INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Shortest-Paths Heuristic for Statistical Data Protection in Positive Tables

Jordi Castro,

INFORMS Journal on Computing
Vol. 19, No. 4, Fall 2007, pp. 520–533
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1904 | 0520

DOI 10.1287/ijoc.1060.0185

A Shortest-Paths Heuristic for Statistical Data Protection in Positive Tables

Jordi Castro
Department of Statistics and Operations Research, Universitat Politècnica de Catalunya,
08034 Barcelona, Catalonia, Spain, jordi.castro@upc.edu

National statistical agencies (NSAs) routinely release large amounts of tabular information. Prior to dissemination, tabular data need to be processed to avoid disclosure of individual confidential information. Cell suppression is one of the most widely used techniques by NSAs. Optimal procedures for cell suppression are computationally expensive with large real-world data sets, so heuristic procedures are used. Most heuristics for positive tables (i.e., cell values are nonnegative) rely on the solution of minimum-cost network-flows subproblems. A very efficient heuristic based on shortest paths already exists, but it is only appropriate for general tables (i.e., cell values can be either positive or negative), whereas in practice most tables are positive. We present a method that sensibly combines and improves previous approaches, overcoming some of their drawbacks: it is designed for positive tables and requires only the solution of shortest-path subproblems—therefore being much more efficient than other network-flows heuristics. We report extensive computational experience in the solution of randomly generated and real-world instances, comparing the heuristic with alternative procedures. The results show that the method, currently included in a software package for statistical data protection, fits NSA needs: it is extremely efficient and provides good solutions.

Key words: statistical disclosure control; cell-suppression problem; linear programming; network optimization; shortest paths
History: Accepted by Amit Basu, Area Editor for Knowledge and Data Management; received November 2004; revised June 2005, November 2005; accepted March 2006. Published online in *Articles in Advance* July 25, 2007.

1. Introduction

The field of statistical disclosure control is composed of a set of tools for preserving confidentiality (i.e., individual information) when releasing statistical data, either as microfiles—files of records, each record providing the values for a set of variables of an individual—or as tables that cross two or more variables. An example of disclosure is illustrated in Figure 1: Table (a) shows the average salary by age interval and ZIP code, while table (b) shows the number of individuals for the same variables. If there were only one individual in ZIP code z_2 and age interval 51–55, then any external attacker would know that the salary of this person is 40,000€. For two individuals, any of them could deduce the salary of the other, becoming an internal attacker. In that example, cells $(51–55, z_2)$ of both tables are sensitive and their values should be protected. There are rules for identification of sensitive cells; a recent discussion about them is Domingo-Ferrer and Torra (2002). Good introductions to the state of the art in statistical disclosure control are Willenborg and de Waal (2000), Domingo-Ferrer (2002) and Domingo-Ferrer and Torra (2004).

Cell suppression is one of the most widely used techniques by national statistical agencies (NSAs) for protection of confidential tabular data. Given a list of cells to be protected, the purpose of the cell-suppression problem (CSP) is to find a pattern of additional (a.k.a. complementary or secondary) cells to be suppressed to avoid disclosure of the sensitive ones. This pattern of suppressions is determined under some criteria as, e.g., minimum number of suppressions, or minimum value suppressed.

CSP was shown to be NP-hard in Kelly et al. (1992). This has motivated most of the former approaches to focus on heuristic methods for approximate solutions (e.g., Gusfield 1988, Kelly et al. 1992, Carvalho et al. 1994, Cox 1995, Dellaert and Luijten 1999, Giessing and Repsilber 2002). In this paper we present a new heuristic approach for positive tables (i.e., cell values are greater than or equal to zero). It relies on the solution of shortest-path subproblems, and is significantly more efficient than most of the alternative methods. A recent exact procedure based on state-of-the-art mixed integer linear programming (MILP) techniques was able to solve to optimality nontrivial CSP instances (Fischetti and Salazar 2001). The main

		z_1	z_2	
⋮	…	…	…	…
51–55	…	38000€	40000€	…
56–60	…	39000€	42000€	…
⋮	…	…	…	…

(a)

		z_1	z_2	
⋮	…	…	…	…
51–55	…	20	1 or 2	…
56–60	…	30	35	…
⋮	…	…	…	…

(b)

Figure 1 Example of Disclosure in Tabular Data

inconvenience of such an approach from the practitioner point of view is that the solution of very large instances—with possibly millions of cells—can result in prohibitive execution times, as shown in the computational results of this work. In practice, tabular data protection is the last stage of the "data cycle," and, in an attempt to meet publication deadlines, NSAs require methods that find fast solutions to protect large tables (Dandekar 2003). It is noteworthy that improvements in new heuristics also benefit the exact procedure, since they provide a fast, hopefully good, feasible starting point.

Although most current heuristics for CSP are based on network optimization, some recent approaches have been devised for obtaining fast solutions to large problems. Among them we find the hypercube method, developed by Destatis (German NSA) (Giessing and Repsilber 2002), that focuses on geometric considerations. Although very efficient, this approach has two drawbacks: it may report nonfeasible solutions (i.e., some cells remain unprotected), and it provides patterns with a large number of secondary cells or value suppressed, compared to alternative approaches (i.e., it suffers from over-suppression). Network-flows heuristics for CSP usually exploit the table information more efficiently and provide better results. The approach described in de Wolf (2002), developed by the Centraal Bureau voor de Statistiek (Dutch NSA), decomposes large tables into smaller ones, independently protecting them at each iteration of a backtracking procedure. This approach also does not guarantee feasible solutions. Our shortest-paths heuristic always reports feasible solutions and, from the computational results with real-world instances, is faster than the approach of de Wolf (2002), and provides better solutions than the hypercube method.

There is a fairly extensive literature on network-flows methods for CSP. For positive tables, they rely on the formulation of minimum-cost network-flows subproblems (Kelly et al. 1992, Cox 1995, Castro 2002). Such approaches have been successfully applied in practice (Jewett 1993). Those heuristics require the table structure to be modeled as a network, which in general can be accomplished only for two-dimensional tables with at most one hierarchical variable (see Section 3 for details). Although minimum-cost network-flows algorithms are fast compared to the equivalent linear-programming formulations (Ahuja et al. 1993, Chaps. 9–11), they still require large execution times for large tables (Castro 2002). Instead, the approach suggested in Carvalho et al. (1994) consists of formulating shortest-path subproblems, which can be solved very efficiently through specialized algorithms (Ahuja et al. 1993, Chaps. 4, 5). The main drawback of that approach based on shortest paths is that it could only be applied to general tables (i.e., cell values can be either positive or negative), which are less common in practice.

To avoid the above problems of current network-flows heuristics (namely, the efficiency of those based on minimum-cost flows subproblems, and the limitation of those based on shortest paths for positive tables) we present a new method that sensibly combines and improves ideas from previous approaches (mainly Kelly et al. 1992, Carvalho et al. 1994, and Cox 1995). The resulting method applies to positive tables and formulates shortest-path subproblems. As shown by the computational results, it is much faster than heuristics based on minimum-cost network problems for large tables. The new approach has been included in the τ-Argus package (Hundepool 2004) in the scope of the project IST-2000-25069 CASC (Computational Aspects of Statistical Confidentiality), funded by the European Union. That project involved 14 institutions from five European countries, including the NSAs of Catalonia, Germany, Italy, The Netherlands, Spain, and The United Kingdom.

This work extends Castro (2004), which outlined the method and reported some preliminary results. In particular, compared with the early version, the current work presents the infeasibility-recovery procedure (in Section 4.5), which guarantees robustness of the method; the computational complexity (Section 4.9), which shows the efficiency of the approach compared to previous ones; and the lower-bounding procedure (Section 5), which can be used to estimate the quality of the solution found by the heuristic. Moreover, unlike the early version, this work presents computational results, including those obtained for real confidential data, reported by the NSAs of Germany and The Netherlands. These results prove the effectiveness of the approach.

Section 2 outlines the formulation of CSP. Section 3 briefly shows how to model a two-dimensional table with at most one hierarchical dimension as a network. Section 4 outlines the antecedents and presents the new shortest-paths heuristic. Section 5 presents an improved version of the lower-bounding procedure introduced in Kelly et al. (1992). Finally, Section 6 reports the computational experience in solving randomly generated and real-world instances, showing the effectiveness of the method.

