A Fast Network Flows Heuristic for Cell Suppression in Positive Tables

Jordi Castro*

Department of Statistics and Operations Research
Universitat Politècnica de Catalunya
Pau Gargallo 5, 08028 Barcelona, Catalonia, Spain
jcastro@eio.upc.es
http://www-eio.upc.es/~jcastro

Abstract. Current network flows heuristics for cell suppression in positive tables (i.e, cell values are greater or equal than zero) rely on the solution of several minimum-cost subproblems. A very efficient heuristic based on shortest-paths was also developed in the past, but it was only appropriate for general tables (i.e., cell values can be either positive or negative), whereas in practice most real tables are positive. The method presented in this work overcomes the lacks of previous approaches: it is designed for positive tables and only requires the solution of shortest-paths subproblems. It follows that it is orders of magnitude faster than previous network flows heuristics. We report an extensive computational experience in the solution of two-dimensional tables, either with or without hierarchical rows, of up to 250000 and 500000 cells, respectively.

Keywords: statistical disclosure control, cell suppression problem, linear programming, network optimization, shortest-paths.

1 Introduction

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

Because of the NP-hardness of CSP [11], most of the former approaches focused on heuristics for approximate solutions. This work deals with them, presenting a new one for positive tables (i.e., cell values are greater or equal than zero) based on shortest-paths, which turned out to be orders of magnitude faster than alternative methods. A recent mixed integer linear programming (MILP) method [5] was able to solve until optimality nontrivial CSP instances.

* Supported by the EU IST-2000-25069 CASC project and the Spanish MCyT Project TIC2003-00997. The author thanks Narcís Nabona for providing him with the implementation of the Dijkstra's shortest-paths algorithm.

J. Domingo-Ferrer and V. Torra (Eds.): PSD 2004, LNCS 3050, pp. 136–148, 2004.

The main inconvenience of this approach, from the practitioner point of view, is that the solution of very large instances (with possibly millions of cells) can result in impractical execution times [6]. It is noteworthy that improvements in new heuristics also benefit the above exact procedure, since they provide it with a fast, hopefully good, feasible starting point.

Most current heuristic methods for CSP are based on the solution of network flows subproblems [1]. Other approaches, as the hypercube method [7], focus on geometric considerations of the problem. Although very efficient, the hypercube method provides patterns with a large number of secondary cells or value suppressed (i.e., it suffers from over-suppression). Network flows heuristics for CSP usually exploit more efficiently the table information and provide better results.

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 [3, 4, 11]. Such approaches have been successfully applied in practice [10, 13]. These heuristics require the table structure to be modeled as a network, which can only be accomplished for two-dimensional tables with, at most, one hierarchical dimension (either rows or columns). Although minimum-cost network flows algorithms are fast compared to the equivalent linear programming formulations [1, Ch. 9–11], for large tables they still require large execution times. Instead, the approach suggested in [2] formulated shortest-paths subproblems, a particular case of minimum-cost network flows that can be solved much more efficiently through specialized algorithms [1, Ch. 4–5]. The main drawback of that approach based on shortest-paths was that it could only be applied to general tables (i.e., cell values can be either positive or negative), which are the less common in practice.

To avoid the above lacks of current network flows heuristics (namely, the efficiency of those based on minimum-cost flows subproblems, and the suitability of that based on shortest-paths for positive tables) we present a new method that sensibly combines and improves ideas of previous approaches (mainly [2] and [4]). The resulting method applies to positive tables and formulates shortest-paths subproblems. As shown by the computational results, it is much faster than previous network-flows heuristics for positive tables. The new approach has been included in the τ-Argus package [8] in the scope of the European Union funded "CASC" project IST-2000-25069, for the protection of two-dimensional tables with at most one hierarchical dimension.

