Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 1613–1623
computers & operations research
www.elsevier.com/locate/cor

# Cell suppression problem: A genetic-based approach

M.T. Almeida[a, b, *], G. Schütz[c], F.D. Carvalho[a, b]

[a] *Dep. Matemática, Instituto Superior de Economia e Gestão, Universidade Técnica de Lisboa, Rua do Quelhas, 6, 1200-781 Lisboa, Portugal*
[b] *CIO—Centro de Investigação Operacional-FC/UL, Portugal*
[c] *Escola Superior de Tecnologia, Universidade do Algarve, Campus da Penha, Estrada da Penha, 8005-139 Faro, Portugal*

Available online 18 October 2006

**Abstract**

Cell suppression is one of the most frequently used techniques to prevent the disclosure of sensitive data in statistical tables. Finding the minimum cost set of nonsensitive entries to suppress, along with the sensitive ones, in order to make a table safe for publication, is a *NP-hard* problem, denoted the cell suppression problem (CSP).

In this paper, we present *GenSup*, a new heuristic for the CSP, which combines the general features of genetic algorithms with safety conditions derived by several authors. The safety conditions are used to develop fast procedures to generate multiple initial solutions and also to recombine, to perturb and to repair solutions in order to improve their quality. The results obtained for 300 tables, with up to more than 90,000 entries, show that *GenSup* is very effective at finding low-cost sets of complementary suppressions to protect confidential data in two-dimensional tables.
© 2006 Elsevier Ltd. All rights reserved.

*Keywords:* Genetic algorithms; Heuristics; Networks; Cell suppression

## 1. Introduction

Statistical offices carry out census and surveys to collect personal and business data as the input for their reports. In order to get full cooperation from the respondents, the offices must guarantee that all confidential data entrusted to them is protected against disclosure. Over the last decade, work by researchers and practitioners has resulted in a multiplicity of methods for statistical disclosure limitation (see [1–6]).

Statistical data is often published as two-dimensional arrays, called statistical tables, resulting from the aggregation of individual data saved in microfiles. For tabular data, one of the preferred disclosure limitation techniques is cell suppression. It consists of omitting the figures that would allow users of the published tables to compute close estimates of the confidential data.

Fig. 1 shows a table with turnover data for companies, aggregated by geographical location and by activity, with confidential values substituted by the missing symbol *. By simply solving a system of linear equations, any user is able to deduce that the missing value in cell (Activity IV, Region B) is 10. If an uncertainty margin for each confidential cell of, for instance, at least ±15% was desired, one could substitute by the missing symbol the nonconfidential value in the single set of cells {(Activity IV, Region G)}, as in Fig. 2. A volume of nonconfidential data equal to 44 would

* Corresponding author. Dep. Matemática, Instituto Superior de Economia e Gestão, Universidade Técnica de Lisboa, Rua do Quelhas, 6, 1200-781 Lisboa, Portugal. Fax: +351 213922781.
*E-mail address:* talmeida@iseg.utl.pt (M.T. Almeida).

0305-0548/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.09.004

| | Reg.A | Reg.B | Reg.C | Reg.D | Reg.E | Reg.F | Reg.G | Total |
|---|---|---|---|---|---|---|---|---|
| Act. I | 20 | 40 | 35 | 31 | 72 | 26 | 10 | 234 |
| Act. II | 18 | * | 45 | 29 | 18 | 25 | * | 215 |
| Act. III | 34 | 27 | * | 50 | * | 34 | 38 | 223 |
| Act. IV | 22 | * | * | 43 | * | 31 | 44 | 180 |
| Act. V | 30 | * | 32 | 33 | 26 | 39 | * | 215 |
| Total | 124 | 137 | 142 | 186 | 156 | 155 | 167 | 1067 |

Fig. 1. Insufficiently protected table.

| | Reg. A | Reg. B | Reg. C | Reg. D | Reg. E | Reg. F | Reg. G | Total |
|---|---|---|---|---|---|---|---|---|
| Act. I | 20 | 40 | 35 | 31 | 72 | 26 | 10 | 234 |
| Act. II | 18 | * | 45 | 29 | 18 | 25 | * | 215 |
| Act. III | 34 | 27 | * | 50 | * | 34 | 38 | 223 |
| Act. IV | 22 | * | * | 43 | * | 31 | * | 180 |
| Act. V | 30 | * | 32 | 33 | 26 | 39 | * | 215 |
| Total | 124 | 137 | 142 | 186 | 156 | 155 | 167 | 1067 |

Fig. 2. Protected table.

be lost. If, instead, the omission was made in the set {(Activity III, Region B)}, then the loss of nonconfidential data would only be 27.

This example illustrates the core of the cell suppression problem (CSP): given a statistical table and a set $S_1$ of confidential cells, whose values shall not be disclosed, find an optimal set $S_2$, of nonconfidential cells, such that omitting the values of all cells in $S_1 \cup S_2$ renders the table safe for publication. A table is considered safe for publication if no intruder can compute estimates, for confidential values, deemed too close to the actual values and a set $S_2$ is optimal if it minimizes the loss of information. The CSP belongs to the class of *NP-hard* problems [7] for which no exact algorithm running in polynomial time exists, unless $P = NP$.