2. Formulation of CSP

Given a positive table (i.e., a set of cells $a_i \geq 0$, $i = 1, \ldots, n$, satisfying m linear relations $Aa = b$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$), a set $\mathcal{P}$ of primary sensitive cells to be protected, and upper and lower protection levels upl_p and lpl_p for each primary cell $p \in \mathcal{P}$, the purpose of CSP is to find a set $\mathcal{S}$ of additional secondary cells whose suppression guarantees that, for each $p \in \mathcal{P}$,

$$\underline{a}_p \leq a_p - lpl_p \quad \text{and} \quad \bar{a}_p \geq a_p + upl_p, \tag{1}$$

$\underline{a}_p$ and $\bar{a}_p$ being defined as

$$\begin{aligned} \underline{a}_p = \min\ & x_p \\ \text{s.t. } & Ax = b \\ & x_i \geq 0 \quad i \in \mathcal{P} \cup \mathcal{S} \\ & x_i = a_i \quad i \notin \mathcal{P} \cup \mathcal{S} \quad \text{and} \\ \bar{a}_p = \max\ & x_p \\ \text{s.t. } & Ax = b \\ & x_i \geq 0 \quad i \in \mathcal{P} \cup \mathcal{S} \\ & x_i = a_i \quad i \notin \mathcal{P} \cup \mathcal{S}. \end{aligned} \tag{2}$$

$\underline{a}_p$ and $\bar{a}_p$ in (2) are the lowest and greatest possible values that can be deduced for each primary cell from the published table, once the entries in $\mathcal{P} \cup \mathcal{S}$ have been suppressed. The lower and upper protection levels lpl_p and upl_p determine the length of the interval $[a_p - lpl_p, a_p + upl_p]$. Imposing (1), the lowest and greatest values deduced for each primary cell from the published table are out of the above interval, making the cell safe. In practice the protection levels are a fraction of the cell value (e.g., 15% and 30% are typical values), but for small cells, whose protection levels can be greater than the cell value. CSP can thus be formulated as an optimization problem of minimizing some function that measures the cost of suppressing additional cells subject to conditions (1) being satisfied for each primary cell. Unlike Kelly et al. (1992) we do not consider a sliding protection level spl_p for primary cell p such that $a_p \in [\underline{a}_p, \bar{a}_p]$ and $\bar{a}_p - \underline{a}_p \geq spl_l$, as this is not a requirement for end users who are mainly interested in lower and upper protection levels.

CSP was first formulated in Kelly et al. (1992) as a large MILP problem. For each cell a_i a binary variable y_i, $i = 1, \ldots, n$ is considered. y_i is set to 1 if the cell is suppressed, otherwise is 0. For each primary cell $p \in \mathcal{P}$, two auxiliary vectors $x^{l,p} \in \mathbb{R}^n$ and $x^{u,p} \in \mathbb{R}^n$ are introduced to impose, respectively, the lower and upper protection requirement of (1). These vectors represent cell deviations (positive or negative) from the original a_i values. The resulting model is

$$\begin{aligned} \min\ & \sum_{i=1}^{n} w_i y_i \\ \text{s.t. } & \left.\begin{aligned} & Ax^{l,p} = 0 \\ & -a_i y_i \leq x_i^{l,p} \leq M y_i \quad i = 1, \ldots, n \\ & x_p^{l,p} \leq -lpl_p \\ & Ax^{u,p} = 0 \\ & -a_i y_i \leq x_i^{u,p} \leq M y_i \quad i = 1, \ldots, n \\ & x_p^{u,p} \geq upl_p \end{aligned}\right\} \forall p \in \mathcal{P} \\ & y_i \in \{0, 1\} \quad i = 1, \ldots, n, \end{aligned} \tag{3}$$

where w_i is the information loss associated with cell a_i, usually set to $w_i = a_i$ (minimize the overall suppressed value) or $w_i = 1$ (minimize the overall number of suppressed cells). The inequality constraints of (3) with both right- and left-hand sides impose bounds on $x_i^{l,p}$ and $x_i^{u,p}$ when $y_i = 1$ (M being a large value), and prevent deviations in nonsuppressed cells (i.e., $y_i = 0$). Clearly, the constraints of (3) guarantee that the solutions of the linear programs (2) will satisfy (1). (3) gives rise to a MILP problem of n binary variables, $2n|\mathcal{P}|$ continuous variables, and $2(m + 2n)|\mathcal{P}|$ constraints. This problem is very large even for tables of moderate size and number of primary cells. For instance, for a table of 8,000 cells, 800 primaries, and 4,000 linear relations, we obtain a MILP with 8,000 binary variables, 12,800,000 continuous variables, and 32,000,000 constraints.

If matrix A could be modeled as a network, we would find a feasible nonoptimal (but hopefully good) solution to (3) through a succession of network-flows subproblems for each primary cell. This is the basis of any network-flows heuristic for CSP. Different network subproblems (which possibly mean different solution algorithms) give rise to alternative heuristics. Before presenting the heuristic in Section 4 we first show the particular networks considered for two-dimensional tables with and without a hierarchical variable.

3. Modeling Tables as Networks

The linear relations of a two-dimensional table of $r+1$ rows and $c+1$ columns a_{ij}, $i = 1, \ldots, r+1$, $j = 1, \ldots, c+1$ (last row and column are marginal) are

$$\begin{aligned} \sum_{j=1}^{c} a_{ij} &= a_{i,c+1} \quad i = 1, \ldots, r \\ \sum_{i=1}^{r} a_{ij} &= a_{r+1,j} \quad j = 1, \ldots, c. \end{aligned} \tag{4}$$

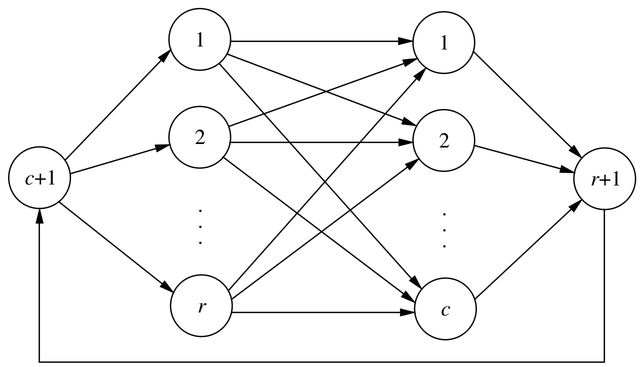

Figure 2 Network Representation of an $(r+1) \times (c+1)$ Table

It is well-known (see, e.g., Ahuja et al. 1993, Chaps. 6, 8) that (4) can be modeled as the network of Figure 2. Arcs are associated with cells and nodes with constraints. The number of undirected arcs n (i.e., the number of cells) and the number of nodes (i.e., the number of linear relations) of the network are

$$n = (r+1)(c+1) \qquad m = r+c+2. \tag{5}$$

(Indeed, the number of linear relations is $r+c+1$, since one is redundant.)

Two-dimensional tables with one hierarchical variable can also be modeled as a network, and are of great practical interest (Cox and George 1989, Jewett 1993). Before providing a general formulation for hierarchical tables, we first illustrate them through a small example. Without loss of generality we will assume that the hierarchical variable appears in rows.

Figure 3 shows a hierarchical table with three subtables. Row R_2 of subtable T_1 has a hierarchical structure: $R_2 = R_{21} + R_{22}$. The decomposition of R_2 is detailed in T_2. And row R_{21} of subtable T_2 is also hierarchical; T_3 shows its structure. For instance, rows R_i could correspond to regions, R_{2i} to cities, and R_{21i} to ZIP codes. Although all the subtables have the same number of rows in the example, this is not required in general. However, the number of columns must be the same for all the subtables; otherwise, we would not preserve the hierarchical structure in only one dimension. Clearly, every subtable can be modeled as a network similar to that of Figure 2.

In general, a hierarchical table—with one hierarchical variable—can be represented by a set of t two-dimensional $(r_k+1) \times (c+1)$ subtables, $k = 1, \ldots, t$,

T_1

	C_1	C_2	C_3
R_1	5	6	11
R_2	10	15	25
R_3	15	21	36

T_2

	C_1	C_2	C_3
R_{21}	8	10	18
R_{22}	2	5	7
R_2	10	15	25

T_3

	C_1	C_2	C_3
R_{211}	6	6	12
R_{212}	2	4	6
R_{21}	8	10	18

Figure 3 Two-Dimensional Table with Hierarchical Rows Made Up of Three $(2+1) \times (2+1)$ Subtables, T_1, T_2 and T_3

plus additional equality side constraints. r_k+1 is the number of rows of subtable k, while $c+1$ is the number of columns for all subtables. The additional equality side constraints impose that common cells of two subtables must have the same values, and can be defined through the set of three-dimensional vectors

$$\mathcal{E} = \{(o_k, u_k, v_k),\ k = 1, \ldots, t-1\}. \tag{6}$$

Vector (o_k, u_k, v_k) indicates that row o_k of table u_k is decomposed in table v_k. For the above example, we have $\mathcal{E} = \{(R_2, T_1, T_2), (R_{21}, T_2, T_3)\}$. Therefore, $\mathcal{E}$ provides the particular structure of the hierarchical tree. Given $\mathcal{E}$, the table relations can be written as

$$\begin{aligned} &\sum_{j=1}^{c} a_{ij}^k = a_{i,c+1}^k \quad k = 1, \ldots, t \quad i = 1, \ldots, r_k \\ &\sum_{i=1}^{r_k} a_{ij}^k = a_{r_k+1,j}^k \quad k = 1, \ldots, t \quad j = 1, \ldots, c \\ &a_{o_k,j}^{u_k} = a_{r_k+1,j}^{v_k} \quad k = 1, \ldots, t-1 \quad j = 1, \ldots, c. \end{aligned} \tag{7}$$

Equations (7) give rise to a network-flow feasibility problem made of t subnetworks, one per table, with $(t-1)c$ side constraints. For $t=1$ no side constraints are considered, and (7) represents a single two-dimensional table.

Equations (7) can be modeled as a network. A fast algorithm—linear in the number of subtables—was described in Castro (2004). Applying this algorithm to the example of Figure 3, we obtain the network of Figure 4. The number of undirected arcs n and nodes m of the network associated with a hierarchical table is

$$n = (c+1)\left(1 + \sum_{k=1}^{t} r_k\right), \qquad m = 2 + tc + \sum_{k=1}^{t} r_k, \tag{8}$$

and (5) is a particular case of (8) for $t=1$.