This paper is organized as follows. 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 shows the heuristic introduced in this work. Finally, Section 5 reports the computational experience in the solution of large instances, showing the efficiency 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 some linear relations $Aa = b$), a set $\mathcal{P}$ of p primary sensitive cells to be protected, and upper

and lower protection levels upl_i and lpl_i for each primary cell $i = 1 \dots p$, the purpose of CSP is to find a set $\mathcal{S}$ of additional secondary cells whose suppression guarantees that, for each $p \in \mathcal{P}$,

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

$\underline{a_p}$ and $\overline{a_p}$ being defined as

$$\underline{a_p} = \begin{array}{ll} \min\limits_{x_i, i=1\dots n} & x_p \\ \text{s.t.} & Ax = b \\ & x_i \geq 0 \;\; i \in \mathcal{P} \cup \mathcal{S} \\ & x_i = a_i \;\; i \notin \mathcal{P} \cup \mathcal{S} \end{array} \quad \text{and} \quad \overline{a_p} = \begin{array}{ll} \max\limits_{x_i, i=1\dots n} & x_p \\ \text{s.t.} & Ax = b \\ & x_i \geq 0 \;\; i \in \mathcal{P} \cup \mathcal{S} \\ & x_i = a_i \;\; i \notin \mathcal{P} \cup \mathcal{S}. \end{array} \tag{2}$$

$\underline{a_p}$ and $\overline{a_p}$ 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. Imposing (1), the desired level of protection is guaranteed. CSP can thus be formulated as an optimization problem of minimizing some function that measures the cost of suppressing additional cells subject to that conditions (1) and (2) are satisfied for each primary cell.

CSP was first formulated in [11] as a large MILP problem. For each entry a_i a binary variable $y_i, i = 1 \dots n$ 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 requirements of (1) and (2). These vectors represent cell deviations (positive or negative) from the original a_i values. The resulting model is

$$\begin{array}{ll} \min & \sum\limits_{i=1}^{n} w_i y_i \\ \text{s.t.} & \\ & \left.\begin{array}{rll} & Ax^{l,p} = 0 & \\ -a_i y_i \leq & x_i^{l,p} \leq My_i & i = 1 \dots n \\ & x_p^{l,p} \leq -lpl_p & \\ & & \\ & Ax^{u,p} = 0 & \\ -a_i y_i \leq & x_i^{u,p} \leq My_i & i = 1 \dots n \\ & x_p^{u,p} \geq upl_p & \end{array}\right\} \text{for each } p \in \mathcal{P} \\ & \\ & y_i \in \{0, 1\}, \end{array} \tag{3}$$

w_i being the information loss associated to cell a_i. In practice they are usually set to $w_i = a_i$ (minimize the overall suppressed value) or $w_i = 1$ (minimize the overall number of suppressed cells). Inequality constraints of (3) with both right and left-hand sides impose the bounds of $x_i^{l,p}$ 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 very large MILP problem even for tables of

moderate sizes and number of primary cells. If matrix A can be modeled as a network, we can find a feasible non-optimal (but hopefully good) solution to (3) through a network-flows heuristic.

3 Modelling Tables as Networks

It is well-known that he linear relations of a two-dimensional $(r+1) \times (c+1)$ table defined by system $Ax = b$ can be modeled as the network of Figure 1. Arcs are associated to cells and nodes to constraints; row $r+1$ and column $c+1$ correspond to marginals.

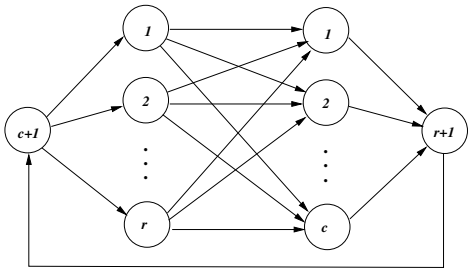

Fig. 1. Network representation of a $(r+1) \times (c+1)$ table

Two-dimensional tables with one hierarchical dimension can also be modeled through a network [10]. Without loss of generality we will assume that hierarchies appear in rows. Before presenting the overall algorithm for computing the network, we first illustrate it through the example of Figure 2. Row R_2 of table 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 table T_2 is also hierarchical; T_3 shows its structure. Although in the example all the tables have the same number of rows, this is not required in general. However, the number of columns must be the same for all the tables; otherwise, we would not preserve the hierarchical structure in only one dimension. Clearly every subtable can be modeled through a network similar to that of Figure 1.

The hierarchical structure tree of the example is shown in Figure 3. That tree has three levels, and one table per level. In general, we can have hierarchical tables of any number of levels, and any number of tables per level (i.e., any number of hierarchical rows for each table).

