Models and Algorithms for Optimizing Cell Suppression in Tabular Data with Linear Constraints*

Matteo Fischetti and Juan-José Salazar-González

October 1998

Abstract

Cell suppression is a widely-used technique for protecting sensitive information in statistical data presented in tabular form. Previous works on the subject mainly concentrate on 2- and 3-dimensional tables whose entries are subject to marginal totals. In this paper we address the problem of protecting sensitive data in a statistical table whose entries are linked by a generic system of linear constraints. This very general setting covers, among others, k-dimensional tables with marginals as well as hierarchical and linked tables. In particular, we address the optimization problem known in the literature as the (complementary or secondary) Cell Suppression Problem in which the information loss due to suppression has to be minimized. We introduce a new integer linear programming model and outline an enumerative algorithm for the exact solution of the problem. The algorithm can also be used as a heuristic procedure to find near-optimal solutions. Computational results are presented on real-world instances, showing the effectiveness of the approach. In particular, we were able to solve to proven optimality all the instances in our test bed, including a 4-dimensional table with marginals and four linked tables (to our knowledge, tables of this kind were never solved optimally by previous authors).

KEY WORDS: Statistical Disclosure Control, Confidentiality, Data Protection, Integer Linear Programming, Polyhedral Combinatorics.

---

*Matteo Fischetti is Full Professor of Operations Research, "Dipartimento di Elettronica e Informatica", University of Padova, Via Gradenigo 6/a, 35131 Padova, Italy (E-mail: fisch@dei.unipd.it); and Juan José Salazar González is Associate Professor of Mathematical Programming, "Departamento de Estadística, Investigación Operativa y Computación, University of La Laguna, Av. Astrofísico Francisco Sánchez s/n, 38271 Tenerife, Spain (E-mail: jjsalaza@ull.es). The first author was supported by "Ministero della Ricerca Scientifica e Tecnologica", Italy, while the second author was supported by "Ministerio de Educacion y Ciencia", Spain. This work was partially supported by the European Union - ESPRIT project 20462-SDC on Statistical Disclosure Control, coordinated by Leon Willenborg (Central Bureau of Statistics, Voorburg, The Netherlands). The authors thank Angela Dale (Center for Census and Survey Research, Manchester, U.K.), Anco Hunderpool (Central Bureau of Statistics, Voorburg, The Netherlands), and Ramesh Dandekar (U.S. Department of Energy, Washington, U.S.A.) for having provided them with the real-world instances used for computational testing.

	A	B	C	Total
Activity I	20	50	10	80
Activity II	8	19	**22**	49
Activity III	17	32	12	61
Total	45	101	44	190

(a) Original table

	A	B	C	Total
Activity I	20	50	10	80
Activity II	*	19	*	49
Activity III	*	32	*	61
Total	45	101	44	190

(b) Published table

Figure 1: Investment of enterprises by activity and region.

1 Introduction

A statistical agency collects data to be processed and published. Raw material is information obtained from individual respondents. Usually, this data is obtained under a pledge of confidentiality: statistical agencies have the responsibility of not releasing any data or data summaries from which individual respondent information can be revealed (*sensitive data*). On the other hand, statistical agencies aim at publishing as much information as possible. This results into a trade-off between privacy rights and information loss, an issue of primary importance in practice. We refer the interested reader to, e.g., Willenborg and Waal (1996) for an in-depth analysis of statistical disclosure control methodologies.

Starting in 1996, the European Union supported through EUROSTAT (the European statistical office) a 3-year ESPRIT research project aimed at developing and testing new methodologies for statistical disclosure control. The project, coordinated by Dr. Leon Willenborg from CBS (Central Bureau of Statistics, Voorburg, The Netherlands), involves several research groups from both academia and national statistical offices. We participate in the project for the definition of mathematical models and solution algorithms for protecting sensitive information in tabular data. A preliminary version of our codes, capable of dealing with 2-, 3- and 4-dimensional tables with marginals, has been embedded within τ-ARGUS, a prototype software package for statistical disclosure control under development at CBS.

Cell suppression is a widely-used technique for disclosure avoidance. Here is an introductory example, taken from Willenborg and Waal (1996). Figure 1(a) exhibits a table giving the investment of enterprises (per millions of guilders), classified by activity and region. Let us assume that the information in the cell (2,3) —the one corresponding to Activity II and Region C— is considered confidential by the statistical office, according to a certain criterion (as discussed, e.g., in Willenborg and Waal (1996)). This cell is then viewed as a *sensitive cell* to be suppressed (*primary suppression*). But that is not enough: by using the marginal totals, an *attacker* interested in the disclosure of the sensitive cell can easily recompute its missing value. Then other table entries have to be suppressed as well (*secondary or complementary suppressions*). For example, with the missing entries in Figure 1(b) an attacker cannot disclose the nominal value of the sensitive cell exactly, although he/she can still compute a range for the values of this cell which are consistent with the published entries. Indeed, the minimum value $\underline{y}_{23}$ for the sensitive cell can be computed by solving a linear program in which the values y_{ij} for the missing cells (i,j) are treated as unknowns, namely

$$\underline{y}_{23} := \min y_{23}$$

subject to

$$\begin{array}{ccccl}
y_{21} & & +y_{23} & & = 30 \\
 & y_{31} & & +y_{33} & = 29 \\
y_{21} & +y_{31} & & & = 25 \\
 & & y_{23} & +y_{33} & = 34 \\
y_{21}, & y_{23}, & y_{31}, & y_{33} & \geq 0.
\end{array}$$

Notice that the right-hand side values are known to the attacker, as they can be obtained as the difference between the marginal and the published values in a row/column.

The maximum value $\overline{y}_{23}$ for the sensitive cell can be computed in a perfectly analogous way by solving the linear program of maximizing y_{23} subject to the same constraints as before.

In the example, $\underline{y}_{23} = 5$ and $\overline{y}_{23} = 30$, i.e., the sensitive information is "protected" within the *protection interval* [5,30]. If this interval is considered sufficiently wide by the statistical office, the sensitive cell is called *protected*; otherwise new suppressions are needed. Notice that the extreme values of interval $[5, 30]$ are only attained if the cell corresponding to Activity II and Region A takes the quite unreasonable values of 0 and 25. Bounding the cell variation to $\pm 50\%$ of the nominal value (say) results in the more realistic protection interval $[18, 26]$. This example illustrates the need of considering "external bounds" on the cell values when computing realistic protection intervals.

The *Cell Suppression Problem* (CSP) consists in finding a set of cells whose suppression guarantees the protection of all the sensitive cells against an attacker, with minimum loss of information associated with the suppressed entries. This problem belongs to the class of the strongly $\mathcal{NP}$-hard problems (see, e.g., Kelly, Golden and Assad (1992), Geurts (1992), Kao (1996)), meaning that it is very unlikely the existence of an algorithm for the exact solution of CSP which guarantees an efficient (i.e., polynomial-time) performance for all possible input instances.

Previous works on CSP from the literature mainly concentrate on heuristic algorithms for 2-dimensional tables with marginals; see Cox (1980, 1995), Sande (1984), Kelly, Golden and Assad (1992), and Carvalho, Dellaert and Osório (1994), among others. Kelly (1990) proposed a mixed-integer linear programming formulation for 2- and

3-dimensional tables with marginals, which requires a huge number of variables and constraints. Geurts (1992) refined slightly the 2-dimensional model, and reported computational experiences on small-size instances, the largest instance solved to optimality being a table with 20 rows, 6 columns and 17 sensitive cells. Gusfield (1988) gave a polynomial-time algorithm for a special case of the problem in 2-dimensional tables. Recently, Fischetti and Salazar (1998) presented a new method capable of solving to proven optimality, on a personal computer, 2-dimensional instances with up to 250,000 cells and 10,000 sensitive cells.

Heuristics for 3-dimensional tables with marginals have been proposed in Robertson (1994), Sande (1984), and Dellaert and Luijten (1996).

In this paper we address the problem of protecting sensitive data in a statistical table whose entries are linked by a generic system of linear equations. This very general setting covers, among others, k-dimensional tables with marginals as well as hierarchical and linked tables. To our knowledge, this is the first attempt to model and solve the complementary cell suppression problem in this generalized context.

The paper is organized as follows. A formal description of the cell suppression problem is given in Section 2. Section 3 introduces and discusses new mathematical models for the problem. In particular, a new Integer Linear Programming model is proposed, having a 0-1 decision variable for each potential suppression but requiring an exponential number of linear constraints enforcing the protection level requirements. Section 4 addresses efficient methods for solving the proposed model within the so-called branch-and-cut framework. Computational results are given in Section 5, where 10 real-world instances are optimally solved, on a PC, within acceptable computing time. In particular, we were able to solve to proven optimality a 4-dimensional table with marginals and four linked tables. Some conclusions are finally drawn in Section 6.