More complex tables, such as two-dimensional tables with hierarchies in both variables, do not accept a pure network representation. A partial proof of this result, valid for the network formulation of the complete controlled rounding problem, was given in Cox and George (1989). However they can be modeled as a network with additional side constraints. Since the number of side constraints can be very large, general state-of-the-art dual simplex implementations (Bixby 2002) can outperform specialized algorithms for network flows with side constraints (Castro and Nabona 1996). Network-flows heuristics for CSP will, in general, not perform well in these situations.

4. Antecedents and the Shortest-Paths Heuristic

We first present a common framework for network-flows heuristics for CSP. In the next three subsections,

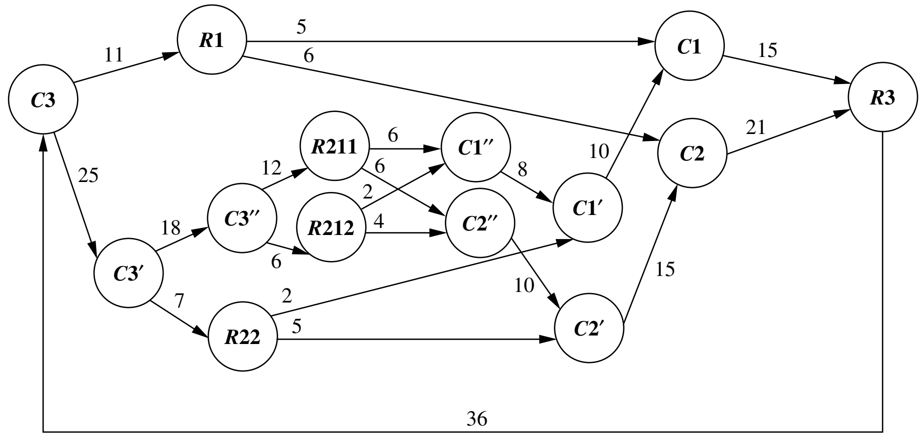

Figure 4 Network from the Example of Figure 3. Nodes with the Same Name in Different Tables are Marked with "′" and "″"

we outline the methods of Kelly et al. (1992), Carvalho et al. (1994), and Cox (1995). We finally introduce the new heuristic, stating the common elements and differences with the previous approaches, and discussing some of its main features.

4.1. Framework for Network-Flows Heuristics

After modeling the table as a network, for each cell a_i two arcs $x_i^+ = (s, t)$ and $x_i^- = (t, s)$ are created, which are respectively related to increments and decrements of the cell value. We use the notation $x = (s, t)$ for an arc x with source and target nodes s and t. Forward arcs x^+ are clockwise oriented, and appear in Figures 2 and 4. Backward arcs x^- are obtained by changing the direction of the arrows of the arcs depicted in Figures 2 and 4.

A feasible solution to (3) can be obtained by solving two minimum-cost network-flows problems for each primary cell as follows. Given the primary cell $p \in \mathcal{P}$, let us consider its forward arc $x_p^+ = (s, t)$ in the network. In the first minimum-cost network-flows problem, associated with the upper protection level, we send upl_p units of flow from t to s through the network excluding arc $x_p^- = (t, s)$. We obtain an augmenting cycle. The cells associated with the arcs in this augmenting cycle are suppressed (note that some of these cells can be primary or secondary cells suppressed in the augmenting cycle for a previous primary). The arc costs, to be discussed later, are chosen to reduce the number of new suppressions. In the second minimum-cost network-flows problem, associated with the lower protection level, we proceed as before, now sending lpl_p units of flow from s to t through the network excluding arc $x_p^+ = (s, t)$. The above procedure is successively repeated for all the primary cells. We finally obtain a set $\mathcal{S}$ of secondary suppressed cells that contains all the nonprimary cells associated with arcs in augmenting cycles. It is now clear that if we solve the two problems (2), the value a_p of the primary cell p can be decreased and increased by respectively lpl_p and upl_p units by readjusting some of the cells in $\mathcal{P} \cup \mathcal{S}$, to satisfy (1).

4.2. The Heuristic of Kelly et al. (1992)

The approach of Kelly et al. (1992) combined the two minimum-cost network-flow problems discussed in Section 4.1 in a single problem. For each primary cell $p \in \mathcal{P}$ an overall flow of $spl_p = lpl_p + upl_p$ is sent through both networks. spl_p is the sliding protection level of primary p. A fraction of spl_p is sent from t to s in one network (the "upper protection" network) and the remaining flow is sent from s to t in the other network (the "lower protection" network). Unlike our approach, this does not satisfy the upper and lower protection requirements, just the sliding protection ones. The main steps of this procedure are shown in Figure 5. $\mathcal{P}$ and $\mathcal{S}$ in the algorithm denote respectively the sets of primary and secondary cells. This heuristic included a clean-up procedure, which is performed at the end of the protection stage. The clean-up is computationally very expensive and it has not been introduced in our heuristic. The approach also computes an initial set of suppressions through a lower-bounding procedure. Such a procedure is discussed and improved for our heuristic in Section 5.

4.3. The Heuristic of Carvalho et al. (1994)

This heuristic was designed for general tables (cells are allowed both positive and negative values). Neither the cell values nor the lower and upper protection levels are required for performing the protection of the table, only the cell positions. In practice this means that infinite upper or lower protection levels are provided for protected cells. This is a consequence of the lack of lower and upper bounds for cell values.

Unlike the heuristics of Sections 4.2 and 4.4, this procedure does not perform minimum-cost network-flows computations, but rather shortest-paths ones. For each primary cell p associated with the arc $x_p^+ = (s, t)$, the heuristic computes the shortest path from t

Algorithm *Heuristic of Section 4.2* (*Table*, $\mathcal{P}$, $spl = lpl + upl$)
$\mathcal{S} = \varnothing$;
Create network, made of two subnetworks (lower and upper protection)
for_each $p \in \mathcal{P}$ **do**
  Set arc costs, using the source and target of primary arc $x_p^+ = (s, t)$;
  Solve minimum-cost network-flows problem;
  $\mathcal{V} = \{$cells associated with arcs with positive flows$\}$;
  $\mathcal{S} := \mathcal{S} \cup \mathcal{V} \backslash \mathcal{P}$;
end_for_each
Clean-up procedure;
Return: $\mathcal{S}$;
End_algorithm

Figure 5 **Outline of Kelly et al. (1992) Heuristic**

to s (or s to t, since we deal with general tables). Costs of arcs are related to the cell values, or some function of the cell values. The cells associated to arcs in the shortest path are protected and considered secondary suppressions. If a primary cell appears in the shortest path, it is protected. The algorithm is outlined in Figure 6. $\mathcal{P}'$ in this algorithm is the current set of unprotected primary cells. The main benefit of this heuristic is that shortest-paths computations are much more efficient than minimum-cost network-flows ones. On the other hand, its drawback is that positive tables cannot be protected with this technique. We adopt the idea of using shortest-paths computations, in combination with the methods of Sections 4.2 and 4.4, thus extending shortest-paths heuristics to positive tables.

4.4. The Heuristic of Cox (1995)

Unlike the framework of Section 4.1, this heuristic protected each primary cell $p \in \mathcal{P}$ through a sequence of minimum-cost network-flows subproblems, instead of a single one. Assigning appropriate costs and capacities, a minimum-cost cycle is computed. The arc $x_p^+ = (s, t)$ of the primary $p \in \mathcal{P}$ is forced to belong to the cycle. The cells associated with arcs in the cycle that are not primary cells are considered secondary cells. Defining γ as the minimum of the values of cells associated with arcs in the cycle, we can guarantee an upper and lower protection of γ for p. If γ is greater than lpl_p and upl_p, cell p is protected. Otherwise we look for additional cycles until the cell is protected. Figure 7 outlines this procedure.

Algorithm *Heuristic of Section 4.3* (*Table*, $\mathcal{P}$)
$\mathcal{S} = \varnothing$; $\mathcal{P}' = \mathcal{P}$;
for_each $p \in \mathcal{P}'$ **do**
  Find source and target nodes of primary arc $x_p^+ = (s, t)$;
  Set arc costs;
  Compute the shortest path SP from t to s;
  $\mathcal{T} = \{$cells associated with arcs $\in SP\}$;
  $\mathcal{S} := \mathcal{S} \cup \mathcal{T} \backslash \mathcal{P}$;
  $\mathcal{P}' := \mathcal{P}' \backslash \mathcal{T}$;
end_for_each
Return: $\mathcal{S}$;
End_algorithm

Figure 6 **Outline of Carvalho et al. (1994) Heuristic**

Algorithm *Heuristic of Section 4.4* (*Table*, $\mathcal{P}$, *upl*, *lpl*)
$\mathcal{S} = \varnothing$;
for_each $p \in \mathcal{P}$ **do**
  $m_p = \max\{lpl_p, upl_p\}$;
  while $m_p > 0$ **do**
    Set arc costs and capacities;
    Solve minimum-cost network-flow subproblem;
    $\mathcal{V} = \{$cells associated with arcs in cycle$\}$;
    $\mathcal{S} := \mathcal{S} \cup \mathcal{V} \backslash \mathcal{P}$;
    Compute $\gamma = \min\{a_l : l \in \mathcal{V}\}$;
    $m_p := m_p - \gamma$;
  end_while
end_for_each
Return: $\mathcal{S}$;
End_algorithm

Figure 7 **Outline of Cox (1995) Heuristic**

The heuristic of Figure 7 protects each primary through a sequence of subproblems as does our heuristic in Section 4.5. The motivation of Cox (1995) for this sequence of subproblems was to better approximate the original combinatorial formulation of CSP. We have a somewhat different motivation, and focus on efficiency. The heuristic of Section 4.5 solves a sequence of shortest-path subproblems, instead of minimum-cost network-flows ones, thus combining the methods of Sections 4.3 and 4.4. This is instrumental, and allows the efficient protection of current large tables managed by NSAs (an impossible mission using minimum-cost network-flows-based approaches). Other main improvements of the heuristic of Section 4.5 compared to that of Figure 7 are as follows:

- The heuristic of Figure 7 can report a subproblem as infeasible, which does not mean infeasibility of the CSP. The heuristic of Section 4.5 includes an infeasibility-recovery procedure for this purpose, which guarantees robustness of the approach.
- The new heuristic deals separately with the upper and lower protection, and updates not only the protection offered by the shortest path to the current primary, but also to other primary cells. This enhances the level of protection provided by each subproblem, significantly reducing the number of shortest-paths computations in some instances, and even allowing the solution of problems with protection levels larger than cell values. This is discussed in Section 4.6.
- Arc costs of the new heuristic are computed following the stratification suggested in Cox (1995), but with slightly different values. In theory this may provide slightly worse protection patterns (in practice the differences are not significant). However, the computation of arc costs, which is the most expensive step

of the heuristic, is more efficient. This is discussed in Section 4.7.

