# A Genetic Approach to Statistical Disclosure Control

Jim E. Smith, Alistair R. Clark, Andrea T. Staggemeier, and Martin C. Serpell

***Abstract*—Statistical disclosure control is the collective name for a range of tools used by data providers such as government departments to protect the confidentiality of individuals or organizations. When the published tables contain magnitude data such as turnover or health statistics, the preferred method is to suppress the values of certain cells. Assigning a cost to the information lost by suppressing any given cell creates the "cell suppression problem." This consists of finding the minimum cost solution which meets the confidentiality constraints. Solving this problem simultaneously for all of the sensitive cells in a table is NP-hard and not possible for medium to large sized tables. In this paper, we describe the development of a heuristic tool for this problem which hybridizes linear programming (to solve a relaxed version for a single sensitive cell) with a genetic algorithm (to seek an order for considering the sensitive cells which minimizes the final cost). Considering a range of real-world and representative "artificial" datasets, we show that the method is able to provide relatively low cost solutions for far larger tables than is possible for the optimal approach to tackle. We show that our genetic approach is able to significantly improve on the initial solutions provided by existing heuristics for cell ordering, and outperforms local search. This approach is then extended and applied to large statistical tables with over 200 000 cells.**

***Index Terms*—Statistical disclosure control.**

## I. Introduction

IN TODAY'S "knowledge economy" many organizations hold large amounts of data gathered from a variety of sources, some of which they wish to publish, sell, or otherwise exploit and disseminate, whilst respecting the privacy of individual sources. As for their counterparts in most countries, the U.K. Office for National Statistics (ONS), Newport, has a duty to protect the confidentiality of "sensitive" data in published tables, achieving this via a number of approaches collectively known as statistical disclosure control [26].

These approaches either change the values of the cells in the tables (perturbative) or do not (non-perturbative). Perturbation methods tend to be less computationally expensive than non-perturbation methods and therefore can be applied to larger tables. The methods known as rounding and controlled rounding [9] either lose the additivity of the table and/or modify the margin totals reducing the usefulness of the table to the end user. To reduce this problem Castro [5] developed a new minimum-$L_2$-distance perturbation method which maintains both additivity and the margin totals and has been shown to protect 3-D tables with up to 1 000 000 cells. The main non-perturbation method is known as cell suppression which, when done optimally or near optimally, can only be applied to smaller tables than the perturbation methods as it involves solving a difficult combinatorial optimization. It is the objective of this paper to extend cell suppression, which preserves more of the original cell values than perturbation methods, so that it can be applied to larger tables. Cell suppression suppresses not only the values of the sensitive cells in a published table, but also those of some additional "secondary" cells. These are chosen to prevent the calculation of the sensitive cells' values while keeping information loss to a minimum. To use an analogy, the problem is similar to that of creating a Sudoko problem where it is impossible to assign a value to one or more specified cells. The equivalent optimization task would be to find a version of the Sudoko table in which as many cells as possible have their values published (or can be calculated), while still meeting the "impossibility" constraint. In practice, an attacker can identify the minimum and maximum possible values of the suppressed cells by solving two similar linear programs (LPs) per cell. The table is considered "protected" if an attacker is unable to estimate the sensitive cells' values within specified limits. Fischetti and Salazar [10], [11] formulated this "cell-suppression" problem as a complex mixed integer program (MIP) and optimally solved it (for small tables) using Benders decomposition and branch-and-cut with valid inequalities. Integral to their approach is the construction of approximate bounds via an efficient LP-based heuristic procedure from [15] and [17]. This constructs a solution by processing a specified sequence of the sensitive cells, gradually building up a secondary suppression pattern so as to meet the protection constraints, while minimizing information loss.

This MIP approach has been incorporated into widely used tools such as $\tau$-Argus [18], [25], along with a range of existing heuristic approaches. However, the current tools leave much to be desired. As currently implemented, the output from the LP heuristic is not available to the user, and because of the large numbers of constraints and variables, the "optimal" approach is only possible for tables with a few hundreds

Manuscript received September 16, 2010; revised January 14, 2011, April 11, 2011, and May 3, 2011; accepted May 7, 2011. Date of current version May 24, 2012. This work was supported by EPSRC through an EPSRC Math CASE Award and by the Office of National Statistics.