Preliminary versions of the present paper were presented by the authors at the meeting *Statistical Disclosure Protection '98*, Lisbon, March 25–27, 1998, and at the *International Symposium on Linked Employer-Employee Data*, Washington, May 21–22, 1998.

2 The Cell Suppression Problem

We next give a formal definition of the cell suppression problem we address in this paper.

A *table* is a data vector $a = [a_1, \ldots, a_n]$ whose entries satisfy a given set of linear constraints known to a possible attacker, namely

$$My = b \tag{1}$$

$$lb_i \le y_i \le ub_i \quad \text{for all } i = 1, \ldots, n. \tag{2}$$

In other words, system (1)–(2) models the whole *a priori* information on the table known to an attacker. Typically, each equation in (1) corresponds to a marginal entry, whereas inequalities (2) enforce the "external bounds" known to the attacker. In the case of k-dimensional tables with marginals, each equation in (1) is of the type $\sum_{j \in Q_i} y_j - y_i = 0$, where index i corresponds to a marginal entry and index set Q_i to the associated internal table entries. Therefore, in this case M is a $\{0, +1, -1\}$-matrix and $b = 0$. Moreover, in case $k = 2$ the linear system (1)–(2) can be represented in a natural way as a *network*, a property having important theoretical and practical implications; see, e.g., Cox (1980, 1995), Kelly, Golden and Assad (1992), and Fischetti and Salazar (1998). Unfortunately, this nice structure is not preserved for $k \geq 3$, unless the table decomposes into a set of independent 2-dimensional subtables.

A *cell* is an index corresponding to an entry of the table.

Given a *nominal* table a, let $PS = \{i_1, \ldots, i_p\}$ be the set of sensitive cells to be protected, as identified by the statistical office according to some criteria. For each sensitive cell i_k ($k = 1, \ldots, p$), the statistical office provides three nonnegative values, namely LPL_k, UPL_k and SPL_k, called *Lower Protection Level*, *Upper Protection Level* and *Sliding Protection Level*, respectively, whose role will be discussed next. In typical applications, these values are computed as a certain percentage of the nominal value a_{i_k}.

A *suppression pattern* is a subset of cells $SUP \subseteq \{1, \ldots, n\}$ corresponding to the unpublished cells. A *consistent table* with respect to a given suppression pattern SUP and to a given nominal table is a vector $y = [y_1, \ldots, y_n]$ satisfying

$$
\begin{aligned}
My &= b \\
lb_i \leq y_i &\leq ub_i \qquad \text{for all } i \in SUP \\
y_i &= a_i \qquad \text{for all } i \notin SUP,
\end{aligned}
$$

where the latter equations impose that the components of y associated with the published entries coincide with the nominal ones. In other words, any consistent table gives a feasible way the attacker can fill the missing entries of the published table.

A suppression pattern is considered *feasible* by the statistical office if it guarantees the required protection intervals against an attacker, in the sense that, for each sensitive cell i_k ($k = 1, \ldots, p$), there exist two feasible tables, say f^k and g^k, such that $f^k_{i_k} \leq a_{i_k} - LPL_k$, $g^k_{i_k} \geq a_{i_k} + UPL_k$, and $g^k_{i_k} - f^k_{i_k} \geq SPL_k$. In other words, it is required that $\underline{y}_{i_k} \leq a_{i_k} - LPL_k$, $\overline{y}_{i_k} \geq a_{i_k} + UPL_k$, and $\overline{y}_{i_k} - \underline{y}_{i_k} \geq SPL_k$, where

$$
\left.
\begin{array}{rrcll}
\underline{y}_{i_k} := & \min y_{i_k} & & & \\
 & My & = & b & \\
 & lb_i \leq y_i & \leq & ub_i & \text{for all } i \in SUP \\
 & y_i & = & a_i & \text{for all } i \notin SUP
\end{array}
\right\}
$$

and

$$\left.\begin{array}{rcll} \overline{y}_{i_k} := & \max y_{i_k} & & \\ & My & = & b \\ & lb_i \le y_i & \le & ub_i \quad \text{for all } i \in SUP \\ & y_i & = & a_i \quad \text{for all } i \notin SUP \end{array}\right\}$$

As already mentioned, the statistical office is interested in selecting, among all feasible suppression patterns, a one with minimum information loss. This issue can be modeled by associating a *weight* $w_i \ge 0$ with each entry of the table, and by requiring the minimization of the overall weight of the suppressed cells, namely $\sum_{i \in SUP} w_i$; in typical applications, the weights w_i provided by the statistical office are proportional to a_i or to $\log(a_i)$. The resulting combinatorial problem is known in the literature as the (complementary or secondary) *Cell Suppression Problem*, or CSP for short.

3 A New Integer Linear Programming Model

In the sequel, for notational convenience we define the relative external bounds:

$$LB_i := a_i - lb_i \ge 0$$

and

$$UB_i := ub_i - a_i \ge 0,$$

i.e., the range of feasible values for cell i known to the attacker is $[a_i - LB_i, a_i + UB_i]$.

In order to obtain a Mixed-Integer Linear Programming (MILP) model for CSP we introduce a binary variable $x_i = 1$ for each cell i, where $x_i = 1$ if $i \in SUP$ (suppressed), and $x_i = 0$ otherwise (published). Clearly, we can fix $x_i = 0$ for all cells that have to be published (if any), and $x_i = 1$ for all cells that have to be suppressed (sensitive cells). Using this set of variables, the model is of the form

$$\min \sum_{i=1}^{n} w_i x_i \tag{3}$$

subject to $x \in \{0,1\}^n$ and, for each sensitive cell i_k ($k = 1, \ldots, p$):

$$\left.\begin{array}{c}\text{The suppression pattern associated with } x \text{ satisfies the} \\ \text{upper protection level requirement with respect to cell } i_k\end{array}\right\} \tag{4}$$

$$\left.\begin{array}{c}\text{The suppression pattern associated with } x \text{ satisfies the} \\ \text{lower protection level requirement with respect to cell } i_k\end{array}\right\} \tag{5}$$

$$\left.\begin{array}{c}\text{The suppression pattern associated with } x \text{ satisfies the} \\ \text{sliding protection level requirement with respect to cell } i_k\end{array}\right\} \tag{6}$$

3.1 The classical model

A possible way to express conditions (4)–(6) through linear constraints requires the explicit introduction, for each $k = 1, \ldots, p$, of auxiliary continuous variables $f^k = [f_1^k, \ldots, f_n^k]$ and $g^k = [g_1^k, \ldots, g_n^k]$ defining tables which are consistent with the published one, and certificating the feasibility of the suppression pattern associated with x. This is in the spirit of the MILP model proposed by Kelly (1990) for 2-dimensional tables with marginals. The resulting MILP model then reads:

$$\min \sum_{i=1}^{n} w_i x_i \tag{7}$$

subject to $x \in \{0,1\}^n$ and, for each sensitive cell i_k ($k = 1, \ldots, p$):

$$\left.\begin{array}{rcll} M & f^k & = b & \\ a_i - LB_i\, x_i \leq & f_i^k & \leq a_i + UB_i\, x_i & \text{for } i = 1, \ldots, n \end{array}\right\} \tag{8}$$

$$\left.\begin{array}{rcll} M & g^k & = b & \\ a_i - LB_i\, x_i \leq & g_i^k & \leq a_i + UB_i\, x_i & \text{for } i = 1, \ldots, n \end{array}\right\} \tag{9}$$

$$g_{i_k}^k \geq a_{i_k} + UPL_k \tag{10}$$

$$f_{i_k}^k \leq a_{i_k} - LPL_k \tag{11}$$

$$g_{i_k}^k - f_{i_k}^k \geq SPL_k. \tag{12}$$

Notice that the lower/upper bounds on the variables f_i^k and g_i^k in (8) and (9) depend on x_i, so as to enforce $f_i^k = g_i^k = a_i$ whenever $x_i = 0$ (cell i is not suppressed), and $lb_i \leq f_i^k \leq ub_i$ and $lb_i \leq g_i^k \leq ub_i$ otherwise (cell i is suppressed). Therefore, (8) and (9) stipulate the consistency of f^k and g^k, respectively, with the published table, whereas (10), (11) and (12) translate the protection level requirements (4), (5) and (6), respectively.

Standard MILP solution techniques such as branch-and-bound or cutting-plane methods (see, e.g., Nemhauser and Wolsey (1988)) require the solution of the Linear Programming (LP) relaxation of the model in hand, obtained by relaxing conditions $x_i \in \{0,1\}$ into $0 \leq x_i \leq 1$ for all i. However, even the LP relaxation of model (7)–(12) is very difficult to solve, in that it involves a really huge number of auxiliary variables f_i^k and g_i^k and of linking constraints between the x and the auxiliary variables. For example, for a 100×100 table with marginals having 5% sensitive cells, the model needs more than 10,000,000 variables and 20,000,000 constraints, a size that cannot be handled explicitly by the today LP technology.