It is also worth noting that no computational experimentation was reported for the heuristic of Cox (1995).

4.5. The Shortest-Paths Heuristic for Positive Tables

The main inconvenience of the approaches of Sections 4.2 and 4.4 is that minimum-cost network-flows subproblems must be solved. As in the approach of Section 4.3, we replace these subproblems by shortest-path computations. In practice no more than a few shortest paths are required for each primary cell. This dramatically reduces the overall running time. Although it cannot be guaranteed that the sequence of shortest paths will always protect the primary cell, such lack of protection occurs only in rare situations. In particular, it never happened in our experiments with either random or real data. However, even in this case we can switch, for this primary cell, to the general framework of Section 4.1. Therefore, the heuristic is always guaranteed to produce a (hopefully good) feasible solution, if one exists.

Figure 8 shows the main steps of the heuristic. It combines some of the ideas in the algorithms of Figures 5–7. It updates the set of secondary cells $\mathcal{S}$, and two vectors *Clpl* and *Cupl* with the current lower and upper protection values of all the primaries. The heuristic performs one major iteration for each primary cell $p \in \mathcal{P}$ (lines 2–37 of Figure 8), and, unlike previous approaches, deals separately with the lower and upper protection values (lines 4–36). If not already done by previous primaries, p is protected through one or possibly several minor iterations (lines 6–35). At each minor iteration we first set the arc costs (see Section 4.7 below). Arcs related to cells that cannot be used are assigned a very large cost; arcs related to primary or already suppressed cells are assigned a low favorable cost. The arc costs are the only information to be updated for the network, unlike previous approaches based on minimum-cost network-flows problems, which also modified node injections and arc bounds. A shortest path from t to s is computed, where $x_p^+ = (s, t)$, and arc $x_p^- = (t, s)$ is assigned a very large cost (so it will not be used). The set $\mathcal{S}$ of secondary cells is updated with the cells associated with arcs in the shortest path (line 32). To avoid the solution of unnecessary shortest-path subproblems for following primaries, we update not only the protection levels of p, but also of all the primary cells in the shortest path (line 33). This is a significant improvement compared to previous heuristics. If several shortest-path problems are needed for p (lines 6–35), cells in previously computed shortest paths for this primary must not be used (otherwise we cannot guarantee the protection of the cell). To this end,

```
Algorithm Shortest-paths Heuristic for CSP (Table, 𝒫, upl, lpl)
 1  𝒮 = ∅; Clpl_i = 0, Cupl_i = 0, i ∈ 𝒫;
 2  for_each p ∈ 𝒫 do
 3    Find source and target nodes of primary arc x_p^+ = (s, t);
 4    for_each type of protection level ∗ ∈ {lpl, upl} do
 5      𝒯𝒯 = ∅; 𝒰 = ∅;
 6      while (C∗_p < ∗_p) do
 7        Set arc costs;
 8        Compute the shortest path SP from t to s;
 9        if SP is empty then
10          //we have to solve two network-flows problems
11          //before reporting this CSP instance as infeasible
12          𝒯𝒯 = ∅;
13          𝒮 := 𝒮\𝒰;
14          Set arc costs, capacities and zero node injections;
15          for_each type of protection level ∗∗ ∈ {lpl, upl} do
16            if (∗∗ = lpl) then
17              Set supply upl_p at node t and demand upl_p
                  at node s;
18            else
19              Set supply lpl_p at node s and demand lpl_p
                  at node t;
20            end_if
21            Solve minimum-cost network-flows problem;
22            if problem is infeasible then
23              Return: this CSP instance is infeasible;
24            end_if
25            𝒱 = {cells associated with arcs with positive
                  flows};
26            𝒮 := 𝒮 ∪ 𝒱 \ 𝒫;
27          end_for_each
28          go to line 37 for next primary;
29        end_if
30        𝒯 = {cells associated with arcs ∈ SP};
31        𝒰 := 𝒰 ∪ (𝒯\(𝒮 ∪ 𝒫));
32        𝒮 := 𝒮 ∪ 𝒯\𝒫;
33        Update Clpl_i and Cupl_i, i ∈ (𝒫 ∩ 𝒯) ∪ {p};
34        𝒯𝒯 := 𝒯𝒯 ∪ 𝒯;
35      end_while
36    end_for_each
37  end_for_each
38  Return: 𝒮;
End_algorithm
```

Figure 8 Shortest-Paths Heuristic for CSP in Positive Tables

$\mathcal{TT}$ in Figure 8 maintains the list of cells already suppressed for the protection of p. Arcs of cells in $\mathcal{TT}$ are assigned a very large cost in line 7.

If the cardinality of $\mathcal{TT}$ significantly increases (i.e., we discard a large number of arcs for the protection of this cell) eventually we may not be able to compute a shortest path (line 9). This means that either the instance is infeasible or that the sequence of shortest paths failed to protect the primary cell. Before concluding the instance is infeasible, we switch to the minimum-cost network-flows approach of Section 4.1 (lines 10–28). The secondary cells specifically needed for the protection of p, stored in $\mathcal{U}$ (lines 5 and 31), are removed from $\mathcal{S}$ (line 13). The two minimum-cost network-flows problems, for the lower and upper protection levels respectively (lines 15–27), determine a set of secondary cells that protect primary p. If there is no solution to at least one of these two network-flows problems, the CSP instance is infeasible. In all the computational experiments of Section 6, the algorithm never entered lines 10–28. If we want these lines to be executed, we can impose very large upper protection levels to some subset of primary cells. However, this is meaningless in practice, unless the cell has a relatively small value. For instance, we randomly generated some 10×10 two-dimensional instances with very large upper protection levels. A solution was obtained through lines 10–28, but it suppressed 85% of the cells of the table. Indeed, this is the only way to guarantee such large protection levels. Aside from the above exceptional and nonmeaningful situations, practical and real tables (i.e., with a large number of cells and upper protection levels less than the cell value) will likely never be exposed to lines 10–28 of the algorithm.

We next discuss some of the relevant points of the heuristic.

4.6. Protection Provided by the Shortest Path

The shortest path SP from t to s is a list of l arcs $x^*_{i_1} - x^*_{i_2} - \cdots - x^*_{i_l}$, * being $^+$ or $^-$ depending on the arc orientation, such that $x^*_{i_1} = (t, t_{i_1})$, $x^*_{i_l} = (s_{i_l}, s)$, and $x^*_{i_j} = (t_{i_{j-1}}, s_{i_{j+1}})$ for all $j = 2, \ldots, l-1$, and $t = s_{i_1}$, $s = t_{i_l}$, and $s_{i_j} = t_{i_{j-1}}$ for all $j = 2, \ldots, l$. $\mathcal{T} = \{i_1, \ldots, i_l\}$ is the set of indexes of cells associated with the arcs in the shortest path. Defining

$$\gamma = \min\{a_p, a_{i_j} \colon i_j \in \mathcal{T}\}, \tag{9}$$

we can send a flow γ through the shortest path in either direction. This means that we can increase or decrease a_p by γ without affecting the feasibility of the table. If $\gamma > \max\{lpl_p, upl_p\}$, it follows from (2) that this cell is protected by this shortest path. This is similar to the approach of Section 4.4.

However, the heuristic exploits the information provided by the shortest path even better. It separately computes

$$\begin{aligned} \gamma^+ &= \min\{a_p, a_{i_j} \colon x^+_{i_j} \in SP\}, \\ \gamma^- &= \min\{a_{i_j} \colon x^-_{i_j} \in SP\}. \end{aligned} \tag{10}$$

If there is no arc $x^-_{i_j}$ in SP, then $\gamma^- = \infty$. γ^+ gives the amount cell p can be decreased without obtaining a negative cell. It is thus the lower protection of p provided by this shortest path. Analogously, the upper protection is provided by γ^-. That permits the lower and upper levels to be updated separately and with different protection values. One immediate benefit of this procedure is that the heuristic can deal with upper protection values greater than the cell value (i.e., $upl_p > a_p$). Such large protections are used for very small cell values. For instance, if only arcs $x^+_{i_j}$ appear in SP, it is possible to increase the value of cell p infinitely without compromising the feasibility of the table. Indeed, in this case the upper protection level provided by the heuristic is $\gamma^- = \infty$. This cannot be done by computing only (9). Current protection levels $Clpl$ and $Cupl$ of p and primary cells in $\mathcal{T}$ are updated using (10) in line 33 of Figure 8.

