text{pl}(T_0))$ .

Then, if the protection level of the primary unsafe cell,  $\text{pl}(x)$ , is close to  $y$  and  $q$  is small, this bound will be the one of case (1), and it will be much too close. The unsafe cell would be underprotected.

On the other hand, if  $y$  is large, then  $\max(\text{pl}(T_0))$  can be smaller as  $q\%$  of  $y$  and the new bound  $\hat{y}$  can be the one of case (2), i.e.  $x + \max(\text{pl}(T_0))$ . If in that case the protection level  $\text{pl}(x)$  of the “initial” primary suppression is much smaller as  $\max(\text{pl}(T_0))$ , the  $\max(\text{pl}(T_0))$  protection level will be much larger than necessary to provide protection to cell value  $x$ .

## A.2 Details of the illustrative example of sec. 5:

### Why Modular is likely to use an unnecessary large protection level for the temporary primary suppression $(P2,A)$ and how this causes infeasibility of the instance

As mentioned in sec. 3.1, Modular uses as protection levels for a temporary primary suppression the minimum of a fixed percentage  $q$  of its cell value and the maximum protection level of all primary suppressions in this other subtable.

In the instance of sec. 5, the largest protection level of unsafe cells in subtable  $(1,1)|(R,BC)$  is 20 (relating to cell  $(P1,O)$ ). As in this simple instance obviously  $\{(P2,O), (P2,I), (P1,I)\}$  are protecting  $(P1,O)$ , in principle they should directly inherit that protection level.

However, the fourth secondary suppression  $(P2,A)$  in subtable  $(1,1)|(R,BC)$  which is secondary suppression to  $(R,A)$ , should in principle inherit the protection level of  $(R,A)$ , assumed to be 4 in the instance.  $(P2,A)$  turns into a temporary primary suppression when processing for example subtable  $(2,1)|(P2,BC)$ . With the (in the instance) correct protection level 4,  $(C22,A)$  (cell value: 5) would be selected to protect  $(P2,A)$  in this subtable. Afterwards the process would continue and finally lead to the secondary suppression pattern presented in fig. 4.

But actually, we obtained the pattern of fig. 4 using the  $\tau$ -Argus full optimization algorithm (“Optimal”). Modular, on the other hand, as explained above, would in fact use as protection level for the temporary primary suppression  $(P2,A)$  the minimum of a fixed small percentage  $q$  of its cell value (1000) and the maximum of the protection

levels in the subtable  $(1,1)|(R,BC)$  which is 20 and thus larger than 5. So, the small percentage  $q$  would have to be at most 0,5% , otherwise  $(C22,A)$  would be too small to be accepted as secondary suppression. In that case,  $(C21,A)$  (cell value 995), would be the only acceptable candidate secondary suppressions. But this causes a problem:  $(C21,A)$  is the only non-zero interior cell in its row of the  $(2,1)|(P2,BC)$  subtable, and the row margin,  $(C21,R)$  is a protected cell. This means, protection of  $(P2,A)$  with protection level larger than 5 in subtable  $(2,1)|(P2,BC)$  is infeasible! In fact, current versions of Modular (Version 4.1.7, for example) abort when attempting to protect this instance.