The procedure first computes a list of all the tables in the hierarchical structure tree, using a breadth-first order. If we denote it by BFL, we have for the example $BFL = \{T_1, T_2, T_3\}$. Next, the first table of the list is extracted (T_1), which is always the top table of the tree (i.e., that with the highest level of aggregation). An initial network is created for T_1. This is the hierarchical network, which will be successively updated. It is depicted in table a) of Figure 4. For the rest of tables in BFL, we start an iterative procedure that extracts one table per iteration, creates its network, and updates the hierarchical network. When

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

Fig. 2. Two-dimensional table with hierarchical rows made up of three $(2+1)\times(2+1)$ subtables, T_1, T_2 and T_3

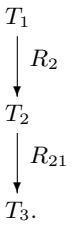

Fig. 3. Hierarchical structure tree of the example of Figure 2

BFL is empty, the hierarchical network models the hierarchical table. In the example, the first iteration extracts $T2$, and creates the network b) of Figure 4. To update the hierarchical network we only need the parent table of T_2 (i.e., T_1) and the row in the parent table expanded by T_2 (i.e., R_2). The node associated to row R_2 of table T_1 in the hierarchical network will be the linking node for the insertion of the network of T_2. We now have (1) to remove node R_2 and the grand total arc in the network of T_2; and (2) to replace node R_2 of T_1 in the hierarchical network by the previously updated network of T_2. This insertion procedure is shown in network c) of Figure 4. Table T_3 would be inserted in a similar way in the last iteration. The overall algorithm is shown in Figure 5.

More complex tables (e.g., two-dimensional tables with hierarchies in both dimensions) can be modeled as a network with additional side constraints. Although the structure of such side constraints is simple ($x_i = x_j$, i.e., the value of cell i is equal to the value of cell j), we can have a fairly large number of them. In that case, specialized algorithms for network flows with side constraints can show a similar performance than general state-of-the-art dual simplex implementations. Network-flows heuristics for CSP will, in general, not show a good performance in these situations.

4 The Shortest-Paths Heuristic

The heuristic sensibly combines and improves the approaches suggested in [2] and [4]. We will use the notation $a = (s, t)$ for an arc a with source and target nodes s and t. After modelling the table as a network, for each cell a_i two arcs $x_i^+ = (s, t)$ and $x_i^- = (t, s)$ are defined, which are respectively related to increments and decrements of the cell value. Arcs x^+ are clockwise, and are those

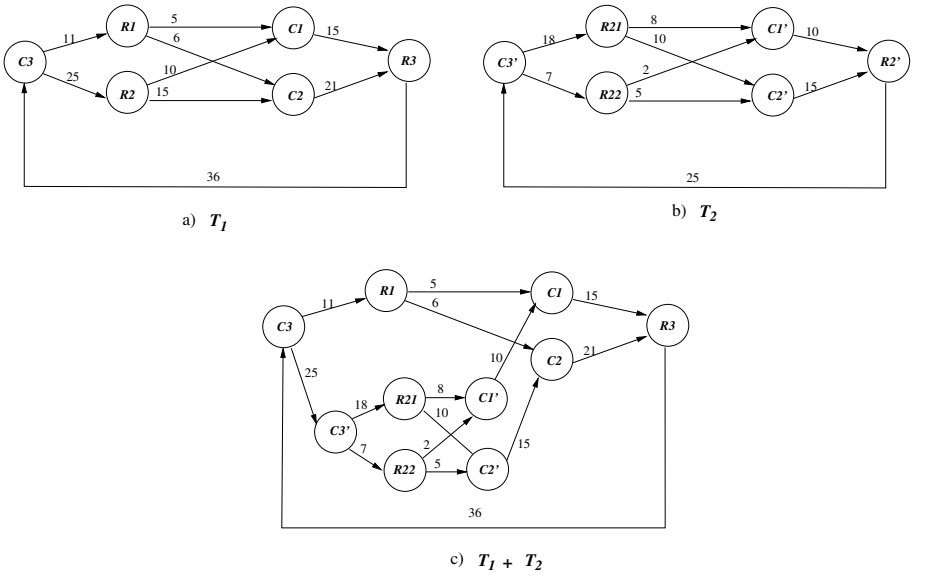

Fig. 4. First iteration of the procedure for computing the network associated to the example of Figure 2. a) Network of table T_1. b) Network of table T_2. c) Network after including T_2 in T_1. Nodes of T_2 with the same name in T_1 are marked with a "′"

