Comparing Ways of Using 'Protection Flow' to Protect Magnitude Data Tables from Disclosures

Paul B. Massell
Statistical Research Division
Room 3209-4, U.S. Census Bureau, Washington, D.C. 20233
paul.b.massell@census.gov

Abstract

We use the term 'protection flow' to describe any solution to a mathematical programming model that describes perturbations (i.e., modifications) of table cell values that preserve table additivity. This term is meant to generalize 'network flow'. Network flow models have been used to describe such perturbations for simple two dimensional tables. Protection flow models can be constructed for tables of any dimension and any degree of hierarchy. Cell suppression is one way of using the protection flow to construct a table with sufficient protection for the sensitive cells. We compare cell suppression with some new ways (e.g., controlled tabular adjustment, variable base rounding, uncertainty intervals) of using the perturbations to provide disclosure protection.

Keywords: disclosure, confidentiality, mathematical programming, protection flow, magnitude data tables, cell suppression, controlled tabular adjustment

1. Introduction[1]

Let's begin with an example that describes a situation that arises frequently at statistical offices. Suppose a statistical office (SO) would like to release a 2-dimensional additive table. Additive in this setting means that there is a sum row; i.e., a row that is the sum of the other rows and a column that is the sum of the other columns. If the sum row is the lowest row and the sum column is the rightmost column then the grand total cell will be the one in the lower right corner. Suppose the SO determines that some of the cells are sensitive according to some sensitivity measure. For concreteness, let's use the p% rule for all of our examples. This rule is typically used with tables that contain magnitude data, rather than count data. For magnitude data, a cell value is usually the sum of the contributions of at least a few contributors (e.g., companies). However, if there were only one contributor and the cell value were released (i.e., published) and it was equal to the single contributed value, it is likely that this would violate the confidentiality guarantee that the SO pledged to all the contributors.

Let assume the SO's confidentiality pledge takes the following form: any contribution will be protected in the sense that the best estimate of a contribution 'x' that is possible for any table user (other than the contributor of 'x') to derive (using any standard mathematical or statistical procedure and) using the published values in the table is either (1) an interval containing x of width at least 2*(p/100)*x; if "sliding protection" is being used or (2) an interval of the form [x-a, x+b] where both 'a' and 'b' are at least (p/100)*x ; if 2-sided protection is being used. The value of p that is used is based on a policy decision of the SO. In order to come up with a sensible value of 'p', the SO may consult with a sample of the organizations or individuals whose data is being protected from exact disclosure.

Thus the goal is to ensure that there is an 'uncertainty interval' of sufficient width about each contribution to each cell. Sliding protection and two-sided protection are two possible 'position rules'; such rules determine the placement of the uncertainty interval with respect to the value being protected. Typically when there are several contributions to a cell, the existence of the 3rd largest and smaller contributions provide sufficient protection to the largest contribution x1 so that the contributor of x2 cannot determine x1 accurately. However, this is not always the case; e.g., x3 may equal zero. When the p% rule determines that a cell is sensitive, it is saying that the uncertainty interval for the largest contribution to the cell, x1, is not large enough to protect it. One must determine how much (additional) protection is needed and decide on a way to provide that amount of protection. Recall that the table is assumed to be additive. In general, whatever methods are used to protect the sensitive cells, we still would like the final table (i.e., the table after protection has

[1] *This report is released to inform interested parties of ongoing research and to encourage discussion of work in progress. The views expressed on statistical, methodological, technical, or operational issues are those of the author and not necessarily those of the U.S. Census Bureau.*

been provided) to be additive. This implies that changing a single cell value is not possible.

1.1 Protection Flow

Let us now, discuss how 'protection flow' is generated. In the case of a 2-dimensional table, changing one cell, say in location (1,1) will force a change in at least one other cell in row 1 (say (1,3)) and at least one other cell in column 1 (say (3,1)). Those changes in turn will force changes in at least one other cell (here (3,3)). This 'domino effect' is what generates 'protection flow'.
Our general definition of 'protection flow' is the assignment of changes to cell values of a given table for the purpose of creating a specified amount of uncertainty about the values of one or more of the (sensitive) cells. Usually these changes satisfy the additive structure of the given table.
The quantity that 'drives (or forces) the system' is the protection requirement calculated by the p% rule. Below we will be more precise about how to use the p% rule to determine needed protection. Below we give two examples for 4 by 4 tables (3 by 3 for the interior cells). In table 1, there is a single (protection flow) rectangle, also called a simple cycle. It involves only 4 cells. Table 2 is more involved; 8 cells are involved and they lie in two overlapping rectangles. In both tables, one sensitive cell, needing 5 units of protection, drives the system. The other cells are selected in a way to be explained below. However, note that the sum of protection changes in each row and column is zero. One could say the 5 units flow in each axis direction (row and column in a simple 2-d table) in which the driver cell lies. This flow is distributed to one or more other rows, and one or more other columns, depending on the protection pattern selected.