In this paper we present the *GenSup* algorithm, a genetic-based heuristic for the CSP, which combines the general features of genetic algorithms [8] with theoretical results derived by several authors for the CSP. The success of *GenSup* results from the use of safety conditions, to develop fast procedures to generate and to repair solutions, in a way that fits a natural representation of feasible solutions for the CSP. To the best of our knowledge this is the first attempt to tackle the CSP with a genetic methodology. The computational experience performed on two sets of test instances, randomly generated to reproduce real-world instances typical of sociological and of business data, showed that the performance of *GenSup* is more stable than that of other heuristics known from the literature, with low-cost solutions consistently produced for both test sets.

The paper is organized as follows. Section 2 presents the background and the notation. Section 3 is devoted to a brief review of constructive heuristics for the CSP. The *GenSup* algorithm is described in Section 4. Section 5 presents the lower-bounding and the clean-up procedures. Sections 6 and 7 contain the computational experience and the conclusions, respectively.

## 2. Background and notation

A two-dimensional statistical table $A = [a_{ij}]$ may be defined as an $(m + 1) \times (n + 1)$ array of nonnegative numbers. The values in the $(m + 1)$th row are the column totals and the values in the $(n + 1)$th column are the row totals. Row and column totals are also called marginal cells. The value $a_{m+1,n+1}$ is the grand total.

The set of confidential cells will be denoted by $S_1$ and its cardinality will be denoted by $p$. For each cell $(i, j) \in S_1$, a lower protection level, $l_{ij}$, with $0 \leqslant l_{ij} \leqslant a_{ij}$, and an upper protection level, $u_{ij}$, with $u_{ij} \geqslant 0$ and $l_{ij} + u_{ij} > 0$, are set a priori, to define the cell's protection interval $[a_{ij} - l_{ij}, a_{ij} + u_{ij}]$. A confidential cell $(i, j)$ is left-protected (respectively,

right-protected) if the tightest range an intruder is able to compute for $a_{ij}$ contains $a_{ij} - l_{ij}$ (respectively, $a_{ij} + u_{ij}$). If a cell is both left-protected and right-protected then it is protected. When convenient to simplify the notation, the set of confidential cells will also be denoted by $S_1 = \{(i_k, j_k) : 1 \leqslant k \leqslant p\}$ and protection ranges denoted by $[a_k - l_k, a_k + u_k]$, $1 \leqslant k \leqslant p$.

A set $S_2$, of complementary suppressions, is feasible if all confidential cells get protected when the values in $S_1 \cup S_2$ are omitted. Its cost is defined by $\xi(S_2) = \sum_{(i,j)\in S_2} a_{ij}$ and represents the total volume of nonconfidential data lost by the users.

A statistical table may be represented by a directed bipartite network $\mathcal{N} = (V, \mathcal{A})$, with capacities defined on its arcs [7,9]. The node set, $V = R \cup C$, is the union of a set $R$ of $m + 1$ nodes, representing the table rows, with a set $C$ of $n + 1$ nodes, representing the table columns. Each cell $(i, j)$ of the table is represented by two directed arcs, namely the forward arc $(i, j)$ from row node $i$ to column node $j$ and the reverse arc $(j, i)$ from column node $j$ to row node $i$. The capacities on the forward arcs are $c_{ij} = +\infty$ if $(i, j)$ is either an internal cell or the grand total cell and $c_{ij} = a_{ij}$ if $(i, j)$ is a marginal cell. The capacities on the reverse arcs are $c_{ji} = a_{ij}$ if $(i, j)$ is either an internal cell or the grand total cell and $c_{ji} = +\infty$ if $(i, j)$ is a marginal cell. Given a set $S_2$ of complementary suppressions, $\mathcal{N}_{S_1\cup S_2} = (V, \mathcal{A}_{S_1\cup S_2})$ is the subnetwork that represents only the suppressed cells. For every $k \in \{1, \ldots, p\}$, $\mathcal{N}^k$ and $\mathcal{N}^k_{S_1\cup S_2}$ are the networks that result from removing arcs $(i_k, j_k)$ and $(j_k, i_k)$ from $\mathcal{A}$ and from $\mathcal{A}_{S_1\cup S_2}$, respectively. Readers unfamiliar with network flow theory and algorithms are referred to [10].

In [7], Kelly et al. derived a network flow condition for a confidential cell to be protected, that can be stated for the whole table as follows: internal cells and the grand total, $(i_k, j_k) \in S_1$, are right-protected if and only if $u_k$ units of flow may be sent from column node $j_k$ to row node $i_k$ and are left-protected if and only if $l_k$ units of flow may be sent from row node $i_k$ to column node $j_k$, in $\mathcal{N}^k_{S_1\cup S_2}$; marginal cells, $(i_k, j_k) \in S_1$, are right-protected if and only if $u_k$ units of flow may be sent from row node $i_k$ to column node $j_k$ and are left-protected if and only if $l_k$ units of flow may be sent from column node $j_k$ to row node $i_k$, in $\mathcal{N}^k_{S_1\cup S_2}$.