J. E. Smith, A. R. Clark, and M. C. Serpell are with the University of the West of England, Bristol BS16 1QY, U.K. (e-mail: james.smith@uwe.ac.uk; alistair.clark@uwe.ac.uk; martin2.serpell@uwe.ac.uk).

A. T. Staggemeier is with Analytic Intelligence Architecture Solutions Ltd., Rogerstone, Newport NP10 9EB, U.K. (e-mail: info@analyticintelligence.co.uk).

Digital Object Identifier 10.1109/TEVC.2011.2159271

(or at best very few thousands) of cells. Compared to the size of tables that ONS and other national statistics agencies wish to publish, these are tiny. To give an example, an analysis of industrial activity broken down by region and activity type might have millions of cells and several dimensions, each with different levels of hierarchy. For 2-D non-hierarchical tables optimal methods based on a "network flow" formulation are possible for larger tables [4], [10], and recently a hybrid genetic approach has been proposed to extend the scalability of this approach [1]. However, these are not applicable to multi-dimensional or hierarchical tables. Alternative heuristic approaches such as the hypercube method [25] can be used to protect larger tables, but it is well known that even on smaller tables they significantly "over-protect"—causing significantly greater than necessary information loss [14].

In this paper, we describe the development and analysis of a heuristic method for solving larger tables. The approach adopted uses a genetic algorithm (GA) to optimize the sequence in which the sensitive cells are fed into the linear program (incremental attacker heuristic) from [11], to build up a suppression pattern. We compare our approach with several fixed heuristics for ordering the sensitive cells, and to the use of local search methods. We also compare the effects of different mutation operators (equivalently search neighborhoods) for the genetic algorithm (respectively local search). As the decision as to what values to assign to the parameters that control a GA have a great impact on its performance we introduce self-adaption of the mutation operator and probability. The self-adaption of mutation parameters has been proved successful in the continuous domain [3], [19] and for binary combinatorial problems [2], [12], [16], but here we use it for a permutation problem. The rest of this paper proceeds as follows. Section II provides a mathematical formulation of the cell suppression problem and of the linear programs used to solve the relaxed incremental version. Section III describes our experimental framework and the data sets used for this paper. Section IV describes the results from experiments comparing local and genetic algorithm searches. Section V looks at ways to reduce the cost of the fitness function. Section VI describes the results from experiments comparing the performance of the genetic algorithm with that of the models in $\tau$-Argus. Section VII considers ways in which the incremental attacker heuristic can be modified to protect larger statistical tables. In Section VIII, we draw conclusions and suggest future work.

## II. Background

### A. A Model of the Cell Suppression Problem

Fischetti and Salazar [11, p. 1010] gave the following formal definition of the cell suppression problem (CSP).

> A *table* is a data vector $a = [a_1, ..., a_n]$ whose entries satisfy a given set of linear constraints known to a possible attacker
>
> $$\left.\begin{array}{rl} My = b & \\ lb_i \le y_i \le ub_i & \forall i = 1, ..., n \end{array}\right\}. \quad (1)$$
>
> In other words, (1) 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 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 $\{0, \pm 1\}$ matrix and $b = 0$.

The attacker can deduce a value $y_i$ for cell $i$, whereas its actual value is $a_i$. Note that an attacker is assumed to know the values $lb_i$ and $ub_i$ of the lower and upper "external bounds." This may not be a realistic assumption. They go on to state:

> Given a *nominal* table **a**, let $PS = \{i_1, ..., i_p\}$ be the set of sensitive cells to be protected, as identified by the statistical office according to some criteria. For each sensitive cell $i_k (k = 1, ..., p)$, the statistical office provides three nonnegative values: $LPL_k$, $UPL_k$, and $SPL_k$, the lower protection level, upper protection level, and sliding protection level, and so on.
>
> A suppression pattern is a subset of cells $SUP \subseteq \{1, ..., n\}$ corresponding to the unpublished cells. A consistent table with respect to a given suppression pattern SUP and to a given nominal table a is a vector $y = [y_1, ..., y_n]$ satisfying
>
> $$\left.\begin{array}{rl} My = b & \\ lb_i \le y_i \le ub_i & \forall i \in SUP \\ y_i = a_i & \forall i \notin SUP \end{array}\right\} \quad (2)$$
>
> 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 $i_k (k = 1, ..., p)$ there exist two feasible tables, say $f^k$ and $g^k$, such that $f^k_{i_k} \le a_{i_k} - LPL_k$, $g^k_{i_k} \ge a_{i_k} + UPL_k$ and $g^k_{i_k} - f^k_{i_k} \ge SPL_k$.