1.2 Geometry of Protection Flow

In 2-dim. tables, protection flow generates a set of overlapping rectangles with flow on vertices. In 3-dim tables, flow involves a set of overlapping 3-d rectangles.

Table 1: A simple protection cycle

+5 (Driver)		-5	
-5		+5	

Table 2: Two overlapping protection rectangles

+5 (Driver)	-2	-3	
-1	+1		
-4	+1	+3	

2. The Major Steps in Using Protection Flow to Protect a Single Table

2.1 Brief Description

a. Determine which cells are sensitive and how much additional protection they need

b. If protection of sensitive cells is to be done sequentially, order them in some reasonable way, perhaps by the amount of protection needed

c. Select a protection flow method to apply to the given table; it should be expressible as an optimization problem using standard mathematical programming (MP) models

d. For each sensitive cell, determine how much protection the other cells can provide to it.

e. Determine the bounds of the MP model; these put limits on the amount of protection flow that can pass through cells; they are based on assumptions of users' prior knowledge of cell values.

f. Determine the constraints of the MP model; these will probably reflect the condition that the protection flow table is additive. This will ensure that the protected table (which for some methods equals the given table plus the protection flow table) is additive.

g. Determine the quantity to be minimized; it will reflect information loss. It will be the objective function (often called the cost function) of the MP model.

h. Run a program that implements the developed MP model on the given table.

Evaluation steps

i. If the method used guarantees that sufficiently wide

uncertainty intervals will be created for each sensitive cell, then there is no need to run an audit program. Otherwise, one should run an audit program to check the size of the uncertainty intervals. If some are too small, one needs to devise a plan to bring them up to required size.

j. The SO must make sure that the uncertainty intervals are not vulnerable to a simple attack. That is, sufficiently wide intervals do not guarantee full protection. One needs to consider what strategies a clever table user might employ to estimate the true cell values that the SO is trying to hide. The simplest of these is perhaps the midpoint attack. Here the table user takes the uncertainty interval (UI) that the SO supplies for a given cell and uses the midpoint of the interval as an estimate for the true cell value. If the SO does not supply the UI for the cell, the table user will generally be able to derive such an UI himself.

k. One needs to assess the overall usefulness of the protected table in the form in which it is released. In particular, one must consider both the simplest uses of the tables and statistical analysis uses (e.g., modeling, etc.).

2.2 Discussion of the steps

A. Assume (magnitude) contributions to a cell are given in decreasing order...$x1 \geq x2 \geq x3$...
Let 'rem' denote the sum of x3 and smaller contributions (if they exist).
Then, according to the p% rule, the cell is sensitive if $rem < (p/100)*x1$ and the amount of protection needed is the difference $(p/100)*x1 - rem$.

B. Certain protection methods such as cell suppression, are typically implemented in a way that protects the sensitive cells sequentially. Sequential protection makes sense if the amount of protection that a non-sensitive cell can provide to a sensitive cell depends on the sensitive cell. That is, for certain types of tabular data, the 'capacity' of a cell to protect a sensitive cell is a function of the sensitive cell. This functionality seems to be required to provide protection to the sum of contributions that come from a single source, e.g., a company or more generally an enterprise. Companies sometimes report data for two or more of their establishments; these contributions may contribute to two or more cells of a given table.

C. For very small tables or tables with only a few sensitive cells it is sometimes possible to find a reasonable protection flow with hand computation. However, in general it is necessary to use a mathematical model and the most successful models used to date have been network flow models, linear programming models, and more general mathematical programming models.
In recent years, meta-heuristic methods have been used to replace use of integer programming solution methods; even in this case, one can describe the problem as an integer program model, with either a linear or non-linear objective function.