In the last couple of decades, one of the most active research streams in combinatorial optimization has been that on heuristic algorithms that attempt to imitate natural processes to solve difficult problems. The algorithms that imitate the evolution of biological species are referred to as genetic algorithms [8]. In short, these algorithms act as follows: starting with a set of solutions (the initial population) generate new solutions (their offsprings) by combining some selected pairs of solutions (crossover) and by perturbing solutions (mutation) until a stopping criterion is verified. These basic features may be combined with constructive heuristics, local search and other traditional optimization tools to develop a wide variety of genetic-based methods [11,12].

## 3. Constructive heuristics

Some constructive heuristics for the CSP, in two-dimensional tables, are based on the network representation of the problem, reviewed above. Other heuristics are adaptations of methods developed for more complex tables.

### 3.1. Network-based heuristics

Based on the network flow condition, Kelly et al. [7] proposed a constructive heuristic for the CSP that protects a confidential cell at a time, by solving a pair of minimum cost flow problems. Later on, Carvalho et al. [13] suggested the substitution of the minimum cost flow problems by a sequence of shortest path computations, to get a more accurate representation of the suppression costs and to reduce the computing time. Several variants based on this general framework have been developed.

*Shortest path heuristic* [9,13,14]: In the shortest path (SP) heuristic the confidential cells are considered, one at a time, in any order. Different solutions may be obtained modifying the order, but experience indicates that there is no best ordering. A feasible solution is obtained including in $S_2$ all nonconfidential cells corresponding to the arcs in the paths generated as follows. For each cell $(i_k, j_k) \in S_1$, a sequence of minimum cost paths, joining nodes $i_k$ and $j_k$ in $\mathcal{N}^k$, with a total capacity at least equal to the cell's right-protection level is built. The procedure is then performed for the left-protection level. In the path computations, the arcs representing confidential cells are assigned zero cost and the arcs representing nonconfidential cells have their costs set to zero after being included in a path for the first time. Different solutions may be generated defining different initial costs for the nonconfidential cells. By default, these costs are equal to the cell values.

*Castro's heuristic* [15]: Castro's heuristic also relies on shortest path computations, to generate feasible solutions for the CSP, but implements them in a different way: every time a shortest path is computed, the protection levels of all confidential cells on it are checked, and those fulfilled are considered solved. To avoid residual capacity updating operations, when two or more shortest paths are required to fulfill a protection level, they are sequentially computed under the additional condition that they share no arc. The cost matrix, for the path computations, is defined following the stratification scheme proposed in [16], to balance the number of new suppressions and the number of shortest path problems to be solved.

In the computational experience reported in [15], the algorithm never failed to find feasible sequences of arc-disjoint paths. To guarantee its robustness, the algorithm includes an infeasibility recovery procedure to deal with failures. It substitutes the computation of shortest paths by the solution of minimum cost network flow problems. This substitution is very resource demanding, even for moderate size tables, due to the high density of the underlying networks and destroys the accuracy of the representation provided by the shortest path approach for the loss of information.

*Parallel bound and path heuristic*: Genetic algorithms require a set of solutions for their initial populations. To obtain a diversified set of feasible solutions, we designed the parallel bound and path (PBP) heuristic that is a two-phase scheme to generate in parallel a given number, $N$, of solutions. It uses the row and column lower bounds, derived in [14], for the volume of nonconfidential data that must be suppressed in each row or column with confidential cells. In phase 1, the confidential cells are considered one at a time, in decreasing order of their values, and $N$ complementary suppression sets are generated, as follows. For each confidential cell, the minimum number of nonconfidential cells in its row, in the sequence resulting from their ordering in increasing value order, necessary to reach the row lower bound are included in $S_2$, for the first individual. The procedure is repeated for the following $N-1$ individuals, beginning the ordering of the cells to be selected at the second position in the ordering for the previous individual. Then, the rationale is repeated for the columns, taking into account the complementary suppressions already assigned to each individual. After processing all confidential cells, phase 1 is concluded with a set of cell interchanges and substitutions aimed at eliminating single suppressions in rows and in columns with no confidential cells, if any. Phase 2 consists of applying the SP heuristic to each solution resulting from phase 1, with zero cost assigned to cells in $S_1 \cup S_2$, to certify that $S_2$ is feasible or to enlarge it in order to achieve feasibility.