4.7. Arc Costs

The behavior of the heuristic is governed by the costs of arcs x^+_i and x^-_i associated with cells a_i. Arcs not allowed in the shortest path are assigned a very large cost. This includes arcs associated with zero cells: the values of such cells are usually known by any attacker and can not be used for protection (e.g., the number of persons 5 years old with an average salary between 30,000 and 40,000€ is clearly 0). For the remaining arcs, as suggested in the heuristic of Cox (1995), costs are chosen to force the selection of: first, cells $i \in \mathcal{P} \cup \mathcal{S}$ and $a_i \geq *_p$ ($* = lpl$ or $* = upl$, following the notation of Figure 8); second, cells $i \notin \mathcal{P} \cup \mathcal{S}$ and $a_i \geq *_p$; third, cells $i \in \mathcal{P} \cup \mathcal{S}$ and $a_i < *_p$; and, finally, cells $i \notin \mathcal{P} \cup \mathcal{S}$ and $a_i < *_p$. This cost stratification attempts to balance the number of new secondary suppressions and shortest-path subproblems to be solved. Clearly, for each of the above four categories, cells with the lowest w_i values are preferred. The particular costs set by the heuristic at line 7 of Figure 8 for all the cells a_i—but those not allowed in the shortest path—when dealing with the primary cell p and the lower protection level are

$$\text{cost } a_i = \begin{cases} 1 & i \in \mathcal{P} \cup \mathcal{S} \text{ and } a_i \geq lpl_p \\ C + w_i & i \notin \mathcal{P} \cup \mathcal{S} \text{ and } a_i \geq lpl_p \\ C(2n - C + 1) + M & \\ & i \in \mathcal{P} \cup \mathcal{S} \text{ and } a_i < lpl_p \\ (C(2n - C + 1) + M)(C + 1) + w_i & \\ & i \notin \mathcal{P} \cup \mathcal{S} \text{ and } a_i < lpl_p, \end{cases} \tag{11}$$

where $C = |\mathcal{P}| + |\mathcal{S}|$ and $M \geq \sum_{i=1}^n w_i$. The costs for the upper protection level are obtained by replacing lpl_p by upl_p in (11). Note that (11) can be computed in a single loop over the n cells, whereas other procedures (Cox 1995) required two loops. In practice this is important, since (11) is the most computationally expensive step of the heuristic.

4.8. Shortest-Path Solver

Shortest-path subproblems are solved through an efficient d-heap implementation of Dijkstra's algorithm (Ahuja et al. 1993, Chap. 4). Since we are interested in the shortest path to a single destination, a bidirectional version is used. In practice, this can be considered the most efficient algorithm for these kind of problems. As shown in the computational results of Section 6, with this solver, the heuristic is from one to three orders of magnitude faster than other approaches based on minimum-cost network codes.

4.9. Complexity of the Shortest-Paths Heuristic

We will assume the following:

(i) The sequence of shortest paths can always protect the primary cell, i.e., the solution of minimum-cost network-flows subproblems is never required to certify the feasibility of the instance. This assumption is satisfied in all the computational experiments of Section 6.

(ii) For all primary cells and protection levels, the number of shortest paths in the sequence is bounded by an integer K, independent of the size of the problem. This assumption was empirically observed in the computational experiments of Section 6. As shown in Figure 9, the average number of shortest paths required for the protection of each primary does not increase with the number of cells of the instance.

From the above assumptions, the running time of the heuristic is obtained as follows. For each primary cell we have to compute K shortest paths for the lower protection level, and K for the upper protection level. The costs of the n arcs must be updated for each shortest-path computation. The running time of Dijkstra's algorithm for shortest paths depends on the variant considered (Ahuja et al. 1993, Chap. 4). For instance, it is $O(m^2)$ for the original Dijkstra implementation, and $O(n\log_{n/m} m)$ for the d-heap ($d = n/m$) implementation used in this work, m being the number of nodes of the network. The running time of the heuristic using the d-heap implementation is thus

$$\begin{aligned} O(|\mathcal{P}|2K(n+n\log_{n/m} m)) &= O(|\mathcal{P}|2Kn(1+\log_{n/m} m)) \\ &= O(|\mathcal{P}|n\log_{n/m} m) \qquad (12) \end{aligned}$$

(note that $\log_{n/m} m \geq 1$, and only for dense networks is equal to 1).

Using (8) we can express the running time in terms of the number of subtables, rows, and columns of the hierarchical table. To simplify the final expression we define $s = \max\{c, r_k\ k = 1, \ldots, t\}$, i.e., s is the maximum number of rows or columns of any subtable. From (8) the number of rows and columns satisfy $n = O(ts^2)$ and $m = O(ts)$. Using these expressions in (12) we obtain the following running time:

$$O(|\mathcal{P}|ts^2\log_s t). \qquad (13)$$

If assumption (ii) above is not considered, we must bound K. The protection provided by any shortest path of the sequence, computed as in (9), satisfies $\gamma \geq \min\{a_i,\ i = 1, \ldots, n,\ a_i \neq 0\} \geq 1$. Moreover, in practice the lower and upper protection levels are a fraction β of the cell value. Therefore,

$$K \leq \frac{\max_{p\in\mathcal{P}}\{lpl_p, upl_p\}}{\gamma} \leq \beta U, \quad \text{where } U = \max_{i=1,\ldots,n}\{a_i\}. \qquad (14)$$

An alternative bound on K can be obtained considering that at least one cell is made secondary after each shortest path, i.e., at least one arc is not allowed to appear in successive shortest paths of the sequence. Thus, K cannot be greater than the number of arcs of the network:

$$K \leq n. \qquad (15)$$

From (14) and (15) we obtain the following running time:

$$O(|\mathcal{P}|\min\{\beta U, ts^2\}ts^2\log_s t). \qquad (16)$$

Note that (15) provides a strongly polynomial-time algorithm even if assumption (ii) does not hold.

Previous heuristics for CSP required the solution of at least one minimum-cost network-flows subproblem for each primary cell. The currently fastest strongly polynomial-time algorithm for the minimum-cost flow problem—the enhanced capacity-scaling algorithm (Ahuja et al. 1993, Chap. 10)—runs in $O((n\log m)(n + m\log m))$. Again using (8), the running time of a minimum-cost network-flows heuristic for CSP is

$$\begin{aligned} &O(|\mathcal{P}|(n\log m)(n+m\log m)) \\ &\quad = O(|\mathcal{P}|(t^2s^4\log ts + t^2s^3(\log ts)^2)). \qquad (17) \end{aligned}$$

The running time reported in (13) is clearly better than (17). Even if assumption (ii) is not considered, (16) provides a better running time.

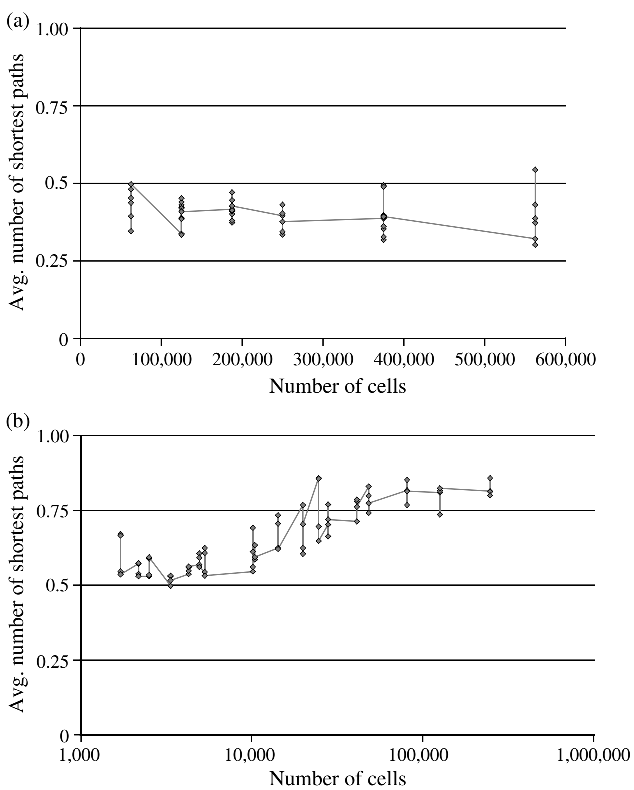

Figure 9 Average Number of Shortest Paths Required for Each Primary Cell vs. Number of Cells for (a) Two-Dimensional Instances, and (b) Hierarchical Instances

5. Computing a Lower Bound

The relative gap between the computed lower bound and the solution provided by the heuristic can be used as an approximate measure of how far the solution is from the optimum. Two improvements to the procedure introduced in Kelly et al. (1992)—and used in Fischetti and Salazar (2001)—have been developed: we extend the procedure for two-dimensional tables with one hierarchical variable, formulated as in (7), and we add extra constraints that provide a higher lower bound. Lower-bounding procedures have also been used in the context of controlled tabular adjustment in Cox et al. (2005).