In fact for the purposes of this paper "less/more than or equal to" inequalities in expressions above are replaced by "strictly less/more than" inequalities to be consistent with ONS' understanding of protection limits. This is not a trivial distinction given that table data values and protection limit values tend to be integer and often small. The result is usually a distinctly larger set of suppressed cells when the table has many integer values, i.e., frequency tables and certain magnitude tables.

Knowing the external bounds $lb_i$ and $ub_i$ for all cells $i = 1, ..., n$ and which cells have been suppressed in the published table, an attacker will try to discover the minimum and maximum possible values, of each cell. For a given sensitive cell $i_k$, solving an LP to minimize (maximize) $y_{i_k}$ subject to constraints (2) provides the minimum (maximum) possible values $f^k_{i_k}$ ($g^k_{i_k}$).

To conform to the *ONS* understanding of sufficient protection, we apply a modified version of the standard model

TABLE I
EXAMPLE OF AN OPTIMAL SUPPRESSION PATTERN. CELL (3, 3), DARK SHADED, IS SENSITIVE. SECONDARY SUPPRESSED CELLS ARE SHOWN IN A LIGHTER SHADE. SUB-OPTIMAL METHODS WOULD SUPPRESS MORE CELLS OR MORE INFORMATIVE ONES SUCH AS ROW/COLUMN TOTALS

| | **1** | **2** | **3** | **4** | **5** | **Total** |
|---|---|---|---|---|---|---|
| **1** | 4 | 4 | 4 | 4 | 4 | 20 |
| **2** | 4 | 4 | 4 | 4 | 4 | 20 |
| **3** | 4 | 4 | 121 | 4 | 4 | 137 |
| **4** | 4 | 4 | 4 | 4 | 4 | 20 |
| **5** | 4 | 4 | 4 | 4 | 4 | 20 |
| **Total** | 20 | 20 | 137 | 20 | 20 | 217 |

which states that the sensitive cell $i_k$ is sufficiently protected if the solutions to these LPs satisfy $min(y_{i_k}) < LPL_k$ and $UPL_k < max(y_{i_k})$. It has been asserted that if this condition is satisfied for all sensitive cells $i_k$, then the whole table is feasible, i.e., sufficiently protected. However, given that the attacker will not know which of the suppressed cells are the sensitive ones, this condition should really be satisfied not just for each sensitive cell $i_k$, but also for each secondarily suppressed cell within the set SUP. If not, then the values of certain secondarily suppressed cells might be guessed, subverting the protection of the sensitive cell.

### *B. Incremental Attacker*

Fischetti and Salazar [11] stated that their branch-and-cut (BC) approach finds an optimal set of secondarily suppressed cells that guarantees protection for all sensitive cells in a table. The approach is sophisticated, time-consuming and identifies optimal solutions only for moderately sized tables. However, the authors do make use of a fast heuristic to find incumbent solutions at each node of the BC tree, based on a heuristic procedure from Kelly *et al.* [15] and Robertson [17]. The heuristic starts by taking as input the set of sensitive cells $P = \{i_1, ..., i_p\}$ and the sequence in which to protect them. The set SUP of suppressed cells is initially set equal to the set of sensitive cells. For each sensitive cell in turn, the set SUP is then augmented by solving two LPs. These use the cell weights, consistency equations, upper and lower bounds, and upper and lower protection limits provided by $\tau$-Argus to determine what extra cells must be suppressed to satisfy the protection requirements. These are added to SUP and the process iterates to protect the next sensitive cell in the sequence.

The sequence used is heuristically determined according to decreasing weight in [11], but our preliminary experimentation confirmed that even for a table with only 70 cells, the ordering can make as much as 30% difference to the total cost. Thus, in our method the permutation is the key decision, as it defines the solution space in our evolutionary algorithm.