### 3.2. Other heuristics

The hypercube method, as described in [17], is a fast heuristic developed to find sets of complementary suppressions to protect confidential data in large complex tables. It decomposes complex tables into subtables, without substructure, and iteratively protects the subtables. For two-dimensional tables, it amounts to finding, for each confidential cell, a minimum cost set of three cells to protect it, forming the corner points of a rectangle, i.e. it amounts to including each confidential cell in a minimum cost feasible circuit of cardinality equal to 4. Due to the cardinality constraint, it has a tendency of oversuppression. The method was designed to guarantee a sliding protection range for each confidential cell: a lower bound is calculated for the width of the cell's protection range, resulting from the suppression of the corner points of each hypercube, and that suppression is considered feasible if it turns out that the bound is sufficiently large. Suggestions on how to set the width of the protection ranges, to obtain a feasible set of suppressions, can be found in [17]. The adaptation of the hypercube method to the a priori fixed interval protection criterion has no theoretical difficulty. However, in practice, this adaptation yields a high rate of failure for tables with large protection levels.

The *HiTaS* method [18] was developed for tables with hierarchical structure. It is a top-down methodology to decompose a hierarchical table into subtables, and then solving the CSP for each subtable. In hierarchical tables, certain cells appear in more than one subtable (e.g. marginal cells in one subtable may be internal cells of another one) and it shall not be possible to use values in one subtable to recompute missing values in another one. So, whenever a particular value is suppressed in one of the subtables, it must be also suppressed in all the others, which may imply some backtracking steps. In [18], the *HiTaS* method is illustrated on a small two-dimensional table, in which both explanatory variables have a hierarchical structure, but no mention is made to the algorithm used to solve the CSP for each subtable.

## 4. *GenSup* algorithm

*GenSup* starts with an initial population generated by constructive heuristics. This population has a number of individuals bounded from above by a parameter $P$, with no two individuals with the same cost allowed. In each population,

reproduction is carried out performing crossover operations on selected individuals and mutations. The selection of the individuals for reproduction favors the smaller cost ones. By crossover, each pair selected for reproduction yields four new individuals. The population renewal is made selecting the best individuals in the current generation and in its offspring. New populations of sets of complementary suppressions are repeatedly generated until a stopping condition is verified.

Each individual is encoded as a vector, $\mathbf{v} = [v_1, \ldots, v_{|S_2|}]$, that stores the corresponding set of complementary suppressions, with each cell $(i, j) \in S_2$ stored as the integer $(i-1) \times (n+1) + j$. This is a quite concise encoding because the number of complementary suppressions is much smaller than $(m+1) \times (n+1)$.

The main components of *GenSup* are detailed below.

*Initial population*: For the initial population, two individuals are generated with the SP heuristic, which is very fast and, in general, produces low-cost solutions (see Section 6.4). One is generated with the default initial costs. For the other, we use the optimal solution of the lower-bounding model (see Section 5), setting to zero the initial costs of nonconfidential cells represented by positive variables. The remaining $P-2$ individuals are generated with the PBP heuristic. If two or more individuals sharing the same cost result, then only one of them, with the minimum cardinality for $S_2$, is accepted in the population.

*Selection*: For selection purposes, the quality of each individual is defined by its cost and each one of the 25% best individuals is considered an elite solution. The set of pairs to be subjected to crossover is selected matching each elite solution with one of the remaining 75% individuals, chosen at random.

*Crossover*: The crossover of each pair of selected individuals yields four offsprings: the first two result from the traditional 1-point crossover operator; the third offspring receives the suppressions shared by both parents; the last offspring receives the suppressions shared by both parents, as its sibling, and receives also the suppressions exclusive of the best parent, with probability of 0.75, and the suppressions exclusive of the other parent, with probability of 0.25. Every offspring is checked for feasibility and repaired, when necessary, as follows. The SP heuristic is run with zero cost assigned to its inherited suppressed cells. If the SP solution has zero cost, then the offspring is feasible. Otherwise, the repair consists of enlarging its set of suppressions with the cells associated with nonzero cost arcs in the SP solution. As no two individuals with the same cost are accepted, an offspring may end up discarded.

*Mutation*: The mutation operator considers one cell $(i, j) \in S_2$ at a time. For each $(i, j) \in S_2$, it consists of setting to zero the cost of all cells in $S_2 \backslash \{(i, j)\}$ and then running the SP heuristic. The mutation operator was applied to each new best individual found. It was also applied to the best offspring of each pair at iteration $k = \lfloor \alpha/2 \rfloor$, where $\alpha$ is the maximum number of iterations before halting the search.

*Population renewal*: The population management is aimed at keeping computing times within reasonable limits, avoiding premature convergence. After the reproduction operations over the current population are concluded, the next generation is obtained selecting the best individuals among all individuals in this population and all their offsprings, under two conditions: (i) no more than $P$ individuals are selected; (ii) no two selected individuals share the same cost. Condition (ii) guarantees that no solution is repeated in the population and determines the rejection of solutions that would, most probably, have a pattern very similar to one present in the population. In general, good solutions move along several generations. If less than $P$ individuals, in the pool of the current population and its offspring, verify the diversity condition, then less than $P$ individuals will constitute the following generation. Note that the number of individuals in a generation is never smaller than that number in the precedent one. As a consequence, as soon as the upper limit is achieved the population size remains stable up to the end of the procedure.