We next propose a new model inspired by Benders' decomposition (see, e.g., Nemhauser and Wolsey (1988)) The idea is to use standard LP duality theory to avoid the introduction of the auxiliary variables f^k and g^k ($k = 1, \ldots, p$) along with the associated linking constraints. In the new model, the protection level requirements are in fact imposed by means of families of linear constraints in the space of the x-variables only. Before formulating the new model, we need a characterization of the vectors x for which system (8)–(12) admits feasible f^k and g^k solutions, which is obtained as follows.

3.2 Imposing the upper protection level requirements

Assume that x is a given (arbitrary but fixed) parameter, and consider any given sensitive cell i_k ($k = 1, \ldots, p$) along with the associated upper protection level requirement. Clearly, the linear system (9) and (10) admits a feasible solution g^k if and only if

$$a_{i_k} + UPL_k \leq \overline{y}_{i_k},$$

where $\overline{y}_{i_k}$ is the optimal value of the linear program:

$$\overline{y}_{i_k} := \max y_{i_k} \tag{13}$$

subject to

$$\begin{aligned} My &= b && && (14)\\ y_i &\leq a_i + UB_i x_i && \text{for all } i = 1, \ldots, n && (15)\\ -y_i &\leq -a_i + LB_i x_i && \text{for all } i = 1, \ldots, n. && (16) \end{aligned}$$

This is a parametric LP problem in the y-variables only, with variable upper/lower bounds depending on the given parameter x. We call (13)–(16) the *attacker subproblem* associated with the upper protection of sensitive cell i_k, with respect to the given parameter x. By LP duality, this subproblem is equivalent to the linear program

$$\overline{y}_{i_k} = \min \gamma^t b + \sum_{i=1}^{n} [\alpha_i (a_i + UB_i x_i) - \beta_i (a_i - LB_i x_i)] \tag{17}$$

subject to

$$\left.\begin{array}{c} \alpha^t - \beta^t + \gamma^t M = e^t_{i_k} \\ \alpha \geq 0,\ \beta \geq 0,\ \gamma \text{ unrestricted in sign} \end{array}\right\} \tag{18}$$

where e_{i_k} denotes the i_k-th column of the identity matrix of order n, and γ, α and β are the dual vectors associated with constraints (14), (15) and (16), respectively.

It then follows that the linear system (9) and (10) has a feasible solution if and only if

$$a_{i_k} + UPL_k \leq \overline{y}_{i_k} = \min \left\{ \gamma^t b + \sum_{i=1}^{n} [\alpha_i (a_i + UB_i x_i) - \beta_i (a_i - LB_i x_i)] : (18) \text{ holds} \right\},$$

i.e., if and only if

$$a_{i_k} + UPL_k \le \gamma^t\, b + \sum_{i=1}^{n}[\alpha_i\,(a_i + UB_i x_i) - \beta_i\,(a_i - LB_i x_i)] \text{ for all } \alpha,\beta,\gamma \text{ satisfying (18).}$$

Because of (18) and $Ma = b$, we have $\gamma^t b + \sum_{i=1}^{n}(\alpha_i a_i - \beta_i a_i) = \gamma^t Ma + (\alpha-\beta)^t a = e_{i_k}^t a = a_{i_k}$. Hence the above system can be rewritten as

$$\sum_{i=1}^{n}(\alpha_i UB_i + \beta_i LB_i)x_i \ge UPL_k, \quad \text{for all } \alpha,\beta,\gamma \text{ satisfying (18).} \tag{19}$$

In other words, system (19) defines a family of linear constraints, in the x-variables only, expressing the condition (4) concerning the upper protection level requirement for sensitive cell i_k.

Notice that (19) contains in principle an infinite number of constraints, each associated with a different point (α,β,γ) of the polyhedron defined by (18). However, it is well known that only the extreme points (and rays) of such polyhedron can lead to undominated constraints (19), i.e., a finite number of such constraints is sufficient to impose the upper protection level requirement for a given sensitive cell i_k.

3.3 Imposing the lower protection level requirements

Analogously, the lower protection level requirement for a given cell i_k is equivalent to imposing $\underline{y}_{i_k} \le a_{i_k} - LPL_k$, where

$$\underline{y}_{i_k} := \min\{y_{i_k} : (14)\text{–}(16) \text{ hold }\} \equiv -\max\{-y_{i_k} : (14)\text{–}(16) \text{ hold }\}. \tag{20}$$

This is called the *attacker subproblem* associated to the lower protection of sensitive cell i_k, with respect to the given parameter x. By LP duality, this subproblem is equivalent to the linear program

$$\underline{y}_{i_k} = -\min \gamma^t\, b + \sum_{i=1}^{n}[\alpha_i\,(a_i + UB_i x_i) - \beta_i\,(a_i - LB_i x_i)] \tag{21}$$

subject to

$$\left.\begin{array}{c} \alpha^t - \beta^t + \gamma^t M = -e_{i_k}^t \\ \alpha \ge 0,\ \beta \ge 0,\ \gamma \text{ unrestricted in sign.} \end{array}\right\} \tag{22}$$

Hence the lower protection level requirement (5) for cell i_k can be formulated as

$$a_{i_k} - LPL_k \ge -\left(\gamma^t\, b + \sum_{i=1}^{n}[\alpha_i\,(a_i + UB_i x_i) - \beta_i\,(a_i - LB_i x_i)]\right) \text{ for all } \alpha,\beta,\gamma \text{ satisfying (22)}$$

or, equivalently,

$$\sum_{i=1}^{n}(\alpha_i UB_i + \beta_i LB_i)x_i \ge LPL_k \quad \text{for all } \alpha,\beta,\gamma \text{ satisfying (22).}$$

3.4 Imposing the sliding protection level requirements

As to the sliding protection level for sensitive cell i_k, the requirement is that

$$SPL_k \leq \overline{y}_{i_k} - \underline{y}_{i_k} := \max\{y_{i_k} : \text{(14)–(16) hold }\} + \max\{-y_{i_k} : \text{(14)–(16) hold }\}.$$

Again, by LP duality, this condition is equivalent to

$$\begin{aligned} SPL_k \quad \leq \quad & \min\{\gamma^t\, b + \sum_{i=1}^{n}[\alpha_i\,(a_i + UB_i x_i) - \beta_i\,(a_i - LB_i x_i)] : \text{(18) holds }\} + \\ & \min\{\gamma^t\, b + \sum_{i=1}^{n}[\alpha_i\,(a_i + UB_i x_i) - \beta_i\,(a_i - LB_i x_i)] : \text{(22) holds }\}. \end{aligned}$$

Therefore, the feasibility condition can now be formulated by requiring

$$\begin{aligned} & SPL_k \leq (\gamma + \gamma')^t\, b + \sum_{i=1}^{n}[(\alpha_i + \alpha'_i)\,(a_i + UB_i x_i) - (\beta_i + \beta'_i)\,(a_i - LB_i x_i)] \\ & \qquad \qquad \text{for all } \alpha, \beta, \gamma \text{ satisfying (18) and for all } \alpha', \beta', \gamma' \text{ satisfying (22),} \end{aligned}$$

or, equivalently,

$$\begin{aligned} & \sum_{i=1}^{n}[(\alpha_i + \alpha'_i) UB_i + (\beta_i + \beta'_i) LB_i]\, x_i \geq SPL_k \\ & \qquad \qquad \text{for all } \alpha, \beta, \gamma \text{ satisfying (18) and for all } \alpha', \beta', \gamma' \text{ satisfying (22).} \end{aligned}$$

3.5 The new model

The above characterization of the feasible vectors x leads to the following new integer linear model for CSP:

$$\min \sum_{i=1}^{n} w_i x_i \tag{23}$$

subject to $x \in \{0,1\}^n$ and, for each sensitive cell i_k $(k = 1, \ldots, p)$:

$$\left.\begin{aligned} & \textstyle\sum_{i=1}^{n}(\alpha_i UB_i + \beta_i LB_i) x_i \geq UPL_k \\ & \qquad \text{for all extreme points } (\alpha, \beta, \gamma) \text{ satisfying (18)} \end{aligned}\right\} \tag{24}$$

$$\left.\begin{aligned} & \textstyle\sum_{i=1}^{n}(\alpha_i UB_i + \beta_i LB_i) x_i \geq LPL_k \\ & \qquad \text{for all extreme points } (\alpha, \beta, \gamma) \text{ satisfying (22)} \end{aligned}\right\} \tag{25}$$

$$\left.\begin{aligned} & \textstyle\sum_{i=1}^{n}[(\alpha_i + \alpha'_i) UB_i + (\beta_i + \beta'_i) LB_i] x_i \geq SPL_k \\ & \qquad \text{for all extreme points } (\alpha, \beta, \gamma) \text{ satisfying (18) and} \\ & \qquad \text{for all extreme points } (\alpha', \beta', \gamma') \text{ satisfying (22)} \end{aligned}\right\} \tag{26}$$

Notice that all the left-hand side coefficients of variables x_i are nonnegative. As a consequence, all the constraints with zero right-hand side value need not to be included in the model, as they do not correspond to a proper protection level requirement.