Given a two-dimensional table with one hierarchical variable a_{ij}^k, $k=1,\dots,t$, $i=1,\dots,r_k+1$, $j=1,\dots,c+1$ with the hierarchical relations of (6), and defining the auxiliary parameters

$$s_{ij}^k = 1 \text{ if cell } a_{ij}^k \text{ is primary, otherwise } s_{ij}^k = 0,$$

$$\alpha_i^k = \max\{a_{ij}^k + upl_{ij}^k \;\; j=1,\dots,c+1\colon a_{ij}^k \text{ is primary}\}, \quad (18)$$

$$\beta_j^k = \max\{a_{ij}^k + upl_{ij}^k \;\; i=1,\dots,r_k+1\colon a_{ij}^k \text{ is primary}\},$$

(upl_{ij}^k being the upper protection limit of primary cell a_{ij}^k), the lower bound is the optimal objective function of the linear relaxation of the following integer problem:

$$\min_{y,u,v} \sum_{k=1}^{t} \sum_{i=1}^{r_k+1} \sum_{j=1}^{c+1} w_{ij}^k y_{ij}^k \quad (19)$$

$$\sum_{j=1}^{c+1} y_{ij}^k \ge 2 \quad \text{for all } k,\, i\colon \sum_{j=1}^{c+1} s_{ij}^k = 1 \quad (20)$$

$$\sum_{j=1}^{c+1} y_{ij}^k \ge 2u_i^k \quad \text{for all } k,\, i\colon \sum_{j=1}^{c+1} s_{ij}^k = 0 \quad (21)$$

$$u_i^k \ge y_{ij}^k \quad \text{for all } k,\, i,\, j\colon \sum_{l=1}^{c+1} s_{il}^k = 0 \quad (22)$$

$$\sum_{j=1}^{c+1} a_{ij}^k y_{ij}^k \ge \alpha_i^k \quad \text{for all } k, i\colon \text{for some } j \; a_{ij}^k \text{ is primary} \quad (23)$$

$$\sum_{i=1}^{r_k+1} y_{ij}^k \ge 2 \quad \text{for all } k,\, j\colon \sum_{i=1}^{r_k+1} s_{ij}^k = 1 \quad (24)$$

$$\sum_{i=1}^{r_k+1} y_{ij}^k \ge 2v_j^k \quad \text{for all } k,\, j\colon \sum_{i=1}^{r_k+1} s_{ij}^k = 0 \quad (25)$$

$$v_j^k \ge y_{ij}^k \quad \text{for all } k, i, j\colon \sum_{l=1}^{r_k+1} s_{lj}^k = 0 \quad (26)$$

$$\sum_{i=1}^{r_k+1} a_{ij}^k y_{ij}^k \ge \beta_j^k \quad \text{for all } k,\, j\colon \text{for some } i \; a_{ij}^k \text{ is primary} \quad (27)$$

$$y_{o_k,j}^{u_k} = y_{r_k+1,j}^{v_k} \quad k=1,\dots,t-1 \;\; j=1,\dots,c+1 \quad (28)$$

$$y_{ij}^k \ge s_{ij}^k \quad \text{for all } k,\, i,\, j \quad (29)$$

$$y_{ij}^k \in \{0,1\} \quad \text{for all } k,\, i,\, j \quad (30)$$

$$0 \le u_i^k \le 1 \quad \text{for all } k,\, i \quad (31)$$

$$0 \le v_j^k \le 1 \quad \text{for all } k,\, j. \quad (32)$$

Variable y_{ij}^k is set to 1 if the cell a_{ij}^k is removed, and otherwise is 0. Constraints (20)–(23) refer to rows, while (24)–(27) are for columns. The linking constraints among subtables in (28) force that the same cell in two subtables must have the same status, either published or suppressed. Finally, (29)–(32) are the integrality constraints and bounds. Constraints (20)–(22), (24)–(26), and (29)–(32) were originally introduced in Kelly et al. (1992), but for the subtable superscript k. They force at least two suppressions in rows and columns with a single primary (constraints (20) and (24)), or with some secondary (constraints (21)–(22) and (25)–(26)). Any optimal solution must clearly satisfy the above constraints.

The new constraints (23) and (27) must also be satisfied by any optimal solution, if the upper protection levels are a fraction of the cell values (which is common practice), as shown by the next proposition.

Proposition 1. *For all primary cells a_{uv}^k, if $upl_{uv}^k \le a_{uv}^k$—i.e., the upper protection level is less than or equal to the cell value—any solution of CSP satisfies*

$$\textit{row constraint}\colon \sum_{j=1}^{c+1} a_{uj}^k y_{uj}^k \ge a_{uv}^k + upl_{uv}^k, \quad (33)$$

$$\textit{column constraint}\colon \sum_{i=1}^{r_k+1} a_{iv}^k y_{iv}^k \ge a_{uv}^k + upl_{uv}^k, \quad (34)$$

y_{ij}^k being 1 if cell a_{ij}^k is suppressed, otherwise is 0.

Proof. We only prove the result for the row constraint. The same procedure can be used for the column constraint. Since a_{uv}^k is primary we know that $y_{uv}^k = 1$. We consider two cases. First, if the marginal row cell $a_{u,c+1}^k$ is suppressed (i.e., $y_{u,c+1}^k = 1$), we have

$$\sum_{j=1}^{c+1} a_{uj}^k y_{uj}^k \ge a_{uv}^k + a_{u,c+1}^k \ge a_{uv}^k + upl_{uv}^k,$$

since the marginal row cell is always greater than or equal to any internal cell, and we assumed that $upl_{uv}^k \le a_{uv}^k$. Second, consider the case where the marginal cell is not suppressed (i.e., $y_{u,c+1}^k = 0$). Therefore, $\sum_{j=1}^{c+1} a_{uj}^k y_{uj}^k = \sum_{j=1}^{c} a_{uj}^k y_{uj}^k$. Assume that (33) is not satisfied. When solving (2) for the primary cell a_{uv}^k, one of the constraints is $\sum_{\forall j:\, y_{uj}^k=1} x_{uj}^k = \sum_{j=1}^{c} a_{uj}^k y_{uj}^k$, and then the maximum value for this primary satisfies $\bar{a}_{uv}^k \le \sum_{j=1}^{c} a_{uj}^k y_{uj}^k < a_{uv}^k + upl_{uv}^k$, which does not

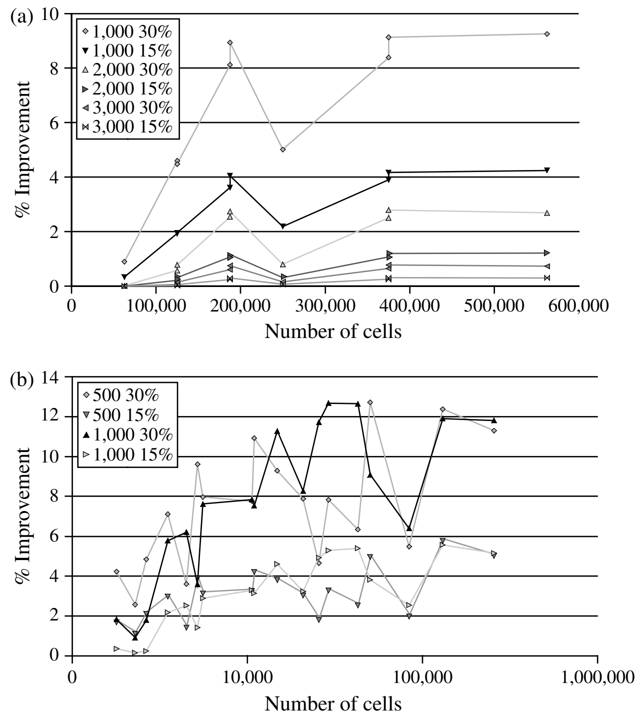

Figure 10 Improvement with the New Lower Bounding Procedure vs. Number of Cells for (a) Two-Dimensional Instances, and (b) Hierarchical Instances

guarantee the protection condition (1). We then conclude that (33) must hold. □

As an immediate result of the above proposition, all the constraints (33) ((34)) of cells of the same row (column) can be replaced by the one with the maximum right-hand side, obtaining (23) ((27)).

In practice, the effectiveness of constraints (23) and (27) (i.e., the quality of the lower bound compared to the original formulation without those constraints) increases with the size of the upper protection levels. This is clearly shown in Figure 10, which plots the percentage improvement in the quality of the lower bound due to constraints (23) and (27) versus the number of cells of the table, for two-dimensional and hierarchical tables. The percentage of improvement is computed as $100(nlb - olb)/olb$, olb being the lower bound computed with the original procedure of Kelly et al. (1992), and nlb the lower bound obtained with the new formulation that includes constraints (23) and (27). Each line plotted in Figure 10 corresponds to a group of instances with the same number of primary cells and upper protection limits expressed as a percentage of the cell value (we considered 15% and 30%, which are usual values). The particular pairs considered for these values, one per line, are shown in the legends in the right margin of the figures. The instances considered are a subset of those used in the computational results of Section 6. From Figure 10 it is clear that for two-dimensional tables, the improvement due to the new lower bound increases with the upper protection limit, and decreases with the number of primary cells of the table. On the other hand, for hierarchical tables, the improvement is only explained by a larger upper protection limit, independent of the number of primary cells. From the positive slopes of the lines in both figures, it can also be stated that the new lower bound improves with the size of the table.