*Stopping criterion*: The algorithm stops after performing $\alpha$ iterations or after performing $\beta$ consecutive iterations without improving the best solution.

## 5. Lower bound and clean-up

To assess the quality of the solutions generated by the constructive heuristics and by *GenSup*, we used the lower bound on the CSP optimum given by the optimum of the *LP*-relaxation of the binary model presented in [14]. This lower bound is very fast to compute (see Section 6.3) and dominates, in the theoretical sense, the lower bounds in [7,15,19].

Protecting confidential cells sequentially often results in final solutions with redundant suppressions. To eliminate them, we developed an adaptation of the delete-and-check procedure [20]: for each cell $(i, j) \in S_2$, the SP heuristic

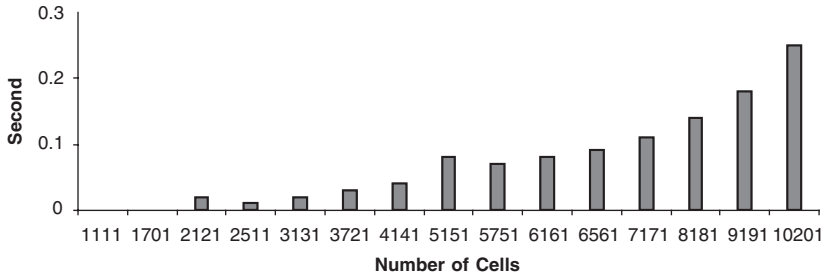


Fig. 3. Preprocessing time (class I).

is run on the network that results from removing from $\mathcal{N}_{S_1 \cup S_2}$ the arcs associated to $(i, j)$. If the result is a feasible solution for the CSP, then $(i, j)$ is redundant and may be removed from $S_2$. This clean-up procedure was applied to the solutions generated with SP to include in the initial population.

## 6. Computational experience

The computational study was carried out on a 3.00 GHz Pentium IV processor with 1.00 GB RAM. The codes for *GenSup* were written in C and compiled in Visual C.

### 6.1. Data

The computational experience was performed on two classes of typical statistical tables randomly generated following the rules used in [7,9]. For each class, we present the average results obtained on 15 sets of instances. Each set has 10 tables, sharing the same dimensions $m$ and $n$.

The 150 tables in class I have a number of internal cells ranging from 1,000 up to 10,000 and dimensions $m \times n$ up to $100 \times 100$. Every internal cell has a random integer value in [0, 499] and all cells with values in [1, 4] are confidential. The upper and lower protection levels are $u_{ij} = a_{ij}$ and $l_{ij} = a_{ij} - 1$. These generating rules were first used in [9], following the advice of a member of the ISTAT, the Italian statistical institute.

The 150 tables in class II have a number of internal cells ranging from 1, 000 up to 90, 000 and dimensions $m \times n$ up to $300 \times 300$. Every internal cell has a random integer value in [0, 1000] and the upper and the lower protection levels are both equal to 15% of the respective confidential cell's value rounded up to the nearest integer. Internal cells and marginal cells are confidential with probabilities of 0.2 and 0.1, respectively. These generating rules were proposed in [7] and later used also in [9].

For both classes, the cost assigned to each complementary suppression $(i, j)$ is its value $a_{ij}$, and no zero-valued cells are suppressed.

### 6.2. Problem preprocessing

The running times of the heuristics depend on the number of protection levels of a table that need to be covered. A confidential cell may be automatically right-protected (respectively, left-protected), i.e. a cell may not require complementary suppressions for its upper (respectively, lower) protection. This happens when, due to the pattern and the values of the other confidential cells, an intruder is unable to compute values that violate the right (respectively, the left) end of the cell's protection interval. To identify redundant protection levels, we run the SP heuristic, with infinity costs assigned to all cells not in $S_1$. To speed up the computations, the shortest path routine is substituted, in this case, by a maximum capacity path routine [10].

Figs. 3 and 4 present the preprocessing computing times for the class I and the class II instances, respectively.

For class I tables, the preprocessing procedure identified almost 80% of the protection levels as redundant, with average computing times below half a second in every set. For class II tables, the overall average running time was 13 s. Only for the last set of tables did the average running time exceed 1 min (with a value of 112 s in this case). Although

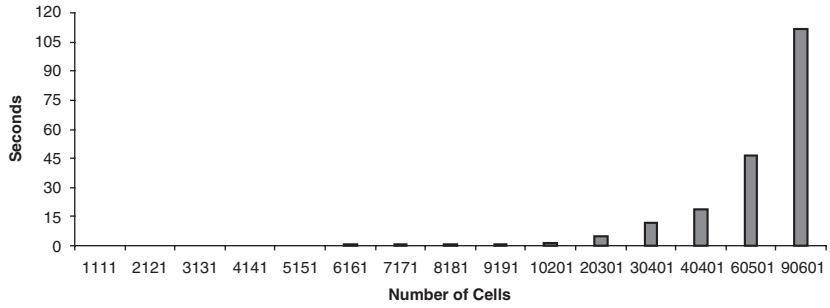


Fig. 4. Preprocessing time (class II).