D. As mentioned in (B), the maximum amount of protection that a general cell can provide to a sensitive cell x is called the 'capacity' of the cell to protect x.

E. In most MP protection models, there are (at least) two variables associated with each cell, one represents the amount of positive (i.e. upward) change in the cell value that is allowed; the other the amount of negative (i.e. downward) change. These bounds sometimes have the same value; in that case the common value is the capacity. Often this capacity equals the cell value. This is based on the assumption that any table user knows that the actual value of the cell is non-negative and the best informed table users know that the true value of the cell lies somewhere in the interval [0, 2*v] where v is the actual cell value. If we assumed a smaller interval but one that is still symmetric about v, say, [0.4*v, 1.6*v] the capacity would equal 0.6*v. If we assumed an asymmetric interval such as [0.4*v, 1.7*v] then we would need to have two capacities; one that bounds downward change at 0.6*v and another one that bounds upward change at 0.7*v.

F. The constraints in MP protection models are usually simply an expression of the requirement that additivity be preserved in the protected table. Additivity in a simple 2d table is a property of each row and each column. For each cell that is either being protected or providing protection, one must include in the model a constraint that expresses the additivity each shaft in which it is contained. A 'shaft' generalizes the notion of row and column to higher dimensions and/or hierarchies. Since one does not usually know beforehand which cells will be providing protection, the protection models usually have an additivity constraint for every row and column in the table. A table is not 'simple' if it has a hierarchical structure on top of its basic structure. For example a hierarchy for row 3 means a decomposition of a row into two or more "subrows" (e.g., a row 3 = row 3.1 + row 3.2). Hierarchical structure imposes additional additivity constraints.

G. An optimization problem requires an objective function that expresses a quantity that is to be

minimized (or maximized). Typically protection flow models minimize some notion of information loss. In cell suppression, traditional measures have been the sum of the values of the suppressed cell, or the number of suppressed cells. These measures are easily to implement in integer programs using binary variables, that equal '1' when there is flow through a cell and '0' otherwise. However, these loss functions cannot be represented exactly in linear (continuous variable) programs because binary variables are not available in such programs. Recently, these loss functions have been questioned. Since a table user is often able to derive an uncertainty interval for a suppressed cell, the information lost when a cell is suppressed does not really equal the full cell value. Instead, information lost for a suppressed cell could be measured by the reciprocal of the width of the uncertainty interval, or, for count data, in terms of the number of integer patterns that "solve" the suppression pattern (ref: Robertson & Ethier (2002)). This leads to more complicated loss functions that require the program to estimate the uncertainty interval as it is being created. Such loss functions are non-linear. Another type of non-linear loss function may arise when one tries to minimize some statistical measure of the difference between the original and modified cell values, such the correlation between old and new cell values. Non-linear loss functions can be handled in meta-heuristic programs or by using certain traditional convex programming methods. To date, there is limited experience with the use of non-linear loss functions in 'protection flow' models. However use of statistical measures may lead to increased use of such loss functions in coming years.

H. The SO now runs the program that implements the optimization model. If the table has no special features one may be able to use general protection software developed by an SO (ref: Argus) which is likely to be free and downloadable or software developed by consultants which is not likely to be free. If the data are complicated and require either pre-processing or protection with complicated requirements (e.g., protecting at the 'company level'), the SO may need to develop its own software. However, the computationally most costly segment of the program can still be done with general purpose packages which the SO-specific program would invoke (e.g., a general linear program solver or an integer program solver).

Evaluation Steps

I. Some methods guarantee that the uncertainty interval for a cell will be at least as large as that requested. For example, network flow for cell suppression runs on simple 2d tables have that property and therefore it is not necessary to run an audit program to check the results. However, network flow programs on 3d tables occasionally fail to provide adequate protection. In that case, an audit run is recommended. If it is discovered that certain uncertainty intervals are not sufficiently wide, the SO can expand the UIs by hand or try running a program based on another method that does guarantee full protection.

J. The SO should assess how vulnerable the protection method is to simple attacks, specifically the midpoint attack. This involves determining what type of UI the table user could easily construct for a given sensitive cell.