Algorithm *Compute_hierarchical_network(tree of 2D tables)*
1 Compute BFL, breadth-first list of tables in tree;
2 T_0 = *extract_first(BFL)*;
3 N = *network2D(T_0)*; // N is the hierarchical network
4 **while** BFL is not empty **do**
5 T = *extract_first(BFL)*;
6 N_T = *network2D(T)*;
7 P= *parent_table_of(T)*;
8 r= *row_in_parent_table(P,T)*;
9 Update N_T removing grand total arc and node of row r;
10 Update N replacing node of row r of P by N_T;
11 **end_while**
End_algorithm

Fig. 5. Algorithm for computing the network of a two-dimensional table with at most one hierarchical dimension

that appear in the Figures 1 and 4. Arcs x^- would be obtained in those figures by changing the sense of the arrows of the arcs.

Figure 6 shows the main steps of the heuristic. Through the process, it updates the set of secondary cells $\mathcal{S}$, and two vectors *Clpl* and *Cupl* with the

```
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 = ∅;
6          while (C*_p < *_p) do
7              Set arc costs;
8              Compute the shortest-path SP from t to s;
9              T = {cells associated to arcs ∈ SP};
10             Update Clpl_i and Cupl_i, i ∈ (P ∩ T) ∪ {p};
11             S := S ∪ T \ P;
12             TT := TT ∪ T;
13         end_while
14     end_for_each
15 end_for_each
End_algorithm
```

Fig. 6. Shortest-paths heuristic for CSP in positive tables

current lower and upper protection of all the primaries. The heuristic performs one major iteration for each primary cell $p \in \mathcal{P}$ (line 2 of Figure 6), and, unlike other approaches, deals separately with the lower and upper protections (line 4). If not already done by previous primaries, p is protected through one or possibly several minor iterations (lines 6–13). At each minor iteration the arc costs of the network are set. Arcs related to cells that can not be used are assigned a very large cost. This is 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 $x_p^+ = (s, t)$. The set $\mathcal{S}$ of secondary cells is updated with the cells associated to arcs in the shortest-path (line 11). To avoid the solution of unnecessary shortest-path subproblems for next primaries, we update not only the protection levels of p, but also of all the primary cells in the shortest-path (line 10). This is a significant improvement compared to other methods. If several shortest-paths problems are needed for p, cells in previously computed shortest-paths for this primary must not be used (otherwise we can not guarantee the protection of the cell). To this end, $\mathcal{TT}$ in Figure 6 maintains the list of cells already suppressed for the protection of p.

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

- **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 $x^*_{i_1} = (t, t_{i_1})$, $x^*_{i_l} = (s_{i_1}, s)$, and $x^*_{i_j} = (t_{i_{j-1}}, s_{i_{j+1}})$ for all $j = 2, \ldots, l-1$. $\mathcal{T} = \{i_1, \ldots, i_l\}$ is the set of cells associated to the arcs in the shortest-path. Defining

$$\gamma = \min\{a_p, a_{i_j} : i_j \in \mathcal{T}\}, \tag{4}$$

then we can send a flow γ through the shortest-path in either direction. That means 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 basically the approach used in [4].
However, our heuristic exploits better the information provided by the shortest-path. It separately computes

$$\gamma^+ = \min\{a_p, a_{i_j} : x^+_{i_j} \in SP\} \qquad \gamma^- = \min\{a_{i_j} : x^-_{i_j} \in SP\}. \tag{5}$$

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 provided is γ^-. That permits to update separately and with different protections the lower and upper levels. One immediate benefit of this procedure is that the heuristic can deal with upper protections greater than the cell value (i.e., $upl_p > a_p$). Such large protections are often used for very small cell values. For instance, if only arcs $x^+_{i_j}$ appear in SP, it is possible to infinitely increase the value of cell p without compromising the feasibility of the table. Indeed, in this case the upper protection level provided by the heuristic is $\gamma^- = \infty$. This can not be done with (4). Current protection levels $Clpl$ and $Cupl$ of p and primary cells in $\mathcal{T}$ are updated using (5) in Figure 6.