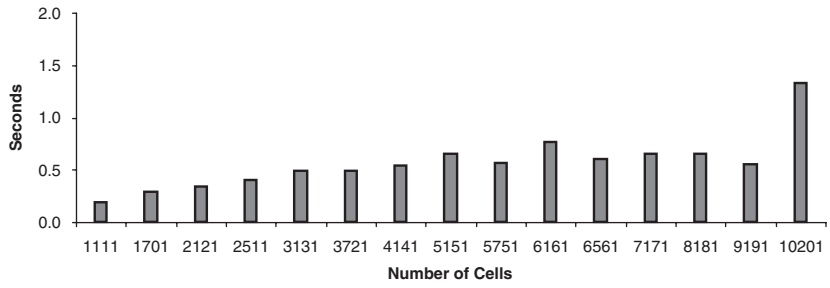


Fig. 5. *LB* computing time (class I).

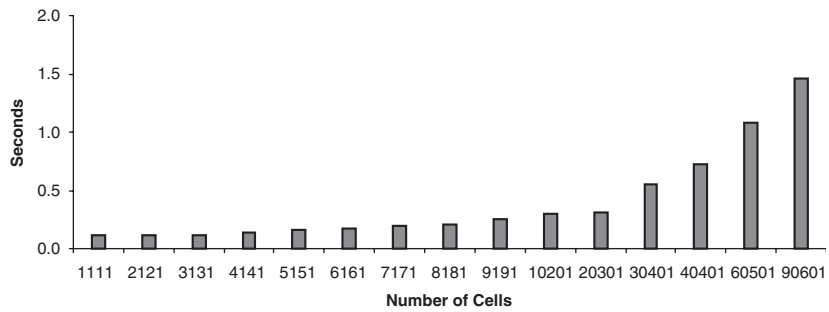


Fig. 6. *LB* computing time (class II).

comparatively larger than for the other class, these running times were justified by the identification of over 99% of the protection levels as redundant, for those instances. For tables generated with the same rules, a similar redundancy ratio is reported in [9].

### 6.3. Lower bound

Figs. 5 and 6 present, for the class I and the class II instances, respectively, the time to compute the lower bound value, *LB*, using the commercial solver Cplex 8.10 on a 866 MHz PC Pentium III processor with 128 Mbyte RAM.

The *LB* values were computed, on average, in less than 2 s in every set, for both classes. The percentage of integer *LP* solutions was much larger in the first class suggesting that, in practice, *LB* is a tighter bound for class I than for class II tables.

| Gap (%) | *SP* | Hypercube | Castro's |
|---|---|---|---|
| Average | 3.67 | 68.5 | 129.78 |
| Best | 0.46 | 13.8 | 11.01 |
| Worst | 7.25 | 97.67 | 255.19 |

Fig. 7. Constructive heuristics (class I).

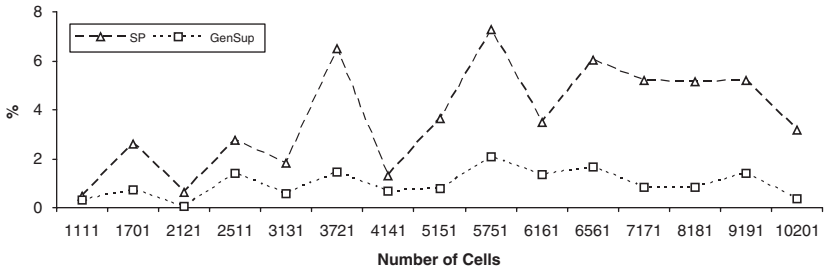


Fig. 8. Percentage gaps (class I).

### 6.4. Computational results

The assessment of the quality of the solutions produced by the heuristics was made, for instances with $LB \neq 0$, according to the standard formula

$$\text{gap} = \frac{UB - LB}{LB} \times 100\%,$$

where *UB* stands for the heuristic solution value. The two instances in the test set with $LB = 0$ did not need any complementary suppressions at all.

We implemented the hypercube and Castro's heuristics to compare their performance with those of the SP heuristic and of *GenSup*. To make the comparison fair, the solutions obtained with the hypercube method and with Castro's heuristic were subjected to the same clean-up procedure used in the SP heuristic to eliminate redundant suppressions in the solutions introduced in the initial population of *GenSup*.

For class I tables the hypercube method never failed to produce a feasible solution and Castro's heuristic never failed to find feasible sequences of arc-disjoint paths. To compare our results with those presented in [15], it must be taken into account that the denominator in our gap formula is *LB* (rather than *UB*) and that in [15] no clean-up was made, because it was considered too time consuming. The overall average gap, the best gap and the worst gap over the 15 sets of tables in class I are presented in Fig. 7.