6. Computational Results

The heuristic of Section 4—including the lower-bounding procedure of Section 5—has been implemented in C. It is currently included in the τ-Argus package (Hundepool 2004) for tabular data protection, which is used by several European NSAs. For testing purposes, we considered two classes of problems. The first class consisted of randomly generated instances, while real-world problems—thus confidential—were used for the second one. Random instances were produced with the generator for two-dimensional tables used in Castro (2002), and with an extension for hierarchical tables. Both generators can be obtained from http://www-eio.upc.es/~jcastro/generators_csp.html. We produced 54 two-dimensional instances, ranging from 62,500 to 562,500 cells, and with $|\mathcal{P}| \in \{1{,}000, 2{,}000, 3{,}000\}$, $|\mathcal{P}|$ being the number of primary cells. Cell weights were set to $w_i = a_i$ (i.e., cell value). We also generated 72 two-dimensional hierarchical tables, ranging from 1,716 to 246,942 cells and from 4 to 185 subtables, with $|\mathcal{P}| \in \{500, 1{,}000\}$. Cells weights were set to $w_i = a_i$ for half of the instances and $w_i = 1$ for the remaining ones. In all the cases the lower and upper protection levels were set at 15% of the cell value. Executions with random instances were carried out on a standard PC with a 1.8 GHz Pentium-4 processor and 1 GB of RAM.

The results obtained with random instances are summarized in Figure 11, which shows for the two-dimensional and hierarchical tables, the CPU time in seconds versus the number of cells of the table, for the different number of primary cells. Clearly, the CPU time increases with both $|\mathcal{P}|$ and the number of cells. However, the shortest-paths heuristic was able to provide a solution in few seconds.

Figure 12 shows, again for the random two-dimensional and hierarchical tables, the efficiency of the shortest-paths heuristic compared to alternative approaches based on network flows. We applied the algorithm of Figure 8 twice, formulating minimum-cost network-flows subproblems (as previous approaches did), and shortest-path ones. The minimum-cost network-flows subproblems were solved with the network simplex solver of CPLEX 7.5 a state-of-the-art implementation (ILOG 2001).

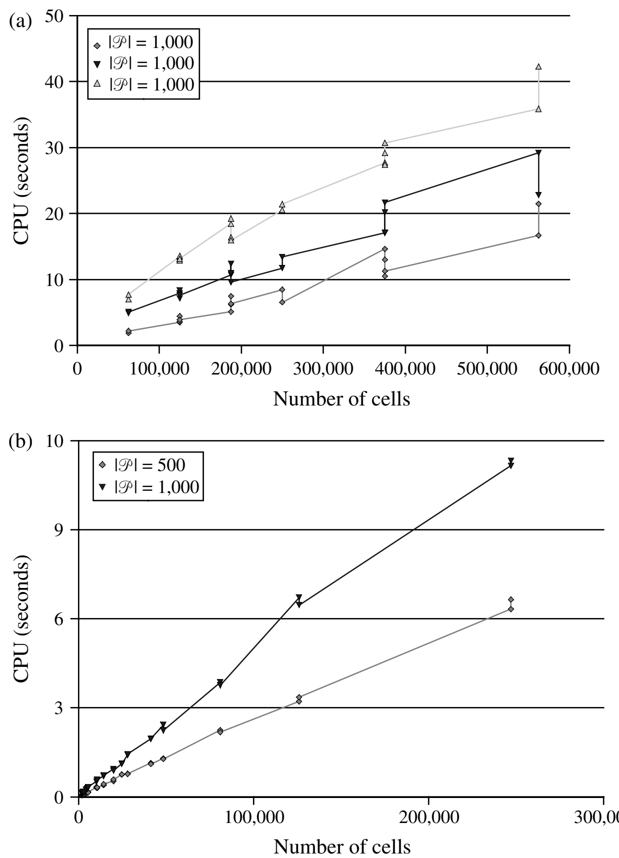

Figure 11 CPU Time vs. Number of Cells for (a) Two-Dimensional Instances, and (b) Hierarchical Instances

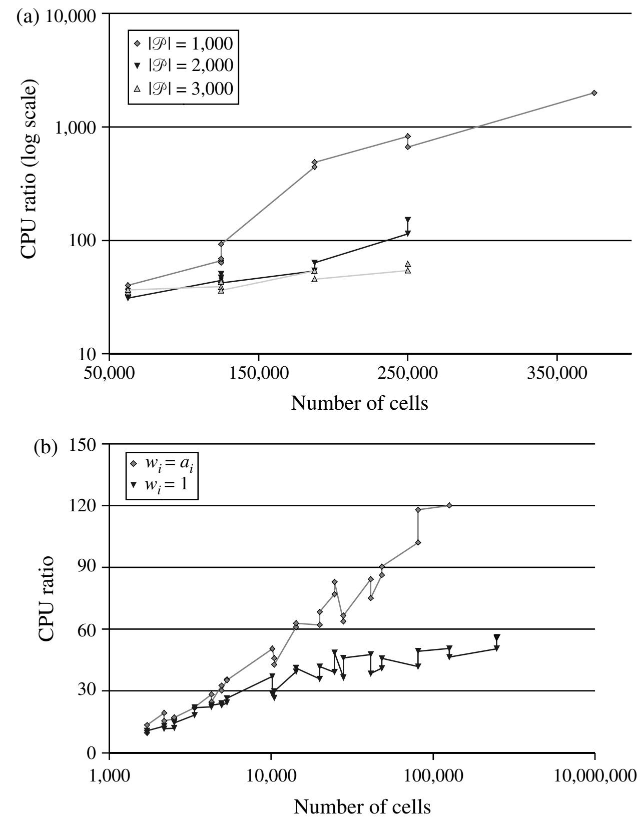

Figure 12 CPU Ratio Between Minimum-Cost Network Flows and Shortest-Paths Heuristics vs. Number of Cells for (a) Two-Dimensional Instances, and (b) Hierarchical Instances

The larger instances were not solved because CPLEX required excessive execution time. The vertical axes of the plots of Figure 12 show the ratio between the CPU time of CPLEX 7.5 and the implementation of Dijkstra's algorithm used in the heuristic. For the two-dimensional tables the instances for the different $|\mathcal{P}|$ values are plotted separately. Similarly, two lines are plotted for hierarchical tables, one for each type of weights ($w_i = a_i$ and $w_i = 1$). The time ratio increases with the table dimension, and it is about 1,900 and 120 for the largest two-dimensional and hierarchical instances, respectively. The shortest path formulation is thus instrumental in performance of the heuristic.

Finally, Figure 13 shows, for the random two-dimensional and hierarchical tables, an estimation of the quality of the solution obtained. The vertical axes show the relative gap $(ws - lb)/ws$, ws being the weight suppressed by the heuristic, and lb the computed lower bound. Those figures must be interpreted with caution, since the computed lower bound can be far away from the optimum, which is unknown for these large instances. At first sight, it could be concluded that the heuristic works much better for two-dimensional than for hierarchical tables. However, the lower-bounding procedure could be providing better bounds for two-dimensional tables. It is thus difficult to know which factor—the quality of the heuristic or the quality of the lower-bounding procedure—explains the much larger gap for hierarchical tables. Both factors likely intervene, and in that case we should conclude that the heuristic behaves better for two-dimensional than for hierarchical tables. Note that for two-dimensional instances with the largest number of primary cells we obtain solutions with an optimality gap of less than 1%.

The second class of problems was made of a small number of real-world hierarchical instances. Cells weights were set to $w_i = a_i$ in all cases. Table 1 shows the dimensions and results obtained with the shortest-paths heuristic. Table 2 gives the results obtained with three alternative procedures. The information reported for instances CBS* and DES* was provided, respectively, by Centraal Bureau voor de Statistiek (Dutch NSA), and Destatis (German NSA). They were obtained with the τ-Argus package, running its four available solvers for tabular data protection: two heuristics, named HiTas (de Wolf 2002) and hypercube (Giessing and Repsilber 2002), the optimal procedure described in Fischetti and Salazar (2001), and the shortest-paths heuristic of this work. For each instance, we provide the total number of cells in the

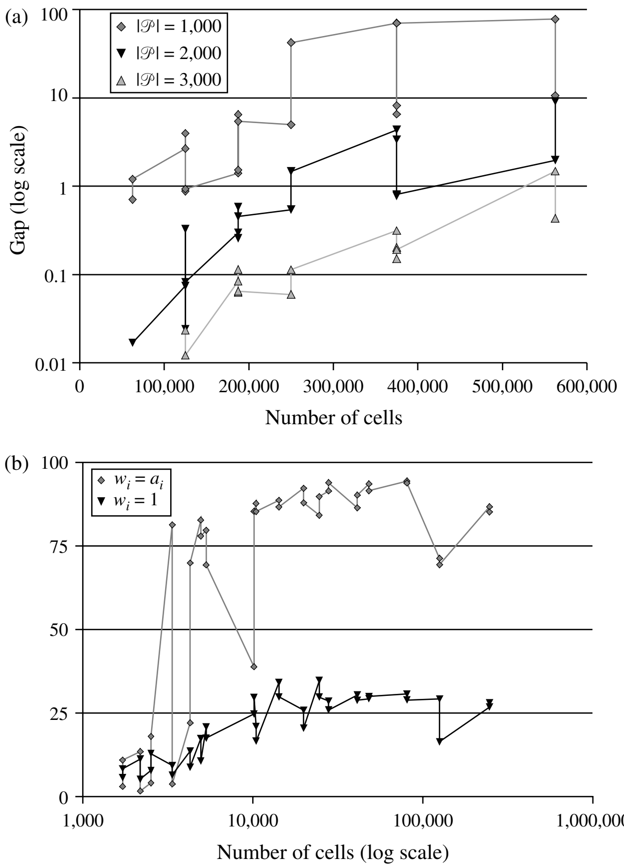

Figure 13 Gap vs. Number of Cells for (a) Two-Dimensional Instances, and (b) Hierarchical Instances

hierarchical table n, the number of primary cells $|\mathcal{P}|$, and the CPU execution time (columns "CPU") and weight of suppressed secondary cells (columns "WS") for each of the above four methods. Problems CBS* and DES* were solved, respectively, on a 1.5 GHz Pentium-4 and a 900 MHz Pentium-3 processor.