The first LP, known as the UPL incremental attacker problem, identifies which cells need to be added to the set SUP so as to guarantee that a given sensitive cell $i_k$ is protected with respect to its upper protection limit $UPL_k$. For a given $i_k$, the LP is as follows:

$$minimize \qquad \sum_{i=1}^{n} c_i(y_i^+ + y_i^-) \tag{3}$$

TABLE II
FACTORS USED TO CREATE THE RANDOMLY GENERATED TABLES

| Rows | Columns | % Sensitive Cells | % Cells with Value 0 |
|---|---|---|---|
| 200 | 5 | 10 | 25 |
| 200 | 5 | 2 | 5 |
| 200 | 50 | 10 | 25 |
| 200 | 50 | 2 | 5 |
| 4000 | 10 | 10 | 25 |
| 4000 | 10 | 2 | 5 |

$$such\ that \qquad M(y^+ - y^-) = b \tag{4}$$

$$0 \le y_i^+ \le UB_i \quad \forall i = 1, ..., n \tag{5}$$

$$0 \le y_i^- \le LB_i \quad \forall i = 1, ..., n \tag{6}$$

$$y_{i_k}^- = 0 \quad and \quad y_{i_k}^+ = UPL_{i_k} \tag{7}$$

where:

1) $y_i = a_i + y_i^+ - y_i^-$ is the attacker's estimate of the value of a cell $i \in \{1, ..., n\}$ so that the non-negative decision variables $y_i^+$ and $y_i^-$ are, respectively, the deviations above and below of $y_i$ from the cell value $a_i$;
2) $y_{i_k}^+$ and $y_{i_k}^-$ are $y_i^+$ and $y_i^-$ for a given sensitive cell $i_k$;
3) $UB_i = ub_i - a_i \ge 0$ is the relative external upper bound on $y_{i_k}^+$;
4) $LB_i = a_i - lb_i \ge 0$ is the relative external lower bound on $y_{i_k}^-$;
5) the objective function coefficient $c_i = 0$ for all $i \in SUP$ and $c_i$ = cell weight $w_i$ for all $i \notin SUP$.

After solving this LP, the set SUP is augmented with all cells $i \notin SUP$ for which $y_i^+ + y_i^- > 0$ in the optimal solution. After setting $c_i = 0$ for the set SUP's newly added cells $i$ resulting from this solution, the second LP similarly identifies which cells need to be added to SUP so that sensitive cell $i_k$ is protected with respect to its lower protection limit $LPL_k$ by replacing the last constraint line with $y_{i_k}^+ = 0$ and $y_{i_k}^- = LPL_k$. As noted above, the working definitions of protection used at ONS are stricter than those used in the formal model, and experimentation has revealed other subtle problems.

## III. METHODOLOGY

### *A. DataSets*

Thirty eight data sets were provided by ONS in the ".jj" format as output by Tau Argus. Of these, four were real world tables (two non-hierarchical and two hierarchical). Another four were hierarchical data tables from the $\tau$-Argus distribution. The remaining 30 non-hierarchical magnitude datasets were created with ONS's randomized data set generator, a sophisticated tool which can be tuned to replicate the distribution of values typically found in different types of tables. There were five randomly created instances for each of the following classes.

### *B. Algorithms*

The algorithms devised and implemented in this paper use different search techniques to find the "best" sequence in which to protect sensitive cells using the linear programming

(incremental attacker) heuristic. The combination of local search or genetic algorithms with the linear programming heuristic was first reported in [23]. Here, we expand on those results and present improvements to two of the major issues cited: the need for automatic selection of mutation operators, and the need for a more effective linear programming model. The hybrid techniques developed are also compared with existing algorithms. The algorithms were implemented in the C++ language using the open source COIN-OR framework—in particular the OSI framework for defining LP problems and the CLP solver [7]. Initial experimentation showed that the code ran approximately five times faster when using a commercial LP solver such as CPLex [6]. However using the public domain CLP solver, facilitated the running the experiments in parallel which more than compensated for its slower speed.

The experiments were designed to determine whether there was any benefit to the use of a population-based approach as opposed to a simple local search method. A second goal was to determine the effect of changing the way in which solutions are perturbed by mutation (in the GA) or in the local search routine. This was then extended to see if the selection of the mutation operator and probability could be left to the GA itself.