K. The SO should assess the overall usefulness of the table after all disclosure protections have been applied. For example, cell suppression creates holes in the table. It takes effort by a user to replace these with the corresponding UI. From there he needs to estimate a single value if he wishes to use the table for either simple purposes or for modeling. By contrast Controlled Tabular Adjustment (CTA) provides single values in each cell immediately. However, with CTA the user might try to construct UIs to determine how representative the published value is of the full range of possible values.

3. Analysis of the Major Steps for Specific Methods

CS = Cell Suppression
CTA= Controlled Tabular Adjustment
VBR=Variable Base Rounding
UI= Uncertainty Intervals

3.1 Cell Suppression (CS)

The application of network flow and linear program models to the finding of cell suppression patterns began in the 1970's. Our analysis refers to certain network flow and linear programming implementations of CS that have been used at the U.S. Census Bureau.

1. Sensitive cells are protected sequentially, i.e., not simultaneously.
2. The order of protection is in descending order of the amount of protection required.
3. Capacities are based on knowledge assumptions which imply intervals of the form [kl*v, ku*v] where kl < 1 and ku > 1. If capacity is a function of the sensitive cell being protected, then adjust capacity accordingly (e.g., protection at 'company level'). This code, which is based on careful analysis of various

cases, is quite complicated (ref: Jewett (1993)). One needs to ensure that kl and ku are large enough to allow for full protection; (e.g., given p, (1-kl) and (ku-1) must be at least (p/100).)

4. Constraints reflect additivity of shafts

5. Cost function: is linear in most implementations ; in integer programming (IP), the cost function is often defined as the sum over all cells of (the cost of the cell * delta(cell)) where delta is a binary variable which equals 1 if the cell is in the cell suppression pattern (has non-zero flow) and equals 0 otherwise. In a linear program (LP), since binary variables are not available, the cost function is often defined as the sum over all cells of (the cost of cell * flow thru cell). Typically the cost of each sensitive is set to 0 since sensitive cells will be suppressed in any pattern. The cost of a non-sensitive cell depends on whether the cell has been selected to be suppressed. After it is selected, its cost is set to 0 and is kept at 0 as protection flow is calculated for the remaining sensitive cells.

6. Incorporating preferences into a CS program. The result of a run of a CS program is a suppression pattern. All sensitive cells need to be suppressed. However, there are typically many possible choices for the non-sensitive cell that are feasible; i.e., satisfy the bounds and constraints. Preferences can be implemented as follows. If one would prefer not to suppress a cell, e.g., a marginal, one can greatly increase its cost; i.e., set cost = k* value where k is large.

3.2 Controlled Tabular Adjustment (CTA)

This is a method that was developed recently (ref: Dandekar and Cox (2002); Cox, Kelly, and Patil (2005)). There are various ways to implement this method; (1) with a Linear Program an Model and LP Solver, (2) with an Integer Programs Model using an IP solver, or (3) with Integer Program Model with a meta-heuristic (tabu search) solver. We have more experience with the latter two implementations; some of our comments below will relate to those. Given a set of sensitive cells and protection intervals for each, CTA is a method for perturbing non-sensitive cells by percentages of the same order as the protection percentages used in the protection intervals. Often a high percentage of the non-sensitive cells must be perturbed ; in other words the modification pattern is dense (with modified cells). The more limited the perturbations are in percentage terms in comparison to the protection percentages, the larger the number of non-sensitive cells that must be perturbed. Additivity is generally preserved. There is an option for preserving marginals.

3.3 Variable Base Rounding (VBR)

This method was developed by Sande (ref: 2003) in recent years. In this description, we take the key ideas from his paper but many of the details may differ from those given in his paper. Suppose a cell value is 44 and the required protection interval (or range) is 44±8 or [36,52] (as determined by some protection rule). In other words, we are requiring that the uncertainty interval be 2-sided and contain [36,52]. In VBR, we use random rounding for two purposes (1) to give an interval (Sande uses the term ‘range’) that is wide enough; i.e., contains the protection interval ; (2) to defend against a midpoint attack by releasing a cell value that is usually at least a few units away from the true value.