It is noteworthy that the HiTas and hypercube heuristics include a control to avoid the protection of single respondent cells (i.e., cells with only one individual) through other single respondent cells. That slightly penalizes the weight suppressed by these two heuristics. On the other hand, these two heuristics can provide—and in practice they do—nonfeasible solutions, i.e., patterns of suppressions that do not guarantee the protection levels. Therefore, comparisons with these two nonfeasible heuristics must be done with caution. The optimal procedure and the shortest-paths heuristic do not include the above control for single respondent cells, but always guarantee feasible solutions.

Table 1 Dimensions and Results with the Shortest-Paths Heuristic for the Real-World Instances

			Shortest paths	
Name	n	$\vert\mathcal{P}\vert$	WS	CPU
CBS1	6,399	570	4.84e+6	4
CBS2	172,965	68,964	2.96e+10	403
DES1	460	18	0.87e+6	6
DES2	1,050	61	2.44e+7	4
DES3	8,230	994	12.9e+7	10
DES4	16,530	2,083	1.83e+8	21
DES4a	29,754	3,494	11.9e+7	65

Table 2 Results for the Real-World Instances Using Alternative Procedures

	HiTas		Hypercube		Optimal	
Name	WS	CPU	WS	CPU	WS	CPU
CBS1	5.85e+6	12	11.8e+6	6	4.85e+6	>3,600
CBS2	1.31e+10	1,151	24.9e+10	177	—	—
DES1	1.68e+6	1	43.2e+6	2	0.90e+6	93
DES2	2.57e+7	4	4.06e+7	4	2.41e+7	98
DES3	9.41e+7	35	42.2e+7	9	10.2e+7	618
DES4	1.54e+8	38	5.98e+8	14	Fail	Fail
DES4a	5.95e+7	119	33.8e+7	24	Fail	Fail

From Tables 1 and 2 we conclude that the shortest-paths heuristic is far more efficient than the optimal procedure and provides good solutions. Indeed, for instances CBS1 and DES1 the shortest-paths heuristic provided a better solution than the optimal procedure with its default optimality tolerance; and it was 900 and 15.5 times faster, respectively, for each of these two instances. Note that CBS2, the largest instance, was not attempted to be solved with the optimal procedure, because of the excessive computational resources required. Moreover, the optimal procedure failed for DES4 and DES4a. As for the other two approaches, the hypercube is more efficient than the shortest-paths heuristic, but provides much worse solutions. On the other hand, HiTas is slower than the shortest-paths heuristic, but provides slightly better solutions. However, these two heuristics do not guarantee the protection of all primary cells. To detect the unprotected cells we need to solve (2) for each primary cell. Then, these unprotected cells must be dealt with using some feasible method, as the optimal one or the shortest-paths heuristic. However, this procedure would significantly increase the overall execution time. The shortest-paths heuristic guarantees feasible good solutions and reasonable execution times.

7. Conclusions

Our shortest-paths heuristic for positive tables is an efficient procedure for the protection of large two-dimensional and hierarchical tables. From the computational experience of end users with real data, it is a competitive approach compared to alternative procedures. This heuristic has been included in the τ-Argus package and it does not require any external solver; thus it can be freely used by any NSA.

There are some possible future extensions. One of them is to avoid protection of single respondent cells (i.e., cells with only one individual) through other single respondent cells. Another extension is to add a post-process for the detection of unnecessary oversuppressed cells; however, unlike the approach of Kelly et al. (1992), and for efficiency reasons, such a procedure would only solve shortest-path subproblems. Yet another extension would be to deal with more complicated tables (e.g., tables with two hierarchical variables, in rows and columns). Such classes of tables cannot be modeled as networks, but as networks with many equal flow constraints (Ahuja et al. 1999, Calvete 2003). Instead of shortest-path subproblems, the heuristic should then solve "equal-flow shortest-path" subproblems (i.e., shortest-path problems with constraints that force that, if a certain arc is in the path, its pair must also be in the path). An efficient procedure for the "equal-flow shortest-path" problem is still an open issue.

Acknowledgments

This work was supported by the European Union IST-2000-25069 CASC project and the Spanish MCyT Project TIC2003-00997. The author thanks Narcís Nabona (Department of Statistics and Operations Research, Universitat Politècnica de Catalunya) for providing the implementation of Dijkstra's shortest-path algorithm. The author also thanks Sarah Giessing and Anco Hundepool, respectively, from Destatis (German NSA), and Centraal Bureau voor de Statistiek (Dutch NSA), for the results of the confidential real-world examples. Finally, the author thanks two anonymous referees for helpful comments and suggestions.

References

Ahuja, R. K., T. L. Magnanti, J. B. Orlin. 1993. *Network Flows. Theory, Algorithms and Applications*. Prentice Hall, Upper Saddle River, NJ.

Ahuja, R. K., J. B. Orlin, P. Zuddas, G. Secki. 1999. Algorithms for the equal flow problem. *Management Sci.* **45** 1440–1455.

Bixby, R. E. 2002. Solving real-world linear programs: A decade and more of progress. *Oper. Res.* **50** 3–15.

Calvete, H. 2003. Network simplex algorithm for the general equal flow problem. *Eur. J. Oper. Res.* **150** 585–600.

Carvalho, F. D., N. P. Dellaert, M. D. Osório. 1994. Statistical disclosure in two-dimensional tables: General tables. *J. Amer. Statist. Assoc.* **89** 1547–1557.

Castro, J. 2002. Network flows heuristics for complementary cell suppression: An empirical evaluation and extensions. J. Domingo-Ferrer, ed. *Inference Control in Statistical Databases, Lecture Notes in Computer Science*, Vol. 2316. Springer, Berlin, Germany, 59–73.

Castro, J. 2004. A fast network flows heuristic for cell suppression in positive tables. J. Domingo-Ferrer, V. Torra, eds. *Privacy in Statistical Databases, Lecture Notes in Computer Science*, Vol. 3050. Springer, Berlin, Germany, 136–148.

Castro, J., N. Nabona. 1996. An implementation of linear and nonlinear multicommodity network flows. *Eur. J. Oper. Res.* **92** 37–53.

Cox, L. H. 1995. Network models for complementary cell suppression. *J. Amer. Statist. Assoc.* **90** 1453–1462.

Cox, L. H., J. A. George. 1989. Controlled rounding for tables with subtotals. *Ann. Oper. Res.* **20** 141–157.

Cox, L. H., J. P. Kelly, R. Patil. 2005. Computational aspects of controlled tabular adjustment: Algorithm and Analysis. B. Golden, S. Raghavan, E. Wassil, eds. *The Next Wave in Computer, Optimization and Decision Technologies*. Kluwer, Boston, MA, 45–59.

Dandekar, R. A. 2003. Energy Information Administration, Department of Energy, Washington, D.C. Personal communication.

Dellaert, N. P., W. A. Luijten. 1999. Statistical disclosure in general three-dimensional tables. *Statistica Neerlandica* **53** 197–221.

de Wolf, P. P. 2002. HiTaS: A heuristic approach to cell suppression in hierarchical tables. J. Domingo-Ferrer, ed. *Inference Control in Statistical Databases, Lecture Notes in Computer Science*, Vol. 2316. Springer, Berlin, Germany, 74–82.

Domingo-Ferrer, J., ed. 2002. *Inference Control in Statistical Databases. Lecture Notes in Computer Science*, Vol. 2316. Springer, Berlin, Germany.

Domingo-Ferrer, J., V. Torra. 2002. A critique of the sensitivity rules usually employed for statistical table protection. *Internat. J. Uncertainty, Fuzziness and Knowledge-Based Systems* **10** 545–556.

Domingo-Ferrer, J., V. Torra, eds. 2004. *Privacy in Statistical Databases. Lecture Notes in Computer Science*, Vol. 3050. Springer, Berlin, Germany.

Fischetti, M., J. J. Salazar. 2001. Solving the cell suppression problem on tabular data with linear constraints. *Management Sci.* **47** 1008–1026.

Giessing, S., D. Repsilber. 2002. Tools and strategies to protect multiple tables with the GHQUAR cell suppression engine. J. Domingo-Ferrer, ed. *Inference Control in Statistical Databases, Lecture Notes in Computer Science*, Vol. 2316. Springer, Berlin, Germany, 181–192.

Gusfield, D. 1988. A graph theoretic approach to statistical data security. *SIAM J. Comput.* **17** 552–571.

Hundepool, A. 2004. The ARGUS software in the CASC project. J. Domingo-Ferrer, V. Torra, eds. *Privacy in Statistical Databases, Lecture Notes in Computer Science*, Vol. 3050. Springer, Berlin, Germany, 323–335.

ILOG. 2001. *ILOG CPLEX 7.5 Reference Manual Library*. ILOG, Gentilly, France.

Jewett, R. 1993. Disclosure analysis for the 1992 economic census. Manuscript, Economic Programming Division, Bureau of the Census, Washington, D.C.

Kelly, J. P., B. L. Golden, A. A. Assad. 1992. Cell suppression: Disclosure protection for sensitive tabular data. *Networks* **22** 28–55.

Willenborg, L., T. de Waal, eds. 2000. *Elements of Statistical Disclosure Control. Lecture Notes in Statistics*, Vol. 155. Springer, New York.