In order to explain the results better we begin by describing the working of the genetic algorithm used.

1) An initial population of potential solutions (i.e., orderings of the sensitive cells) is created using the following heuristics:
   a) ordered by weight (cost) of the cells as per [11];
   b) using random permutations.
2) Each solution in this population is evaluated by creating the suppression set and counting its total cost.
3) Two parents are selected by tournament and an offspring produced by recombination, then mutation.
4) The new offspring is evaluated and compared to the member of the population with the highest cost, replacing it if the offspring's cost is lower.
5) If the criteria for ending the run has been met, the process stops, otherwise it returns to step 3.

This framework was explicitly designed to be flexible and allow the use of different mutation operators to perturb existing solutions. For local search the population size is simply set to one. Preliminary work showed population sizes of 50 to be too large, since on the bigger tables the time-allowance was used before the initial population had been evaluated, i.e., before the processes of simulated evolution had time to create and select new lower-cost solutions. In the light of this experience, it was necessary to use a smaller population size. Given the existence of two heuristics (increasing weight and decreasing weight) for creating solutions with which to "innoculate" the search (see below), and the findings in [24] that these should not represent the major part of the initial population, we used a population size of ten for the genetic algorithm. Thus, the choice of population size is driven by the desire to solve bigger tables, where in general it takes longer to solve each LP, rather than by specific characteristics of any particular data. Given the small population size, and the use of heuristics to innoculate the initial population, it was important to avoid the risks of premature convergence. Therefore, rather than using fitness-proportionate selection, we used rank-based binary tournaments, always selecting the fittest candidate.

The recombination operator used was Davis' "order-based" crossover, a permutation-specific operator [8]. This was chosen as it was specifically designed to mix the *absolute* order in which items (in our case primary cells) occur on the two parents whilst also preserving that information that is common to both parents, i.e., that has been "learnt" by the algorithm. Order-based crossover copies a segment, between two random crossover points, from one parent to the offspring. Then starting from the second crossover point and wrapping around at the end of the list copies the remaining unused numbers from the other parent to the offspring. A fixed rate of 0.7 was taken as standard from the literature.

Three different neighborhood generation operators were used for the local search/mutation steps, namely the following.

1) Insertion: pick two random values in the permutation, and move the second to just behind the first, moving the intermediate elements along to accommodate the change.
2) Swap: swap the position of two randomly chosen elements.
3) Inversion: reverse the order of a randomly selected substring.

These combinations of operators and parameters produced three local search algorithms (LS-Swap, LS-Insert, and LS-Invert) and three genetic algorithms (GA-Swap, GA-Insert, and GA-Invert), initially using a fixed mutation probability of $1/len$ where $len$ is the length of the sequence to be optimized and again this is taken as standard from the literature.

The fitness cost used by the GA is $\sum_{i=1}^{n} z_i w_i$, where $z_i = 1$ if cell $i$ is suppressed and 0 if it is not suppressed. $w_i$ is the weighting given to the information loss should cell $i$ be suppressed. Which cells are suppressed is determined using the linear programming (incremental attacker) heuristic which as 3 shows necessarily works by minimizing a partial fraction ($y^+$ and $y^-$ are continuous variables) rather than the full amount since that would present a non-linear problem. Therefore, the fitness function for the GA is the combination of the linear programming (incremental attacker) heuristic followed by the summation of the *full* cost of every suppressed cell.

### C. Presentation of Results

Initial inspection of the data showed that even for the same table size, the differences in the values of the results obtained depended far more on the contents of the table (i.e., on the value used to seed the randomized table creation process) than on the approach taken. Naturally, the size of the table was also a major contributing factor, since bigger tables with more primary cells almost inevitably had higher cost solutions associated with them. While the analysis of the strength of different effects has to be treated with caution, a highly significant finding was that in all cases the result obtained was better than or equal to that produced by any