We call (24)–(26) the *capacity constraints* in analogy with similar constraints introduced in Fischetti and Salazar (1998) for enforcing a sufficient "capacity" of certain cuts in the network representation of the CSP on 2-dimensional tables with marginals. Intuitively, the capacity constraints force to suppress (i.e., to set $x_i = 1$) a sufficient number of cells whose positions within the table and contributions to the overall protection are specified by the dual variables (α, β, γ) of the attacker subproblems.

4 Solving the New Model

The solution of model (23)–(26) can be achieved through an enumerative scheme, based on the solution of the LP relaxation obtained by weakening conditions $x_i \in \{0, 1\}$ into $0 \leq x_i \leq 1$ for each cell i. In order to achieve integrality, we use an enumerative scheme in which the fractional components of the optimal solution of the LP relaxation are iteratively fixed to either 0 or 1, in a recursive way. We refer the reader to, e.g., Nemhauser and Wolsey (1988) for a detailed description of enumerative methods in integer programming.

4.1 Solving the LP relaxation

At first glance, the LP relaxation of the new model seems even worse to solve than the one of the classical model (7)–(12). Actually, in the new model the reduction on the number of variables was obtained through the introduction of a really huge number of capacity constraints, which cannot be handled explicitly by any LP solver. On the other hand, for a given CSP instance just a few capacity constraints are typically needed in the LP relaxation to force the protection level requirements. In fact, our experience shows that the number of capacity constraints that need to be incorporated explicitly in the model is of the same order of magnitude as the number of sensitive cells, a quite reasonable figure for practical solution approaches.

Unfortunately, we cannot decide in advance which capacity constraints are the relevant ones. Instead, we can apply a dynamic constraint generation technique that identifies them on the fly during the solution of the LP relaxation. This methodology is commonly known as *branch-and-cut* (see Padberg and Rinaldi (1991), and Caprara and Fischetti (1997) for a recent annotated bibliography), and often leads to very effective solution schemes for hard optimization problems. For example, very large instances of the famous Traveling Salesman Problem involving the visit of up to 13,509 cities have been recently solved to proven optimality by using this approach (Applegate, Bixby, Chvátal and Cook 1998).

To be more specific, we approach the solution of the LP relaxation of the new model through the following iterative scheme.

We start by solving the so-called *master LP*:

$$\min\{\sum_{i=1}^{n} w_i x_i : x_{i_1} = \ldots = x_{i_p} = 1,\ x \in [0,1]^n\},$$

in which we only impose the suppression of the sensitive cells. Let x^* be the optimal solution found. Our order of business it to check whether the vector x^* (now viewed as a given parameter) guarantees the required protection levels. In geometrical terms, this is equivalent to finding an hyperplane in the x-space that separates x^* from the polyhedron defined by the capacity constraints, hence it is called the *separation problem* associated with the family of capacity constraints (24)–(26). This separation problem can be solved efficiently as follows.

For each sensitive cell i_k, in turn, we apply the following steps:

1. We first solve the attacker subproblem (13)–(16) for $x = x^*$ and check whether $a_{i_k} + UPL_k \le \overline{y}_{i_k}$. If this is the case, then x^* satisfies the upper protection level requirement for the given i_k, hence all the exponentially many capacity constraints (24) are certainly fulfilled. Otherwise, the optimal dual solution $(\overline{\alpha}, \overline{\beta}, \overline{\gamma})$ of the attacker subproblem satisfies (18) and

   $$\overline{\gamma}^t b + \sum_{i=1}^{n} [\overline{\alpha}_i (a_i + UB_i x_i^*) - \overline{\beta}_i (a_i - LB_i x_i^*)] = \overline{y}_{i_k} < a_{i_k} + UPL_k,$$

   hence it induces a capacity constraint $\sum_{i=1}^{n} (\overline{\alpha}_i UB_i + \overline{\beta}_i LB_i) x_i \ge UPL_k$ in family (24) that is violated by x^*. This constraint is then added to the master LP.

2. We then check whether x^* satisfies the lower protection level requirement for i_k, which requires the solution of the attacker subproblem (20) associated with the lower protection level of cell i_k, and possibly add to the master LP the violated capacity constraint in family (25) associated with an optimal dual solution $(\underline{\alpha}, \underline{\beta}, \underline{\gamma})$ of the corresponding attacker supbroblem.

3. Finally, we check whether x^* satisfies the sliding protection level for i_k. This simply requires checking whether the values $\overline{y}_{i_k}$ and $\underline{y}_{i_k}$ computed in the previous steps satisfy $\overline{y}_{i_k} - \underline{y}_{i_k} \ge SPL_k$. If this is not the case, setting $(\alpha, \beta, \gamma) := (\overline{\alpha}, \overline{\beta}, \overline{\gamma})$ and $(\alpha', \beta', \gamma') := (\underline{\alpha}, \underline{\beta}, \underline{\gamma})$ leads to a violated capacity cut (26) that is added to the master LP.

Clearly, Steps 1 and 3 (respectively, Steps 2 and 3) can be avoided if $UPL_k = SPL_k = 0$ (respectively, $LPL_k = SPL_k = 0$).

After having considered all sensitive cells i_k we have two possible cases. If no capacity constraint has been added to the master LP, then all of them are satisfied by x^* which is therefore an optimal solution of the LP relaxation of model (23)–(26). Otherwise, the master LP amended by the new capacity constraints is reoptimized and the approach is iterated on the (possibly fractional) optimal solution x^* of the new master LP.

By using the above cutting-plane scheme one can solve efficiently the overall LP relaxation of our model. Actually, it can be shown that this approach can be implemented to run in polynomial time; see, e.g., Nemhauser and Wolsey (1988) for a discussion about the equivalence of optimization and separation.

4.2 Strengthening the LP relaxation

The effectiveness of the branch-and-cut approach greatly depends on how tightly the LP relaxation approximates the integer solution set. In this respect, adding to the model new classes of linear constrains can be greatly beneficial, in that the additional constraints (which are redundant when the integrality condition on the variables is active) can produce tightened LP relaxations, and hence a significant speed-up in the overall problem resolution.

We next outline some families of additional constraints that we found effective in our computational experience. As in the case of capacity constraints, these new inequalities are added to the LP relaxation on the fly, when they are violated by the solution x^* of the current master LP. This requires the exact/heuristic solution of the separation problem associated with each new family of constraints.

4.2.1 Strengthened capacity constraints

Capacity constraints have been derived without using the information on the integrality of the x-variables. Then let

$$\sum_{i=1}^{n} s_i x_i \geq s_0 \tag{27}$$

represent any capacity inequality (24)–(26), whose coefficients $s_1, \ldots, s_n$ are all nonnegative. We claim that any integer vector $x \geq 0$ satisfying (27) must also satisfy

$$\sum_{i=1}^{n} \min\{s_i, s_0\} x_i \geq s_0. \tag{28}$$

Indeed, let $T := \{i \in \{1, \ldots, n\} : s_i > s_0\}$. Given any integer $x \geq 0$ satisfying (27), if $x_i = 0$ for all $i \in T$ then $\sum_{i=1}^{n} \min\{s_i, s_0\} x_i = \sum_{i=1}^{n} s_i x_i \geq s_0$. Otherwise, we have $\sum_{i=1}^{n} \min\{s_i, s_0\} x_i \geq s_0 x_j \geq s_0$, where j is any index in T such that $x_j \neq 0$ (hence $x_j \geq 1$).

Notice that condition (28) is stronger than (27) when x has fractional components, a case of interest when solving the LP relaxation of our model.

As already discussed, the use of the strengthened capacity constraints requires to address the associated separation problem. In our implementation we use a simple separation heuristic in which we apply the strengthening procedure only to the capacity constraints associated with the optimal dual solutions of the attacker subproblems, computed as described in Subsection 4.1.

Although very simple, the above improvement is very effective in practice, mainly when UPL_k, LPL_k, and/or SPL_k are small in comparison with the given external relative bounds UB_i and LB_i. Indeed, our computational experience shows that deactivating this improvement has a dramatic impact on the model quality, and hence on the convergence properties of our solution algorithm.