- **Arc Costs.** The behaviour of the heuristic is governed by the costs of arcs x^+_i and x^-_i associated to cells a_i. Arcs not allowed in the shortest-path are assigned a very large cost. For instance, this is done for arc x^-_p to avoid a trivial shortest-path from t to s. As suggested in [4], 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 6); 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-paths subproblems to be solved. Clearly, for each of the above four categories, cells with the lowest a_i values are preferred. The particular costs used by the heuristic can be computed in a single loop over the n cells; those suggested in [4] required two such loops.
- **Shortest-Path Solver.** Shortest-paths subproblems were solved through an efficient implementation of the Dijkstra's algorithm [1, Ch. 4]. Since we are interested in the shortest-path to a single destination, a bidirectional version was used. In practice, this can be considered the most efficient algorithm for these kind of problems. As shown in the computational results of Section 5, it is orders of magnitude faster than minimum-cost network flows codes for large instances.
- **Lower Bounding Procedure.** We also included an improved version of the lower bounding procedure introduced in [11]. It is used to obtain a lower bound on the optimal solution. The relative gap between this lower bound and the solution provided by the heuristic can be used as an approximate

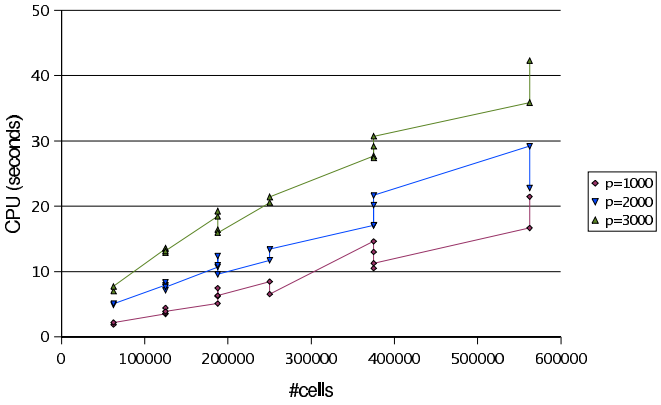

Fig. 7. CPU time vs. number of cells for two-dimensional instances

measure of how far we are from the optimum. Unlike that of [11], our procedure includes, for all primary cell p, two constraints that force the value of the other primary and secondary cells in the same row and column to be greater than $\max\{lpl_p, upl_p\}$ (otherwise, p is not protected). This new formulation provides slightly better lower bounds. The procedure, originally developed for two-dimensional tables, was extended to deal with hierarchical ones.

5 Computational Results

The heuristic of Section 4 has been implemented in C. It is currently included in the τ-Argus package [8] for tabular data protection. For testing purposes, two instances generators for two-dimensional tables were developed, similar to that of [11] (see [3] for details). A generator for hierarchical tables was also designed and implemented. They can be obtained from the author on request. We produced 54 two-dimensional instances, ranging from 62500 to 562500 cells, and with $p \in \{1000, 2000, 3000\}$, p being the number of primary cells. Cells weights were set to $w_i = a_i$ (i.e., cell value). We also generated 72 two-dimensional hierarchical tables, ranging from 1716 to 246942 cells and from 4 to 185 subtables, with $p \in \{500, 1000\}$. Cells weights were set to $w_i = a_i$ in half of the instances and $w_i = 1$ in the remaining ones. In all the cases the lower and upper protection levels were a 15% of the cell value. Primary cells were assigned a value much lower than for the other cells, which is usual in real data. All the executions were carried on a standard PC with a 1.8 GHz Pentium-4 processor. Preliminary runs on real data sets have only been performed for small two-dimensional tables [12]. In those instances, the shortest-paths heuristic was more efficient and provided better solutions than the hypercube method [7].

The results obtained are summarized in Figures 7–12. Figures 7–8 show, respectively for the two-dimensional and hierarchical tables, the CPU time in