TABLE III
AVERAGE PERCENTAGE IMPROVEMENT OF THE FIVE NORMALIZED SUPPRESSION PATTERN COSTS FOR PROTECTING THE 38 STATISTICAL TABLES

| Table Type | LS Swap | LS Insert | LS Invert | GA Swap | GA Insert | GA Invert |
|---|---|---|---|---|---|---|
| 200×5 | 9.66 | 8.60 | 9.89 | 10.27 | **10.47** | 8.51 |
| sens = 0.02 | **6.33** | 2.88 | 3.85 | 5.78 | 4.81 | 3.04 |
| zeros = 0.05 | 16.31 | 7.66 | 17.25 | **22.08** | 20.09 | 20.58 |
| | 24.12 | 18.15 | 25.34 | **25.68** | 25.35 | 25.46 |
| | **1.02** | 0.76 | 0.58 | 0.69 | 0.85 | 0.74 |
| 200×5 | 13.66 | 14.63 | 15.57 | 12.88 | 13.84 | **17.03** |
| sens = 0.10 | **1.86** | 1.67 | 1.10 | 1.24 | 1.25 | 1.21 |
| zeros = 0.25 | **17.54** | 17.13 | 14.25 | 14.91 | 10.30 | 16.71 |
| | 6.77 | 7.94 | 8.00 | 9.35 | **21.86** | 17.59 |
| | 13.72 | 12.21 | 11.42 | **15.40** | 12.32 | 14.54 |
| 200×50 | 1.96 | 1.87 | 7.40 | **8.95** | 6.50 | 5.69 |
| sens = 0.02 | 2.77 | 2.21 | 2.58 | 7.77 | 8.48 | **9.27** |
| zeros = 0.05 | 3.54 | 1.99 | 8.70 | 7.39 | **9.25** | 6.45 |
| | 6.22 | 1.80 | 12.36 | **13.72** | 13.36 | 13.21 |
| | 3.03 | 4.28 | 3.71 | **5.59** | 5.19 | 2.11 |
| 200×50 | 1.10 | 0.94 | 2.12 | 1.37 | 0.19 | **2.40** |
| sens = 0.10 | 6.24 | 0.13 | 9.41 | 10.33 | 9.28 | **14.10** |
| zeros = 0.25 | 3.12 | 2.96 | 0.75 | 10.62 | 11.51 | **15.12** |
| | 4.41 | 1.41 | 11.24 | 10.52 | 9.96 | **12.78** |
| | 3.29 | 1.79 | **5.96** | 2.66 | 0.75 | 1.61 |
| 14×654 | 1.35 | 0.00 | 1.08 | 1.26 | 1.43 | **2.46** |
| sens = 0.186 | | | | | | |
| zeros = 0.16 | | | | | | |
| 14×654 | 0.00 | 0.00 | 0.68 | 0.00 | **0.84** | 0.20 |
| sens = 0.19 | | | | | | |
| zeros = 0.16 | | | | | | |
| 712×10 | 0.59 | 0.62 | **0.72** | 0.71 | 0.62 | **0.72** |
| sens = 0.06 | | | | | | |
| zeros = 0.49 | | | | | | |
| 712×10 | 0.27 | 0.46 | 0.94 | **1.18** | 1.05 | 1.17 |
| sens = 0.08 | | | | | | |
| zeros = 0.49 | | | | | | |
| 712×19 | 0.00 | 0.00 | 0.07 | 1.17 | 0.91 | **1.66** |
| sens = 0.11 | | | | | | |
| zeros = 0.35 | | | | | | |
| 712×19 | 0.07 | 0.07 | 0.52 | 1.10 | 1.00 | **1.31** |
| sens = 0.13 | | | | | | |
| zeros = 0.35 | | | | | | |
| 14×1433 | 0.00 | 0.15 | 1.17 | 1.35 | **3.10** | 1.16 |
| sens = 0.16 | 3.65 | 1.73 | 3.11 | 4.38 | 4.66 | **6.33** |
| zeros = 0.14 | | | | | | |
| 4000×10 | **0.82** | 0.65 | 0.43 | 0.18 | 0.25 | 0.25 |
| sens = 0.02 | 1.22 | 0.61 | 1.60 | 2.20 | **2.34** | 2.30 |
| zeros = 0.05 | 0.59 | 1.05 | 0.66 | **1.21** | 1.16 | 1.16 |
| | 0.86 | 0.78 | 1.10 | 0.94 | 1.01 | **1.14** |
| | **0.35** | 0.00 | 0.28 | 0.11 | 0.16 | 0.31 |
| 4000×10 | 0.14 | 0.14 | 0.02 | **0.41** | 0.33 | 0.24 |
| sens = 0.10 | 0.79 | 0.52 | **1.03** | 1.02 | 0.81 | 0.60 |
| zeros = 0.25 | 0.69 | 0.66 | 0.72 | 0.72 | **0.90** | 0.60 |
| | 0.38 | 0.39 | 0.41 | **0.52** | 0.39 | 0.47 |
| | 0.14 | 0.31 | **0.61** | 0.57 | 0.58 | **0.61** |