4.2.2 Cover inequalities

Following the seminal work of Crowder, Johnson and Padberg (1983) on the solution of general integer programming models, we can observe that each single capacity inequality implies a number of "more combinatorial" restrictions. To be specific, let $\sum_i s'_i x_i \geq s_0$ represent any strengthened capacity constraints in (28), whose coefficients $s'_i := \min\{s_i, s_0\}$ are all nonnegative. Clearly, one has to suppress at least one cell for any subset $Q \subseteq \{1, \ldots, n\}$ such that $\sum_{i \notin Q} s'_i < s_0$, a condition that can be expressed by means of the following *cover inequalities*:

$$\sum_{i \in Q} x_i \geq 1, \text{ for each cell subset } Q : \sum_{i \notin Q} s'_i < s_0, \tag{29}$$

These inequalities can easily be improved to their *lifted* form:

$$\sum_{i \in EXT(Q)} x_i \geq |EXT(Q)| - |Q| + 1, \text{ for all cell subset } Q : \sum_{i \notin Q} s'_i < s_0, \tag{30}$$

where $EXT(Q) := Q \cup \{i : s'_i \geq \max\{s'_j : j \in Q\}\}$. We refer the interested reader to, e.g., Nemhauser and Wolsey (1988) for a discussion on the validity of lifted cover inequalities, and for possible procedures for solving the associated separation problem.

4.2.3 Bridgeless inequalities

As the weights w_i are assumed to be nonnegative, every CSP instance has an optimal solution in which no suppression is redundant. Therefore, one can require that the value of each cell with $x_h = 1$ cannot be recomputed exactly. This is equivalent to associate a very small fictitious sliding protection level $\epsilon > 0$ to each suppressed non-sensitive cell, and to set-up the associated attacker subproblems with the requirement that

$$\overline{y}_h - \underline{y}_h \geq \epsilon x_h.$$

Notice that this condition is only active for suppressed cells h, and vanishes when $x_h = 0$.

As already discussed, the above condition on the optimal values $\overline{y}_h$ and $\underline{y}_h$ of the attacker subproblems can be enforced by the following class of capacity constraints

$$\sum_{i=1}^{n} \left[(\alpha_i + \alpha'_i) UB_i + (\beta_i + \beta'_i) LB_i \right] x_i \geq \epsilon x_h, \tag{31}$$

valid for all extreme points (α, β, γ) satisfying (18) and for all extreme points $(\alpha', \beta', \gamma')$ satisfying (22), with cell h playing the role of i_k. These constraints are of the same nature as capacity constraints (26), but have a right-hand side value equal to 0 when $x_h = 0$, and equal to ϵ when $x_h = 1$.

As stated, conditions (31) can be very weak, in that the right-hand side value is very close to zero. However, they become effective in their strengthened form:

$$\sum_{i \in Q} x_i \geq x_h, \tag{32}$$

where $Q := \{i \in \{1, \ldots, n\} : (\alpha_i + \alpha'_i) UB_i + (\beta_i + \beta'_i) LB_i > 0\}$, and (α, β, γ) and $(\alpha', \beta', \gamma')$ are as before. We call (32) the *bridgeless inequalities*, as in the 2-dimensional case they forbid the presence of "bridges" in a certain network structure associated with the suppressed cells; see Fischetti and Salazar (1998) for details.

The separation problem for bridgeless inequalities is perfectly analogous to the one described for the strengthened capacity constraints (sliding case), and requires the solution of the two attacker subproblems associated with any non-sensitive cell h with $x^*_h > 0$.

4.3 Constraint management

A crucial ingredient of our branch-and-cut scheme is the run-time generation of constraints to be added to the master LP at every node of the enumeration tree. Although we generate just a few (violated) constraints for each family at each call of the separation procedures, the number of constraints produced during the overall branch-and-cut scheme can become rather large. On the other hand, not all the generated constraints are active at the same time, hence one can think of a dynamic mechanism to drop slack constraints from the master LP at a certain iteration, and possibly to reinsert them at a later time. This is achieved through a data-structure called the *constraint pool* which stores all the constraints that have been generated by the separation procedures. After the solution of each master LP, a clean-up procedure removes from the LP (but not from the pool) all the constraints that are slack at the optimal solution x^*. However, the removed constraints can become active (i.e., violated by the LP master solution x^*) at a later time, possibly when processing a different node of the branching tree. Therefore, before applying the separation phase we check whether the pool contains already violated constraints to be added to the current master LP. Only when no violated constraints are found in the pool, the separation procedures are applied to hopefully find new violated constraints to insert both in the pool and in the master LP.

In this way one can effectively manage a very large number of constraints, while keeping the current LP to be solved at a reasonable size.

4.4 Problem reduction

The computing time spent for the solution of a given instance depends on its size, expressed in terms of both the number of decision variables involved and the number of nonzero protection levels (recall that zero protection levels do not induce capacity constraints). We next outline simple criteria to reduce the size of a given CSP instance.

A typically substantial reduction on the number of nonzero protection levels is achieved in a *preprocessing* phase, to be applied before entering the branch-and-cut scheme. Preprocessing is based on the observation that primary suppressions alone may be sufficient to protect some of the sensitive cells. To be specific, we consider the suppression pattern $SUP := \{i_1, \ldots, i_p\}$ and for each sensitive cell i_k we solve the attacker subproblem (13)–(16). In case $\overline{y}_{i_k} \geq a_{i_k} + UPL_k$ one can clearly set $UPL_k = 0$, thus deactivating the upper protection level requirement for i_k. Otherwise, the capacity constraint associated with the dual optimal solution of the attacker subproblem qualifies as a relevant constraint, hence it is stored (in its strengthened form) in the constraint pool data structure. A similar reasoning applies to lower and sliding protection levels.

A naive implementation of the above idea may be excessively time consuming for large instances, in that it may require even more computing time than the whole branch-and-cut algorithm applied on the original instance. Hence a parametric resolution of the several attacker subproblems involved in preprocessing is needed. We suggest the following approach.

We introduce two p-dimensional arrays HIGH and LOW, whose entries HIGH_k and LOW_k give, respectively, lower and upper bounds on the solution value $\overline{y}_{i_k}$ and $\underline{y}_{i_k}$ of the attacker subproblems associated with i_k, with respect to the suppression pattern $SUP = \{i_1, \ldots, i_p\}$. We initialize $\mathrm{HIGH}_k{:=}\mathrm{LOW}_k := a_{i_k}$ for all $k = 1, \ldots, p$, and then consider the sensitive cells i_k according to a nonincreasing sequence of the associated values $\max\{SPL_k, UPL_k + LPL_k\}$. For each i_k, we first check whether $\mathrm{HIGH}_k < a_{i_k} + UPL_k$ or $\mathrm{HIGH}_k - \mathrm{LOW}_k < SPL_k$, in which case we solve the attacker subproblem (13)–(16) and obtain a consistent table $\overline{y}$ maximizing y_{i_k}. The entries of $\overline{y}$ are then used to update all the entries of HIGH and LOW, by setting $\mathrm{HIGH}_h = \max\{\mathrm{HIGH}_h, \overline{y}_{i_h}\}$ and $\mathrm{LOW}_h = \min\{\mathrm{LOW}_h, \overline{y}_{i_h}\}$ for all $h = 1, \ldots, p$. We then check whether $\mathrm{LOW}_k > a_{i_k} - LPL_k$ and $\mathrm{HIGH}_k - \mathrm{LOW}_k < SPL_k$, in which case we solve the attacker subproblem

(20) and obtain a consistent table $\underline{y}$ minimizing y_{i_k}. As before, the entries of $\underline{y}$ are used to update all the entries of HIGH and LOW. Finally, we use the updated values HIGH_k and LOW_k to set $UPL_k = 0$ (if $\mathrm{HIGH}_k \geq a_{i_k} + UPL_k$), $LPL_k = 0$ (if $\mathrm{LOW}_k \leq a_{i_k} - LPL_k$), and $SPL_k = 0$ (if $\mathrm{HIGH}_k - \mathrm{LOW}_k \geq SPL_k$). In this way we avoid a typically substantial number of attacker subproblem resolutions.

Whenever a protection level associated to i_k is not satisfied, we have at hand a capacity constraint associated with the dual optimal solution of the corresponding attacker subproblem, which can be used to initialize the constraint pool. In this way, with no extra computing time we perform both the preprocessing phase and initialize the constraint pool with a number of relevant constraints.

We now address the reduction of variables in the LP programs to be solved within our branch-and-cut algorithm. Our approach is to fix to 0 or 1 some decision variables during the processing of each node of the branch-decision tree. We use the classical criteria based on LP reduced cost, as described, e.g., in Nemhauser and Wolsey (1988). According to our computational experience, these criteria allow one to fix a large percentage of the variables very early during the computation. Moreover, we have implemented a *variable-pricing* technique to speed-up the overall computation and to drastically reduce memory requirement when instances with more than 10,000 variables are considered; see, e.g., Nemhauser and Wolsey (1988) for details.