In this example, since the numbers are of order 10, we begin by rounding both the true value 44 and the right (and left) width of 8 to base 10; thus 44 rounds to 40 and 8 ‘up-rounds’ to 10 where by ‘up-round’ we mean finding the smallest multiple of the base that is greater than or equal to the given number. For simplicity, we are using conventional rather than random rounding. Next we form the interval that corresponds to a published value of 40 to base 10, viz. [30,50]. This interval is not quite wide enough since it does not contain the protection interval. If it did contain it, we would be done. Since it is not wide enough we must increase the base. Suppose our next larger base is 20. Then we still have 44 rounding to 40 (to base 20) but a value of 40 now corresponds to an interval of 40±20 or [20,60]. Since this ‘rounding interval’ does contain the protection interval, the process terminates and we would publish the value 40 in the cell and somehow indicate that the base is 20. Sande recommends using a system of fonts to indicate which base applies to a given cell value; this scheme has the advantage that it obviates the need for extra characters. We think it might be easier for the table reader simply to see the interval in the form 40±20 or [20,60]. Now the reason why this method is called **variable** base rounding is that one cannot predict which base (10, 20, etc.) will be used for the published cell value, since the base used in a given cell depends on the increasing sequence of bases used in the algorithm and the true cell value. Since the rounding process is applied to each cell independently, it is very likely that a variety of bases will be used in the published table, i.e., some cells will use base 10, some 20, etc. For a given table, a list of potential bases might be given initially; these depend on the order of magnitude of the cell values. If the values are in the range 20 to 500; a base sequence of 10,20,50, ...is reasonable. There are probably many reasonable ways of selecting a sequence of rounding bases.

Recall that this method is designed to protect against a midpoint attack. Rounding almost always provides a symmetric uncertainty interval about the published value; e.g., 40 ± 20 where 40 is the published value. Thus to defend against the midpoint attack, rounding must produce a published value which is often at least some distance (e.g., a few units) from the true value. In the above example, 44 was rounded to 40 in the final base 20; if the true value were 34, it would still be rounded to 40. Thus a table user who assumes the rounded published cell value is the true value will be wrong almost all the time and will be at least several units off most of the time. Many SOs would consider this sufficient protection against a midpoint attack by a table user.

So far we have described the procedure only for sensitive cells. Protection flow can be computed just as in cell suppression, and we can then run an audit program on the table to compute uncertainty intervals for each cell that admits flow. Then we can apply the rounding procedure to the uncertainty interval to get a rounding interval for the cell; this latter interval would be published in either the format Sande suggests or the one we suggest. One might view VBR as a method that is between cell suppression and CTA. Like suppression, the sensitive cells would be protected sequentially; this allows for protection at the company level (a feature which CTA does not currently have). Like suppression, there may be several flows going through a given cell. Publishing a rounding interval that is wide enough to accommodate all these flows allows for a more compact modification pattern than is the case for CTA.

It is natural to ask if additivity is preserved. Of course to answer this, we need to define what we mean by additivity when a subset of the cells are represented not by a single value but by a rounding interval. A desirable property would be for the sum of intervals to include the interval of the sum point. It appears that this property does hold for the VBR intervals. Note that adding rounding intervals is similar to adding random variables in that the uncertainty of a sum of intervals increases as one adds more intervals just as the variance of a sum of random variables increases as one adds more random variables.

3.4 Uncertainty Intervals (UI)

This method is being developed by the author. We are using the short expression 'uncertainty intervals' to represent the more specific expression 'protection flow based uncertainty intervals'. This method is closer in spirit to VBR than to CS or CTA. The idea is that the advantages of using VBR can be achieved more easily computationally using the following procedure. One constructs a (two-sided) flow interval by simply tracking the protection flow through each cell as sensitive cells are protected sequentially as in CS or VBR. For each cell one defines a "flow meter" that keeps track of the maximum and minimum directional flow through the cell. For example, if for 7 sensitive cells, the one-sided protection flows thru cell A had the values, +3, +8, 0,0, 0, -1, -4 then the one-sided protection flow interval would be [- 4, +8]. We call this interval one-sided because normally one "drives" the constraint system by setting the **upper** protection to a desired value. Then, one drives the system by setting the **lower** protection to a desired value; often having the same magnitude. Often this 2nd step is not needed because certain assumptions are made about the capacity of cells that lead to a symmetry of the upper and lower flows. Above we assumed the 7 sensitive cells were protected using their upper protections, p1,....,p7 and led to a one-sided flow interval of [-4,+8]. Then, if the symmetry conditions hold, the set of lower protections, (-p1),....,(-p7), would lead to a second flow interval of [-8, +4]. The union of these two intervals, [-8,+8], is the two-sided flow interval. Finally one constructs the uncertainty interval UI about the value of A (denoted v(A)), by adding v(A) to the 2-sided protection flow interval; this yields [v(A)-8, v(A)+8].