The largest improvement for each of the statistical tables has been placed in bold, these indicate the algorithm that has performed the best for each statistical table.

of the original heuristics. In this light, it was decided to undertake a further analysis where the results on each table are normalized relative to the cost of the equivalent order-by-weight solution. This metric indicates the relative magnitude of the improvement found over current methods, and partially alleviates the strength of the table as factor. It is this metric that is presented for comparison in Table III.

### D. Analysis

The results from each run were analyzed using the statistical package SPSS. To see if there was any significant difference in the performance of the different algorithms we used the non-parametric Friedman's ANOVA test. This uses rank ordering of the suppression pattern costs generated by the different algorithms for each of the statistical tables in turn it is not effected by the different table properties like size or number of primary cells. Wilcoxon's signed rank test has been used when a comparison of the performance of just two of the algorithms was required.

TABLE IV
FRIEDMAN'S ANOVA RANKING OF THE AVERAGE PERCENTAGE IMPROVEMENT OF THE SUPPRESSION PATTERN COSTS FOR THE SIX ALGORITHMS FOR SIX OF THE STATISTICAL TABLES

| Performance | Algorithm | Mean Rank |
|---|---|---|
| Best | GA Invert | 4.33 |
| | GA Swap | 4.25 |
| | GA Insert | 4.11 |
| | LS Invert | 3.47 |
| | LS Swap | 2.83 |
| Worse | LS Insert | 2.01 |

## IV. COMPARING LOCAL AGAINST GENETIC ALGORITHM SEARCH

### A. Procedure

In this section, we compare the performance of local and genetic algorithm searches when applied to finding the lowest cost suppression patterns used to protect statistical tables. For each of the six algorithms (LS-Swap, LS-Insert, LS-Invert, GA-Swap, GA-Insert, and GA-Invert) five runs were made on each of the 38 tables provided. All runs used the following termination criteria, stopping whichever occurred first:

1) 3 h of computer time were used up;
2) 10 000 evaluations were used;
3) 1000 evaluations had passed since the last improvement;
4) the population mean cost was within 1% of the best cost for 100 successive iterations (not used for local search).

For each run we recorded the cost of the best solution found, the number of evaluations after which it was found, and the suppression set. The final suppression set was then fed back in to the maximum and minimum attacker programs to confirm that it provided adequate protection for the statistical table. The averaged normalized results are presented in Table III.

### B. Analysis

Of the 38 statistical tables 13 were best (or equally best) protected by GA-Invert, ten by GA-Swap, seven by GA-Insert, six by LS-Swap, four by LS-Invert, and none by LS-Insert. Thirty of the statistical tables were best (or equally best) protected by the genetic algorithms and ten by the local searches, as shown in Table III. Table IV shows how the Friedman's ANOVA test ranked the three local search and three genetic algorithms for the 38 statistical tables.

Further analysis using the Wilcoxon's signed rank test indicated that with over 99.9% confidence the GA using the mutation operator Invert (GA-Invert) outperformed the equivalent local search, that with 99.9% confidence the GA

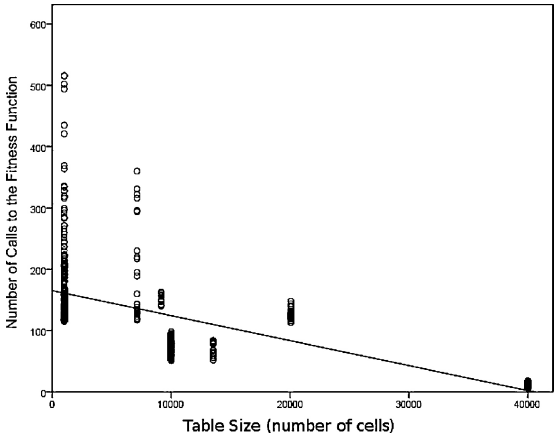


Fig. 1. Number of calls to the fitness function made by the genetic algorithms by table size.