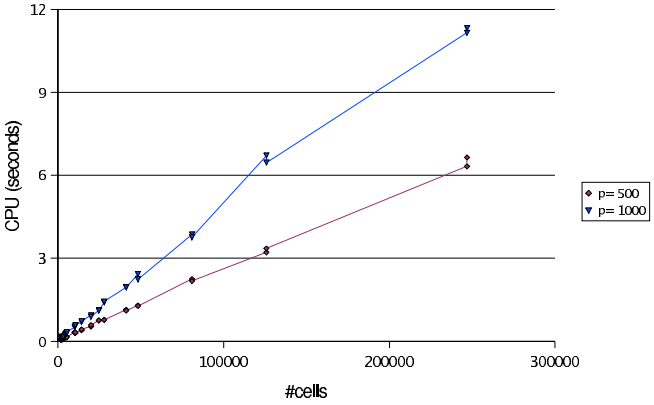

Fig. 8. CPU time vs. number of cells for hierarchical instances

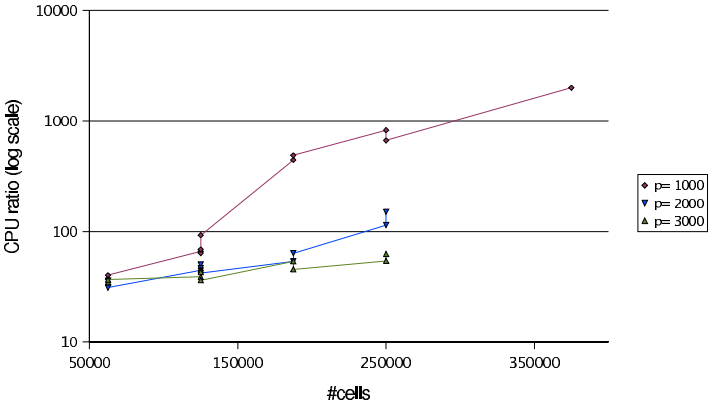

Fig. 9. CPU ratio between CPLEX and Dijkstra vs. number of cells for two-dimensional instances

seconds vs. the number of cells of the table, for the different number of primary cells. Clearly, the CPU time increases with both p and the number of cells. However, the shortest-path heuristic was able to solve all the instances in few seconds.

Figures 9–10 show, again for the two-dimensional and hierarchical tables, the efficiency of the shortest-path heuristic compared to other methods. We applied the heuristic twice, formulating the subproblems as minimum-cost network flows (as previous approaches did), and as shortest-paths. The minimum-cost network flows subproblems were solved with the network simplex option of CPLEX 7.5, a state-of-the-arc implementation. The largest instances were not solved because CPLEX required an excessive execution time. The vertical axes of the figures show the ratio of the CPU time of CPLEX 7.5 and the implementation of Di-

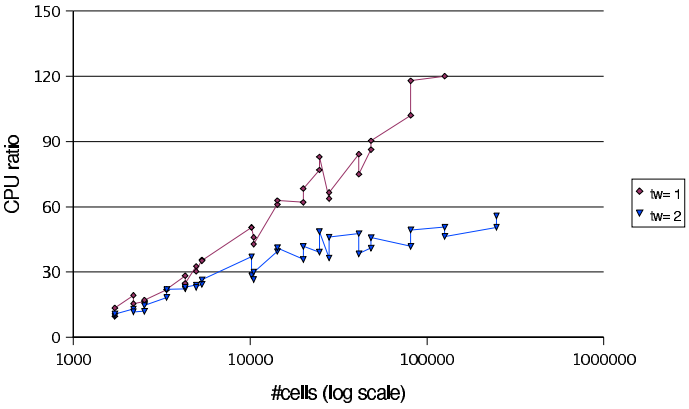

Fig. 10. CPU ratio between CPLEX and Dijkstra vs. number of cells for hierarchical instances

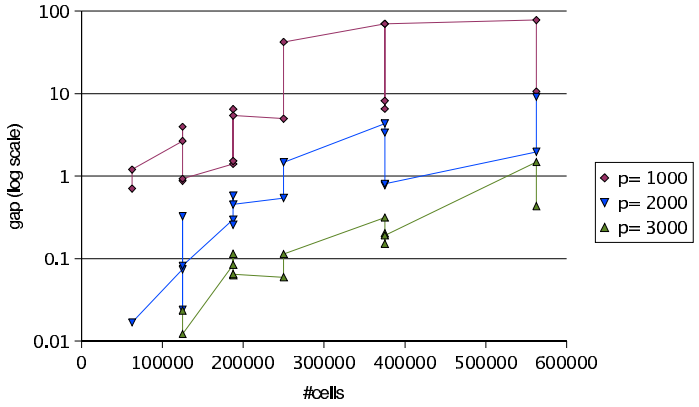

Fig. 11. Gap vs. number of cells for two-dimensional instances

jkstra's algorithm used in the heuristic. For the two-dimensional tables we plot separately the instances for the different p values. Similarly, two lines are plotted in Figure 10, one for each type of weights ($tw = 1$ and $tw = 2$ in the figure correspond to $w_i = a_i$ and $w_i = 1$, respectively). We observe that the ratio time increases with the table dimension, and it is of about 1900 for the largest instance. Therefore, it can be clearly stated that the shortest-paths formulation is instrumental in the performance of the heuristic.