So far, this is likely to be identical to the VBR procedure although Sande (2003) is not explicit about how the intervals (his 'ranges') are formed. Now if we want to protect against a midpoint attack, we need to create an interval that contains the computed flow interval but is often asymmetric with respect to the true value. We believe it is possible to construct a simple algorithm that sometimes 'pads' the flow interval on the left; i.e., creates an interval of the form [v(A)-8-x, v(A)+8] where x is positive or otherwise 'pads' the flow interval on the right'; i.e., creates an interval of the form [v(A)-8, v(A)+8+x]. Actually, one possible algorithm would be the rounding algorithm of the VBR procedure that is described above. If the rounding algorithm is selected, the only difference between VBR and the UI methods would be the format of the published result; in VBR a single value is published with a special font that denotes its rounding base; in UI an interval is given that contains the true value; i.e., two values (the UI endpoints) are given for each cell. In fact, to make it easy for a table user to adjust to using intervals; one might decide to publish the interval in the form: midpoint ± (1/2 * length(UI)). Since tables are increasingly being published in a digital format and released on websites, this extra "bulk" is not significant.

One can construct alternatives to the rounding algorithm that are still fairly simple. It is easy to construct intervals for which the direction and extent of the 'padding' depends on the output from a random generator. However, to make the computations more readily reproducible, it is probably better to construct a deterministic algorithm. One could base it on the final digit on the right end-point; for example if [19,41] were the initial UI (symmetric about the true value 30), one could add 50% of the interval width to the right end-point whenever there is a 1 in the final digit of the right end-point. In this example, we get [19,52] with a mid-point of 35. If the final digit were a 2, one might add 50% to the left ; so an initial interval of [18,42] would be transformed to [6,42] with a midpoint of 24. One could easily specify various extension rules for the other final digits, 0, 3, ...9. We suggest using the last digit because it is deterministic but acts somewhat like a random number generator in that it is hard to predict. However, there are probably a variety of other simple algorithms that would extend the initial flow interval by less than 100% and defend against a midpoint attack. In fact, each agency could decide on the details of the algorithms so that a nice balance of additional uncertainty and protection against the midpoint attack is reached.

An interesting question is how much uncertainty the protection flow based uncertainty interval introduces compared with that from the audit interval. We raise this question because agencies have sometimes considered releasing the audit intervals for each suppressed cell since these intervals could be computed fairly easily by a data intruder who has some facility with linear programming. We wouldn't expect the flow and audit intervals to be equal because the audit program does not use microdata as input and therefore doesn't have the ability to compute capacities other than the simplest case of setting the capacity of a cell to its value. The suppression program does use microdata and can compute complicated capacities such as those use to protect at the 'company level'. Then the question reduces to: how do the lower and upper end-points of the flow interval (with one side extended to protect against a midpoint attack) compare with the corresponding points of the audit interval ?

We constructed a program that outputs those four end-points side by side. We have results for a few small test tables. In those cases, the UIs are often less than 1/2 as wide as the audit intervals. If this result holds for larger, more realistic tables, than it appears that releasing the UI really would provide the user with more information than a suppressed table, while providing adequate protection of sensitive cells including protection against a midpoint attack.

In general, a table user will have no need to add UIs since a UI is supplied for every cell. However, the following relationships do apply. Suppose in the original table there are 3 cells, with values that satisfy: $x1 + x2 = x3$. Suppose we define the operation of addition for intervals as: $[a1, b1] + [a2, b2] = [a1+a2, b1+b2]$. Using this definition, we claim the following property holds: $UI(x3) \leq UI(x1) + UI(x2)$ where '$\leq$' represents set containment.
This latter property we call 'subadditivity' since when $\leq$ and + have their usual meanings for real numbers, and UI is replaced by a function on the reals, this is the standard mathematical definition. In other words, when a user derives the UI for a cell by addition rather than using the UI supplied for the cell, he will, in general, get a larger interval which gives a less precise estimate of the value of interest.