For class II tables the results obtained do not allow a meaningful comparison. The hypercube method failed to produce a feasible solution for 143 instances and Castro's heuristic failed to find feasible sequences of arc-disjoint paths in 132 instances. To obtain feasible solutions with Castro's heuristic, one could resort to the infeasibility recovery procedure. However, solutions requiring a systematic call of that procedure would not be representative of the performance of the algorithm, as it was presented by the author: a method that avoids the efficiency problems of minimum cost network flow heuristics (see [15, p. 4]). The failures in class II tables were due to the large protection levels (remember that both levels are set to 15% of the cell value and that 10% of the marginal cells and 20% of the internal cells are confidential).

Thus, we next compare the performance of *GenSup* with that of the SP heuristic. For *GenSup*, the parameter values were set to $P = 20$, $\alpha = 20$ and $\beta = 10$, based on some experience performed with 30 tables, chosen arbitrarily in the whole set of instances.

Fig. 8 compares, for class I instances, the quality of the best solution generated by *GenSup* with the quality of the solution obtained with the SP heuristic.

The overall average gap for *GenSup* was 0.97% and its worst value, 2.07%, was smaller than the SP average, 3.67%.

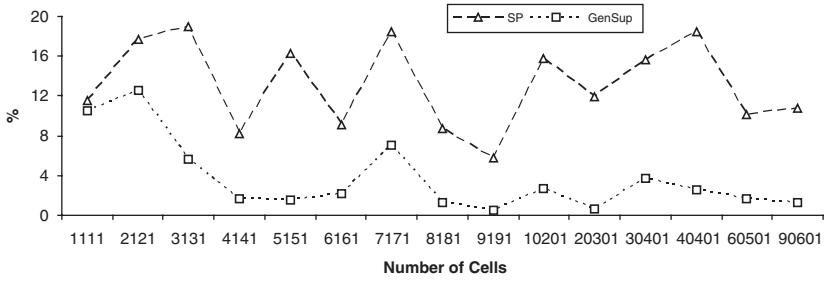


Fig. 9. Percentage gaps (class II).

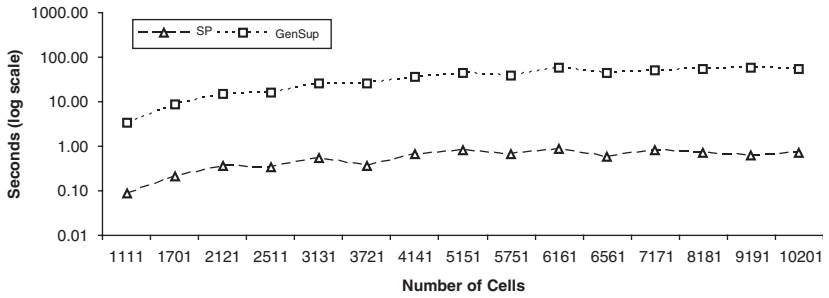


Fig. 10. Running times (class I).

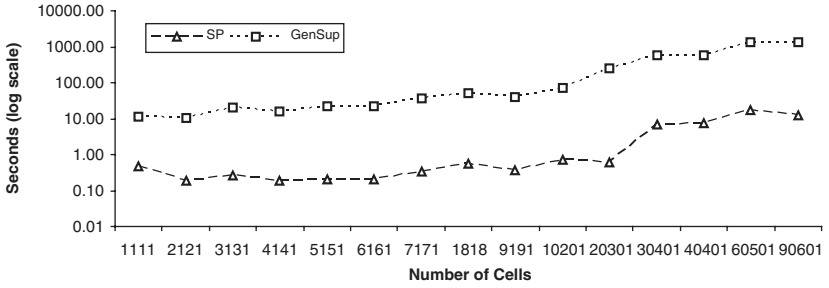


Fig. 11. Running times (class II).

We intended to run *GenSup* for larger tables (as for class II) but our experience was that, for tables with more than 10,000 internal cells, a dominant proportion of instances did not require complementary suppressions at all—no table with more than 300 rows, in our test set, required complementary suppressions. On top of that, for the tables requiring some protection, the average gap of the SP heuristic solutions was less than 0.5%, indicating that there is no need of a more powerful heuristic, in this case.

Fig. 9 presents the results obtained on class II instances.

The overall average gap was 3.7% for *GenSup* and 13.16% for the SP heuristic. *GenSup* gaps only reached two digits for the two smallest table sets, whereas for the SP heuristic two-digit gaps were scattered across the graphic. The performance of *GenSup* improved with the number of cells: the average gaps were 4.55%, for tables with less than 20,000 cells, and 2.00% for the remaining tables. For this class, the *GenSup* performance is still good for tables with more than 100,000 cells, in what solution quality is concerned. However, being a population-based method, it requires computing times that we consider too long.

To compare the results obtained in classes I and II it should be noted that, as mentioned before, the lower bound used to compute the gap is likely to be tighter for the class I tables.

Figs. 10 and 11 present the SP heuristic and the *GenSup* running times, for classes I and II, respectively. Times presented for *GenSup* include the constructive heuristics' time to generate the initial population.