4.5 Heuristic

The convergence of the overall branch-and-bound scheme can be speeded up if a near-optimal CSP solution is found very early during computation. Therefore one is interested in an efficient heuristic algorithm, to be applied (possibly several times) at each node of the branch-decision tree.

The availability of a good heuristic solution is also important when the convergence of the branch-and-cut scheme requires large computing time, and one has to stop the algorithm before converge.

We have implemented a heuristic procedure in the spirit of the one proposed by Kelly, Golden and Assad (1992) and Robertson (1994). Our procedure works in stages, in each of which one finds heuristically a set of new suppressions needed to guarantee the required protection levels for a certain sensitive cell i_k. To be more specific, we start by defining the current set of suppressions $SUP := \{i_1, \ldots, i_p\}$, and define $c_i := 0$ for all $i \in SUP$ and $c_i := w_i$ for all $i \notin SUP$. Then we consider all the sensitive cells i_k according to some heuristically-defined sequence.

For each such i_k, in turn, we first consider the following *incremental attacker sub-*

problem associated with the upper protection level UPL_k (if different from 0):

$$\min \sum_{i=1}^{n} c_i(y_i^+ + y_i^-) \tag{33}$$

subject to

$$M(y^+ - y^-) = 0 \tag{34}$$
$$0 \leq y_i^+ \leq UB_i \qquad \text{for all } i = 1, \ldots, n \tag{35}$$
$$0 \leq y_i^- \leq LB_i \qquad \text{for all } i = 1, \ldots, n \tag{36}$$
$$y_{i_k}^- = 0 \text{ and } y_{i_k}^+ = UPL_k. \tag{37}$$

Variables y_i^+ and y_i^- correspond to possible increments or decrements of value a_i in a consistent table $y := a + y^+ - y^-$. Constraints (34)–(36) then stipulate consistency of table y, whereas (37) imposes $y_{i_k} = a_{i_k} + UPL_k$. Objective function (33) gives an estimate of the additional weight associated with the suppression of the entries $i \notin SUP$ with $y_i \neq a_i$ (i.e., with $y_i^+ + y_i^- > 0$).

We solve problem (33)–(37) and insert in SUP all the cells $i \notin SUP$ having $y_i^+ + y_i^- > 0$ in the optimal solution. This guarantees the fulfillment of the upper protection level requirement for i_k with respect to the new set SUP of suppressions. Afterwards, we set $c_i = 0$ for all $i \in SUP$, and apply a similar technique to extend SUP so as to guarantee the fulfillment of the lower and sliding protection levels. This requires to solve model (33)–(37) two additional times, a first time with (37) replaced by

$$y_{i_k}^+ = 0 \text{ and } y_{i_k}^- = LPL_k$$

and a second time with (37) replaced by

$$y_{i_k}^+ + y_{i_k}^- = SPL_k.$$

As in the preprocessing problem reduction, a parametric resolution of the incremental attacker subproblems typically reduces considerably the computational effort spent within the heuristic.

In some cases, the above heuristic can introduce redundant suppressions. Hence it may be worth applying a clean-up procedure to detect and remove such redundancies; see, e.g., Kelly, Golden and Assad (1992). To this end, let SUP denote the feasible suppression pattern found by the heuristic. The clean-up procedure considers, in sequence, all the complementary suppressions $h \in SUP \setminus \{i_1, \ldots, i_p\}$, according to decreasing weights w_h, and checks whether $SUP \setminus \{h\}$ is a feasible suppression pattern as well, in which case SUP is replaced by $SUP \setminus \{h\}$.

Clean up can be very time consuming as it requires the solution, for each $h \in SUP \setminus \{i_1, \ldots, i_p\}$, of $2p$ attacker subproblems associated with the sensitive cells. A considerable speed-up is obtained by using the "dual information" associated with the capacity constraints stored in the current pool. Indeed, at each iteration of the clean-up procedure let x^* be defined as $x_i^* = 1$ if $i \in SUP \setminus \{h\}$ and $x_i^* = 0$ otherwise. Our order of business is to check whether x^* does not define a feasible suppression pattern. Clearly, a sufficient condition for pattern infeasibility is that x^* violates any constraint in the pool. Therefore, before solving the time-consuming attacker subproblems one can very quickly scan and check for violation the constraints stored in the pool: in case a violated constraint is found, the computation can be stopped immediately as we have a proof of the fact that $SUP \setminus \{h\}$ is not a feasible pattern, and we can proceed with the next candidate suppression h. Otherwise, we need to check $SUP \setminus \{h\}$ for feasibility by solving parametrically a sequence of attacker subproblems, as discussed in the problem reduction subsection.

We apply the above heuristic at the very beginning of our branch-and-cut code, right after the preprocessing phase for reducing the number of nonzero protection levels and the constraint pool initialization. In addition, we have implemented a modified version of the heuristic which exploits the information associated with the fractional optimal solutions x^* of the master LP problems solved during the branch-and-cut execution. In this version, the cell costs in (33) are defined as $c_i := (1 - x^*)w_i$ so as to encourage the suppression of cells i with x_i^* close to 1, which are likely to be part of the optimal CSP solution. The modified heuristic is applied right after the processing of each branch-decision node.

5 Computational Results

The algorithm described in the previous section was implemented in ANSI C. We e-valuated the performance of the code on a set of real-world statistical tables. The software was compiled with Watcom C/C++ 10.6 and run, under Windows 95, on a PC Pentium II/266 with 32 RAM Mbytes. As to the LP-solver, we used the commercial package CPLEX 3.0. Our test bed consists of ten real-world instances provides by people from different national statistical offices. It includes three 2-dimensional tables, two 3-dimensional tables, one 4-dimensional table, and four linked tables. The first linked table (USDE1) corresponds to a 2-section of a 6-dimensional $6 \times 4 \times 16 \times 4 \times 4 \times 4$ table, whereas the second linked table (USDE2) corresponds to a 4-section of a 9-dimensional $4 \times 29 \times 3 \times 4 \times 5 \times 6 \times 5 \times 4 \times 5$ table; for both instances $UPL_i = LPL_i$ holds for each cell i . The third linked table (USDE1a) is identical to USDE1, but we set $UPL_i = 2\,LPL_i$ for each cell i. The fourth linked table (USDE1b) was obtained from USDE1a by dividing by 1,000 and rounding up to the nearest integer all cell weights w_i.

For all instances in our test bed, the external bounds are $lb_i = 0$ and $ub_i = +\infty$ for

Table 1: Statistics on real-world instances.

name	type	cells	links	p	pl	pl_0	t_0	HEU_1	t_1
CBS1	41×31	1271	72	3	6	6	0.3	73.79	0.4
CBS2	183×61	11163	244	2467	4934	2	6.1	0.00	0.2
CCSR	359×46	16514	405	4923	9846	54	36.3	0.00	24.6
CBS3	$6 \times 8 \times 13$	624	230	17	34	26	0.1	5.91	0.3
CBS4	$6 \times 33 \times 8$	1584	510	146	292	201	1.3	1.68	7.5
CBS5	$6 \times 8 \times 8 \times 13$	4992	2464	517	1034	947	119.9	90.38	1533.8
USDE1	linked	1254	1148	165	330	320	0.5	30.21	16.8
USDE2	linked	1141	1000	310	620	572	8.9	33.18	29.8
USDE1a	linked	1254	1148	165	330	322	0.8	26.43	17.1
USDE1b	linked	1254	1148	165	330	322	0.8	27.31	17.3

all $i = 1, \ldots, n$, whereas the sliding protection levels SPL_k are zero for all sensitive cells.

Table 1 reports information about the test bed and the performance of our initial heuristic when applied before entering branch-and-cut computation. For each instance, the table gives:

name: name of the instance;

type: size (for k-dimensional tables) or structure of the table;

cells: number of cells in the table (=n);

links: number of equations in the table (=number of rows of matrix M);

p: number of sensitive cells (primary suppressions);

pl: number of nonzero protection levels before problem reduction;

pl_0: number of nonzero protection levels after problem reduction;

t_0: Pentium II/266 wall clock seconds spent in the preprocessing phase for reducing the number of non-zero protection levels;

HEU_1: percentage ratio $100 \times$ (*HEU'* $-$ optimal solution value)/ (optimal solution value), where *HEU'* is the upper bound value computed by our initial heuristic before entering branch-and-cut computation;

t_1: Pentium II/266 wall clock seconds spent by our initial heuristic.