In order to publish an associated traditional table; i.e., one with a single value (rather than an interval) for each cell, we need to construct a representative value for each cell interval. The simplest representative value is the midpoint of the cell's interval. This representative value is acceptable from a disclosure control point of view, providing the midpoint is, in general, not equal to, and not very close, to the true value. This desirable property will hold if the interval has been 'randomly asymmetrized'; i.e., intentionally made asymmetric about the true value in a way that is unpredictable by a user. This can be done in various ways; one such way was described above. In order to ensure additivity of this associated traditional table, it is probably easiest to construct the representative value for each interior cell and then to recompute marginals.

4. Additional Topics (Attacks and Comparison with microdata noise methods)

4.1 Data Intrusion Attacks and Defending Against Them

What attacks does an SO need to be concerned about ?

For example, how should an SO protect against a midpoint attack ?

How asymmetric must an interval be to protect against a midpoint attack ?

Which of the above methods do an adequate job of defending against an attack ?

4.2 Comparison with microdata noise methods

Compare VBR and UI to noise methods such as Evans-

Zayatz-Slanta (EZS) (ref: Evans, et al, 1998) that add noise to microdata and then form tables from the noisy microdata.

VBR and UI advantages:

1. VBR and UI make clear how much uncertainty is associated with each cell value.

2. VBR and UI always provide the required amount of protection, whereas noise methods may provide the required protection only a high percentage of the time.

Advantages of the EZS noise method:

1. Ensures consistency among linked tables without need to backtrack.

2. Easier to program; computationally less costly (i.e., quicker and simpler).

5. Conclusions

All methods in this paper are useful in the right setting. If an agency needs to protect data at the company level, either VBR or its slight generalization UI are acceptable ways to provide more information to the table user than is provided by cell suppression. Both methods provide the required amount of protection. Both clearly indicate to table users which cells are not assigned a single released true value but instead are assigned a range of values in which the true value lies. In the case of VBR, the range is given indirectly with a rounded value to which an interval can be associated by deriving the rounding base from the font in which the cell value is published. In the case of UI, the uncertainty interval is given directly. Thus both methods do not publish a single perturbed (and false) value but rather protect cells that would have been suppressed under cell suppression by providing a range of values. These methods may appeal to some SOs more than CTA. However, if publication of specific perturbed values (rather than intervals) is not considered a negative feature by an SO, and if protection of data at the company level is not required, a SO should consider CTA since it has the convenience of providing a full table of single values that can be used easily by a table user.

References

Argus: Software website for free downloadable software for cell suppression developed under the CASC project funded by the European Union. http://neon.vb.cbs.nl/casc/TAU.html

Cox, Lawrence H., James P. Kelly, Rahul J. Patil, (2005) "Computational Aspects of Controlled Tabular Adjustment: Algorithm and Analysis", in conference proceedings book :The Next Wave in Computing, Optimization, and Decision Technologies, ed. By Bruce L. Golden, S. Raghavan, Edward A. Wasil, publ. Springer

Dandekar, Ramesh A., Lawrence H. Cox, (2002) "Synthetic Tabular Data - An Alternative to Complementary Cell Suppression", unpublished manuscript.

Evans, Tim; Zayatz, Laura; Slanta, John; "Using Noise for Disclosure Limitation Establishment Tabular Data", J. Official Statistics, Vol. 14, No.4, 1998 http://www.jos.nu/Articles/abstract.asp?article=144537

Jewett, Robert (1993), "Disclosure Analysis for the 1992 Economic Census", unpublished Census report 1993.

Massell, Paul B.,"Comparing Statistical Disclosure Control Methods for Tables: Identifying the Key Factors", Proceedings of the JSM2004 http://www.census.gov/srd/sdc/Massell.JSM2004.paper.v3.pdf

Massell, Paul B., "Statistical Disclosure Control for Tables: Determining Which Method to Use" Statistics Canada Conf. on Statistical Methodology, Oct. 2003 http://www.census.gov/srd/sdc/Massell%20StatCan%20Meth%20Symp%20english.pdf

Robertson, Dale A., Richard Ethier (2002) "Cell Suppression: Experience and Theory", in Inference Control in Statistical Databases, Josep Domingo-Ferrer (ed.), Springer Lecture notes in Computer Science ,(vol. LNCS2316)

Sande, Gordon, "A Less Intrusive Variant on Cell Suppression to Protect the Confidentiality of Business Statistics", FCSM 2003 Conference.Proceedings. http://www.fcsm.gov/03papers/Sande.pdf