Finally, Figures 11–12 show, for respectively the two-dimensional and hierarchical tables, 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. At first sight, it could be concluded that the heuristic works much better for

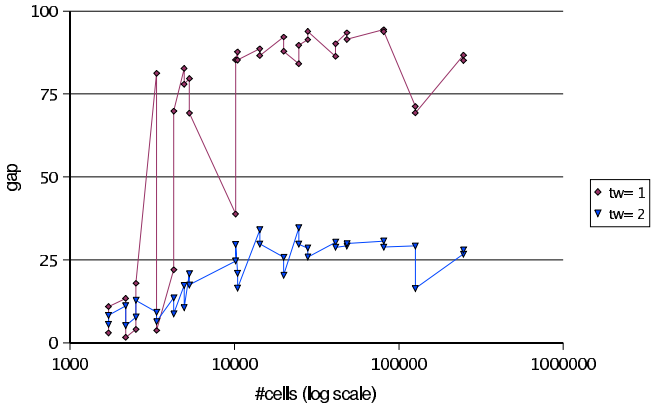

Fig. 12. Gap vs. number of cells for hierarchical instances

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 network flows heuristics (either formulating minimum-cost or shortest paths subproblems) would behave better for two-dimensional than for hierarchical tables.

6 Conclusions

From our computational experience, it can be concluded that shortest-paths heuristics are much more efficient than those based on minimum-cost network flows formulations. It was also observed that such heuristics seem to work better for two-dimensional than for hierarchical tables. To avoid over-suppression, the former would require a clean-up procedure, similar to that of [11]. However, unlike previous approaches, and for efficiency reasons, this clean-up procedure would only solve shortest-paths subproblems. This is part of the future work to be done.

References

1. Ahuja, R.K, Magnanti, T.L., Orlin, J.B.: Network Flows. Prentice Hall (1993)
2. Carvalho, F.D., Dellaert, N.P., Osório, M.D.: Statistical disclosure in two-dimensional tables: general tables. J. Am. Stat. Assoc. **89** (1994) 1547–1557
3. Castro, J.: Network flows heuristics for complementary cell suppression: an empirical evaluation and extensions. Lect. Notes in Comp. Sci. **2316** (2002) 59–73. Volume Inference Control in Statistical Databases, ed. J. Domingo-Ferrer. Springer.
4. Cox, L.H.: Network models for complementary cell suppression. J. Am. Stat. Assoc. **90** (1995) 1453–1462

5. Fischetti, M., Salazar, J.J.: Models and algorithms for optimizing cell suppression in tabular data with linear constraints. J. Am. Stat. Assoc. **95** (2000) 916–928
6. Giessing, S.: New tools for cell-suppression in τ-Argus: one piece of the CASC project work draft. Joint ECE/Eurostat Work Session on Statistical Data Confidentiality, Skopje (2001).
7. Giessing, S, Repsilber, D.: Tools and strategies to protect multiples tables with the GHQUAR cell suppression engine. Lect. Notes in Comp. Sci. **2316** (2002) 181–192. Volume Inference Control in Statistical Databases, ed. J. Domingo-Ferrer. Springer.
8. Hundepool, A.: The CASC project. Lect. Notes in Comp. Sci. **2316** (2002) 172–180. Volume Inference Control in Statistical Databases, ed. J. Domingo-Ferrer. Springer.
9. ILOG CPLEX: ILOG CPLEX 7.5 Reference Manual Library. ILOG (2001)
10. Jewett, R.: Disclosure analysis for the 1992 Economic Census. Manuscript, Economic Programming Division, Bureau of the Census (1993)
11. Kelly, J.P., Golden, B.L, Assad, A.A.: Cell Suppression: disclosure protection for sensitive tabular data. Networks **22** (1992) 28–55
12. Mas, M.: Test report on network solution for large unstructured 2 dimensional tables in τ–Argus 2.2.2. CASC Project Deliverable 6-D6. Basque Statistics Office (2003).
13. Robertson, D.: Improving Statistic's Canada cell suppression software (CONFID). Proceedings of Compstat 2000.