A comparison of columns *pl* and pl_0 shows that pl_0 is often significantly smaller than *pl*, meaning that our preprocessing procedure was effective in detecting redundant

Table 2: Statistics on branch-and-cut executions.

name	r-HEU	r-LB	r-time	optimum	sup	node	iter	time
CBS1	29.13	2.91	5.1	103	5	5	75	9.4
CBS2	0.00	0.00	6.4	403	2	1	1	6.4
CCSR	0.00	0.00	61.3	256	27	1	1	61.3
CBS3	1.88	1.00	8.4	22590362	27	32	416	66.0
CBS4	1.25	0.20	40.9	186433	51	19	70	123.7
CBS5	0.00	0.00	4924.1	6312	261	1	76	4924.1
USDE1	3.42	0.55	626.6	2228523	254	22	202	1187.0
USDE2	1.38	2.68	702.0	4643198	181	46	238	2397.2
USDE1a	1.43	1.92	689.1	2325788	273	97	473	2614.6
USDE1b	1.25	1.21	670.6	2157	274	16	240	1311.5

protection levels. This is particularly true in case of 2-dimensional tables, whose simple structure often leads to large patterns of “self-protected” sensitive cells.

The quality of our initial heuristic solution appears rather poor when compared with the optimal solution, in that columns HEU_1 exhibits significant percentage errors. In our opinion, however, the performance of our initial heuristic is at least comparable (and often significantly better) than that of the suppression procedures commonly used by practitioners. In other words, we believe that commonly-used suppression methodologies are likely to produce suppression patterns with excessively high information loss. This behaviour was probably underestimated in the past since no technique was available to solve complex instances to proven optimality, nor to compute reliable lower bounds on the optimal solution value. The capability of benchmarking known heuristics is therefore another important feature of our exact solution methodology.

Table 2 reports the following information on the overall branch-and-cut algorithm:

r-HEU: percentage ratio $100 \times$ (*r-HEU'* $-$ optimal solution value)/ (optimal solution value), where *r-HEU'* is the upper bound value computed by our heuristic at the end of the root node of the branch-decision tree;

r-LB: percentage ratio $100 \times$ (optimal solution value $-$ *r-LB'*)/ (optimal solution value), where *r-LB'* is the lower bound value available at the end of the root node of the branch-decision tree;

r-time: Pentium II/266 wall clock seconds spent at the root node, including the preprocessing time t_0 and the heuristic time t_1 reported in table 1;

optimum: optimal solution value;

sup: number of complementary (non-sensitive) suppressions in the optimal solution found; note that this number is not necessarily minimized, i.e., it is possible that other solutions require a larger information loss but fewer suppressions;

node: number of elaborated nodes in the decision tree;

iter: overall number of cutting-plane iterations;

time: Pentium II/266 wall clock seconds for the overall branch-and-cut algorithm.

As shown in Table 2, our branch-and-cut code was able to solve all the instances of our test bed within acceptable computing time, even on a slow personal computer with limited amount of RAM memory.

The 2-dimensional instances were solved easily by our code. This confirms the findings reported in Fischetti and Salazar (1998). The 3-dimensional instances were also solved within short computing time.

The 4-dimensional instance appears much more difficult to solve, instead. This is of course due to the large number of table links (equations) to be considered. In addition, the number of nonzero protection levels after preprocessing (as reported in column pl_0) is significantly larger than for the other instances. This results into a large number of time-consuming attacker subproblems that need to be solved for capacity cut separation, and into a large number of capacity cut constraints to be inserted explicitly into the master LP. Moreover, we have observed that the optimal solutions of the master LP's tend to have more fractional components with respect to to the case of 2-dimensional tables with about the same number of cells. In other words, increasing the table dimension seems to have much larger impact on the number of fractionalities than just increasing the size of a table. As a consequence, 4-dimensional tables tend to require a larger number of branchings to enforce the integrality of the variables. In addition, the heuristic approaches become much more time consuming as they work explicitly with all the nonzero variables of the current fractional LP solution.

As to linked tables, their exact solution can be obtained within reasonable computing time. As expected, instance USDE1a requires significantly more computing time than instance USDE1, due to the larger upper protection levels imposed, whereas an optimal solution of instance USDE1b can be found more easily due to the reduced weight range.

A comparison of columns HEU_1 and $r\text{-}HEU$ shows the effectiveness of our heuristic when driven by the LP solution available at the end of the root node. In particular, stopping branch-and-cut execution right after the root node would produce a heuristic procedure comparing very favorably with the initial heuristic, while also returning a reliable optimistic estimate (lower bound) on the optimal solution value.

Table 3: Statistics on the generated cuts.

name	cap_0	cap	bri	cov	pool	LProws
CBS1	10	70	184	92	109	168
CBS2	2	0	0	0	2	2
CCSR	27	0	0	0	27	27
CBS3	25	226	504	523	3744	255
CBS4	125	90	52	69	153	166
CBS5	639	1500	0	0	418	502
USDE1	217	978	781	86	1760	937
USDE2	301	965	364	96	1311	535
USDE1a	226	1291	923	196	3569	923
USDE1b	226	1371	849	137	2256	993

Column r-*LB* shows that very tight lower bounds on the optimal solution value are available already at the root node of the branch-decision tree. Quite surprisingly, this is mainly due to the LP relaxation tightening introduced in Section 4.2, and in particular to the simple capacity constraint strengthening described in Section 4.2.1. Indeed, deactivating the model improvements introduced in Section 4.2 results into a dramatic lower bound deterioration.

Table 3 gives the following statistics on the generated cuts by the branch-and-cut scheme:

$\mathbf{cap_0}$: number of constraints stored in the constraint pool during the preprocessing and the initial heuristic procedures;

cap: overall number of capacity constraints generated;

bri: overall number of bridgeless inequalities generated;

cov: overall number of cover inequalities generated;

pool: overall number of constraints recovered from the constraint pool;

LProws: maximum number of rows in the master LP.

According to the table, the number of capacity constraints that need to be generated explicitly is rather small (recall that, in principle, the family of capacity constraints contains an exponential number of members). Moreover, the pool add/drop mechanism allows us to keep the master LP's to a manageable size; see column *LProws* of the table. Finally, we observe that a significant number of bridgeless and cover inequalities are generated during the branch-and-cut execution to reinforce the quality of the LP relaxation of the several master problems to be solved.

6 Conclusions

Cell suppression is a widely-used methodology in Statistical Disclosure Control. In this paper we have introduced a new integer linear programming model for the cell suppression problem, in the very general context of tables whose entries are subject to a generic system of linear constraints. Our model then covers k-dimensional tables with marginals as well as hierarchical and linked tables. To our knowledge, this is the first attempt to model and solve the cell suppression problem in such a very general context. We have also outlined a possible solution procedure in the branch-and-cut framework. Computational results on real-world instances have been reported. In particular, we were able to solve to proven optimality, for the first time, real-world 4-dimensional tables with marginals as well as linked tables.

We are convinced that the availability of a common test-bed for comparing different solution procedures can be of great help to researchers in the field. Possibly in cooperation with EUROSTAT, we are planning to set-up and maintain on the Web CSPLIB, a test-problem library containing real-world (no longer confidential) and synthetic data available in the public domain. Possible contributors can contact the second author for more information; see the Appendix for the data format we are planning to use.

Appendix: Input Data Format for CSPLIB Instances

CSPLIB instances are described by ASCII files. Two input formats are possible, one for k-dimensional tables with marginals and $1 \leq k \leq 4$, and another for more general tables with entries linked by linear equations. The second format allows one to also describe k-dimensional tables with marginals, for any k. For both formats, an example is given which refers to the following 2-dimensional table with marginals (sensitive cells are in boldface):

1	**3**	**5**	7	16
10	12	**11**	9	42
8	**6**	4	**2**	20
19	**21**	20	**18**	78

The input format for a k-dimensional table ($1 \leq k \leq 4$) starts with the INTEGER k, followed by k INTEGER numbers giving the table size for each dimension. A list of lines (records) follows, each associated with a different table cell and containing the following $k + 8$ items:

- the first k items are INTEGER numbers identifying the cell indices i (0 for marginals);

- a REAL number representing the nominal value a_i of the cell;
- an INTEGER number representing the loss of information w_i incurred if the cell is suppressed;
- a CHARACTER representing the status of the cell ('u' for a sensitive cell; 'z' for enforcing cell publication; 's' for a potential complementary suppression);
- a REAL number representing the lower bound lb_i on the cell value known to the attacker;
- a REAL number representing the upper bound ub_i on the cell value known to the attacker;
- a REAL number representing the lower protection level LPL_i of the cell required by the statistical office;
- a REAL number representing the upper protection level UPL_i of the cell required by the statistical office;
- a REAL number representing the sliding protection level SPL_i of the cell required by the statistical office.