These running times represent a good trade-off between time and solution quality. The gap reductions yielded by *GenSup* were in the range [35%, 92%], for class I tables, and in the range [9%, 95%], for class II tables. For tables with up to 10,000 internal cells, *GenSup* average running times were all less than 2 min. For class II tables with less than 60,000 cells, the average running times were still less than 10 min. In the last two sets (class II tables with more than 60,000 cells) the average computing times increased to approximately 22 min, but the gaps were reduced from about 10.5% to less than 1.5%.

## 7. Conclusions

This paper presents a genetic-based algorithm to solve the CSP. Its performance was tested on two classes of randomly generated instances, with up to 90,000 internal cells: class I instances reproduce real-world count tables, typical of sociological data, and class II instances reproduce real-world business data tables. In practice, class II tables are more difficult to protect than class I tables because, to cover large protection levels at a low loss of information, it is necessary to devise a complex pattern of suppressions. For class II, no results for tables with more than 10,000 internal cells are reported for the best exact algorithm in the open literature [9], and it did not solve to proven optimality 45 out of the 550 instances in the test set. The computational experience, reported in Section 6, shows that *GenSup* is very effective for both classes: it bridged over 70% of the optimality gap of the constructive heuristic solutions. For tables with up to 10,000 internal cells *GenSup* average running times were all less than 2 min. For the tables with more than 20,000 cells the *GenSup* average running times increased to 13 min but the percentage of the optimality gap bridged increased to 85%, which represents a very good quality vs. time trade-off.

## Acknowledgments

The authors thank two anonymous referees and the editor for their comments and suggestions that helped improve the paper.

## References

[1] Willenborg L, de Waal T. Elements of statistical disclosure control. New York: Springer; 2001.
[2] Doyle P, Lane JI, Theeuwes JM, Zayatz LV, editors. Confidentiality, disclosure, and data access—theory and practical applications for statistical agencies. Amsterdam: North-Holland; 2001.
[3] Domingo-Ferrer J, editor. Inference control in statistical databases—from theory to practice. Lecture notes in computer science, vol. 2316. Berlin: Springer; 2002.
[4] Domingo-Ferrer J, Torra V, editors. Privacy in statistical databases. Lecture notes in computer science, vol. 3050. Berlin: Springer; 2004.
[5] Fischetti M, Salazar JJ. Solving the cell suppression problem on tabular data with linear constraints. Management Science 2001;47(7): 1008–27.
[6] Gonzalez JF, Cox LH. Software for tabular data protection. Statistics in Medicine 2005;24:659–69.
[7] Kelly J, Golden B, Assad A. Cell suppression: disclosure protection for sensitive tabular data. Networks 1992;22:397–417.
[8] Goldberg D. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.
[9] Fischetti M, Salazar JJ. Models and algorithms for the 2-dimensional cell suppression problem in statistical disclosure control. Mathematical Programming 1999;84:283–312.
[10] Ahuja RK, Magnanti TL, Orlin JB. Network flows—theory, algorithms and applications. Englewood Cliffs, NJ: Prentice-Hall; 1993.
[11] Michalewicz Z. Genetic algorithms + data structures = evolution programs. Berlin: Springer; 1996.
[12] Reeves C, Rowe J. Genetic algorithms—principles and perspectives, a guide to GA theory. Boston: Kluwer Academic Publishers; 2003.
[13] Carvalho FD, Dellaert N, Osório M. Statistical disclosure in two-dimensional tables: positive tables. Report 9441/a, Econometric Institute, Erasmus University Rotterdam, The Netherlands, 1994.
[14] Carvalho FD, Almeida MT. An integer approach for the cell suppression problem in two-dimensional statistical tables. Working Paper #3-05, Centro de Investigação Operacional, 2005.
[15] Castro J. A shortest paths heuristic for statistical data protection in positive tables. 2005. Available at: ⟨http://www-eio.upc.es/∼jcastro⟩.
[16] Cox LH. Network models for complementary cell suppression. Journal of the American Statistical Association 1995;90(432):1453–62.
[17] Giessing S, Repsilber D. Tools and strategies to protect tables with the GHQUAR cell suppression engine. In: Domingo-Ferrer J, editor. Inference Control in Statistical Databases—from Theory to Practice. Lecture notes in computer science, vol. 2316. Berlin: Springer; 2002. p. 181–92.

[18] de Wolf P-P. HiTaS: a heuristic approach to cell suppression in hierarchical tables. In: Domingo-Ferrer J, editor. Inference control in statistical databases—from theory to practice. Lecture notes in computer science, vol. 2316. Berlin: Springer; 2002. p. 74–82.
[19] Carvalho FD, Almeida MT. Lower-bounding procedures for the 2-dimensional cell suppression problem. European Journal of Operational Research 2000;123:29–41.
[20] Carvalho FD, Dellaert N, Osório M. Statistical disclosure in two-dimensional tables: general tables. Journal of the American Statistical Association 1994;89:1547–57.