Here is the input file for the sample instance.

```
============================================= top-of-file
2
3 4
0 0 78.0  4 s 0.0 100.0  0.0  0.0  0.0
0 1 19.0  2 s 0.0 100.0  0.0  0.0  0.0
0 2 21.0  0 u 0.0 100.0  2.0  4.0  0.0
0 3 20.0  2 s 0.0 100.0  0.0  0.0  0.0
0 4 18.0  0 u 0.0 100.0  1.0  3.0  0.0
1 0 16.0  2 s 0.0 100.0  0.0  0.0  0.0
1 1  1.0  0 u 0.0 100.0  0.0  4.0  0.0
1 2  3.0  0 u 0.0 100.0  1.0  3.0  0.0
1 3  5.0  0 u 0.0 100.0  0.0  1.0  0.0
1 4  7.0  1 s 0.0 100.0  0.0  0.0  0.0
2 0 42.0  2 s 0.0 100.0  0.0  0.0  0.0
2 1 10.0  0 u 0.0 100.0  1.0  4.0  0.0
2 2 12.0  1 s 0.0 100.0  0.0  0.0  0.0
2 3 11.0  0 u 0.0 100.0  2.0  3.0  0.0
```

```
2 4  9.0  1 s 0.0 100.0  0.0  0.0  0.0
3 0 20.0  2 s 0.0 100.0  0.0  0.0  0.0
3 1  8.0  1 s 0.0 100.0  0.0  0.0  0.0
3 2  6.0  0 u 0.0 100.0  2.0  0.0  0.0
3 3  4.0  1 s 0.0 100.0  0.0  0.0  0.0
3 4  2.0  0 u 0.0 100.0  0.0  4.0  0.0
============================================ end-of-file
```

The input format for a generic table is organized in two sections containing the cell description and the equation description, respectively.

In order to discriminate the generic table format from the previous format, the first item of the file must be '0' (i.e., INTEGER zero).

The cell description section starts with the INTEGER number n of cells, followed by n lines containing 9 items each:

- an INTEGER number from 0 to $n-1$ with the cell index i;
- a REAL number representing the nominal value a_i of the cell;
- an INTEGER number representing the loss of information w_i incurred if the cell is suppressed;
- a CHARACTER representing the status of the cell ('u' for a sensitive cell; 'z' for enforcing cell publication; 's' for a potential complementary suppression);
- a REAL number representing the lower bound lb_i on the cell value known to the attacker;
- a REAL number representing the upper bound ub_i on the cell value known to the attacker;
- a REAL number representing the lower protection level LPL_i of the cell required by the statistical office;
- a REAL number representing the upper protection level UPL_i of the cell required by the statistical office;
- a REAL number representing the sliding protection level SPL_i of the cell required by the statistical office.

The equation description section starts with the INTEGER number m of equations, followed by m lines. Each line starts with a REAL number giving the right-hand side value of the correspondent equation, followed by an INTEGER number k giving the number of its nonzero left-hand side coefficients. The line continues with the separator ':', followed by k pairs of the form i (c_i), where the INTEGER number i gives the cell index and the INTEGER number c_i gives the corresponding nonzero left-hand side coefficient in the equation.

Here is the input file for the sample instance.

```
============================================= top-of-file
0
20
 0 78.0  4 s 0.0 100.0  0.0  0.0  0.0
 1 19.0  2 s 0.0 100.0  0.0  0.0  0.0
 2 21.0  0 u 0.0 100.0  2.0  4.0  0.0
 3 20.0  2 s 0.0 100.0  0.0  0.0  0.0
 4 18.0  0 u 0.0 100.0  1.0  3.0  0.0
 5 16.0  2 s 0.0 100.0  0.0  0.0  0.0
 6  1.0  0 u 0.0 100.0  0.0  4.0  0.0
 7  3.0  0 u 0.0 100.0  1.0  3.0  0.0
 8  5.0  0 u 0.0 100.0  0.0  1.0  0.0
 9  7.0  1 s 0.0 100.0  0.0  0.0  0.0
10 42.0  2 s 0.0 100.0  0.0  0.0  0.0
11 10.0  0 u 0.0 100.0  1.0  4.0  0.0
12 12.0  1 s 0.0 100.0  0.0  0.0  0.0
13 11.0  0 u 0.0 100.0  2.0  3.0  0.0
14  9.0  1 s 0.0 100.0  0.0  0.0  0.0
15 20.0  2 s 0.0 100.0  0.0  0.0  0.0
16  8.0  1 s 0.0 100.0  0.0  0.0  0.0
17  6.0  0 u 0.0 100.0  2.0  0.0  0.0
18  4.0  1 s 0.0 100.0  0.0  0.0  0.0
19  2.0  0 u 0.0 100.0  0.0  4.0  0.0
9
0.0  5 : 0 (-1) 1 (1) 2 (1) 3 (1) 4 (1)
0.0  5 : 5 (-1) 6 (1) 7 (1) 8 (1) 9 (1)
0.0  5 : 10 (-1) 11 (1) 12 (1) 13 (1) 14 (1)
0.0  5 : 15 (-1) 16 (1) 17 (1) 18 (1) 19 (1)
0.0  4 : 0 (-1) 5 (1) 10 (1) 15 (1)
0.0  4 : 1 (-1) 6 (1) 11 (1) 16 (1)
0.0  4 : 2 (-1) 7 (1) 12 (1) 17 (1)
0.0  4 : 3 (-1) 8 (1) 13 (1) 18 (1)
0.0  4 : 4 (-1) 9 (1) 14 (1) 19 (1)
```

== end-of-file

References

[1] Applegate, D., Bixby, R., Cook, W., and Chvátal, V. (1998), "CRPC Researchers Solve Traveling Salesman Problem for Record-Breaking 13,509 Cities," Internal Report, Center for Research on Parallel Computation, Rice University.

[2] Caprara, A., and Fischetti, M. (1997), "Branch-and-Cut Algorithms," in *Annotated Bibliographies in Combinatorial Optimization*, edited by Dell'Amico, M., Maffioli, F., and Martello, S., John Wiley & Sons.

[3] Cox, L. H. (1980), "Suppression Methodology and Statistical Disclosure Control," *Journal of the American Statistical Association*, 75, 377–385.

[4] Cox, L. H. (1995), "Network Models for Complementary Cell Suppression," *Journal of the American Statistical Association*, 90, 1453–1462.

[5] Crowder, H. P., Johnson, E. L., and Padberg, M. W. (1983), "Solving Large-Scale Zero-One Linear Programming Problems," *Operations Research*, 31, 803–834.

[6] Carvalho, F. D., Dellaert, N. P., and Osório, M. S. (1994) "Statistical Disclosure in Two-Dimensional Tables: General Tables," *Journal of the American Statistical Association*, 89, 1547–1557.

[7] Dellaert, N. P., and Luijten, W. A. (1996), "Statistical Disclosure in General Three-Dimensional Tables," Technical Paper TI 96-114/9, Tinbergen Institute.

[8] Fischetti, M., and Salazar, J. J. (1998), "Models and Algorithms for the 2-Dimensional Cell Suppression Problem in Statistical Disclosure Control," *Mathematical Programming.*

[9] Geurts, J. (1992), "Heuristics for Cell Suppression in Tables," working paper, Netherlands Central Bureau of Statistics, Voorburg.

[10] Gusfield, D. (1988), "A Graph Theoretic Approach to Statistical Data Security," *SIAM Journal on Computing* 17, 552–571.

[11] Kao, M. Y. (1996), "Data Security Equals Graph Connectivity," *SIAM Journal on Discrete Mathematics*, 9, 87–100.

[12] Kelly, J. P. (1990), "Confidentiality Protection in Two and Three-Dimensional Tables," Ph.D. dissertation, University of Maryland, College Park, Maryland.

[13] Kelly, J. P., Golden, B. L., and Assad, A. A. (1992), “Cell Suppression: Disclosure Protection for Sensitive Tabular Data,” *Networks* 22, 397–417.

[14] Nemhauser, G. L., and Wolsey, L. A. (1988), “Integer and Combinatorial Optimization,” John Wiley & Sons.

[15] Padberg, M., and Rinaldi, G. (1991), “A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems,” *SIAM Reviews*, 33, 60–100.

[16] Robertson, D. A. (1994), “Cell Suppression at Statistics Canada,” Proceedings of the *Second International Conference on Statistical Confidentiality*, Luxembourg.

[17] Sande, G. (1984), “Automated cell suppression to preserve confidentiality of business statistics,” *Statistical Journal of the United Nations ECE* 2, 33–41.

[18] Willenborg, L. C. R. J., and Waal, T. de (1996) “Statistical Disclosure Control in Practice,” *Lecture Notes in Statistics 111*, *Springer*, New York.