using the mutation operator Swap (GA-Swap) outperformed the equivalent local search and that with 97.6% confidence the GA using the mutation operator Insert (GA-Insert) outperformed the equivalent local search. The Wilcoxon's signed rank test did not find a significant difference between the average performances of GA-Swap, GA-Insert, and GA-Invert.

The percentage improvements shown in Table III indicate that the best improvements occurred for the smallest statistical tables. An analysis of the number of calls to the fitness function made by the genetic algorithms by the table size showed, with over 99.9% confidence, that the number of calls to the fitness function decreases as the size of the statistical table increases (see Fig. 1). This clearly shows that when protecting statistical tables with 40 000 cells that in the 3 h allowed for the genetic algorithms to run they could do little more than initialize their parent pools. This is because as the table size grows, so does the time taken in the fitness function, which involves solving $2 \cdot |P|$ linear programs to identify the suppression pattern. For the smaller statistical tables we can see that there is a wide range of calls to the fitness function and this has three possible explanations. The first is that it is due to the different values and locations of the primary cells in each of the tables, the second is that the genetic algorithms are getting stuck in a local optima, and the third is that it may indicate early termination due to the population mean cost being within 99% of the best cost for 100 successive iterations.

### C. Conclusion

The results presented in Table III clearly indicate that the use of a meta-heuristic search strategy is beneficial. Solutions to the cell suppression problem are found with lower information loss than are achieved via the current "order-by-weight" heuristics. In cases where the heuristic are able to run for a significant number of iterations, the suppression pattern cost is reduced on average by 25.7% of the original cost. As expected, the results demonstrate that the total numbers of cells, rows, and columns are major factors in determining the magnitude of the improvements attainable. Since these factors directly affect the number of constraints which must be dealt with by the linear programs, they directly relate to the run-time needed to evaluate the partial solution for each primary cell. For the same reasons the proportion of sensitive cells to be protected is also a factor. Since most runs terminate due to the time criteria, it is reasonable to expect that a faster LP implementation would permit greater numbers of iterations and hence better results. The time allowed is dictated by the practical constraints of the workplace. For more subtle reasons, the specific cell contents can have a major effect. Thus if one respondent is much the biggest, then the corresponding cell will dominate the marginal totals in which they participate, and so it may be necessary to suppress many other cells, regardless of the order in which the primaries are considered.

As the number of evaluations is limited by time, the size of the initial population was successfully reduced to ten allowing more time to search the fitness landscape. This is probably as small as it can go without risking a serious loss of diversity which could adversely affect evolvability, especially given that the initial population is not purely random, but includes results from existing heuristics.

The analysis of the results presented in Table III has clearly shown that in this case using a genetic algorithm outperforms using local search. The analysis of the number of calls to the fitness function has shown that the genetic algorithms require longer than the 3 h that they have been given to search the fitness landscape. To allow this for the larger statistical tables the time limit is increased in all later tests, and the termination criteria that the population mean cost being within 1% of the best cost for 100 successive iterations is removed.

On individual statistical tables there are significant differences between fixed operator combinations. In [21], we showed that similar behavior was shown for the traveling salesman problem and that self-adaption avoided the possible pitfalls of choosing the wrong operator. The use of self-adaption in this case was tested experimentally and shown to perform at least as well as the use of fixed operators, the results are not shown to save space. Therefore, all future genetic algorithms in this paper will use self-adaption to select the mutation operator and mutation probability. These are encoded in two extra genes with a 0.1 probability of randomly changing for each iteration of the genetic algorithm [21].

## V. Reducing the Cost of the Fitness Function

### A. Procedure

To protect a statistical table using this approach requires that two linear programs are run for each primary suppressed cell in the statistical table. As the size of the statistical table that is being protected increases so does the average number of primary suppressed cells that need to be protected and therefore the number of linear programs that need to be run. Simultaneously, the time taken to run each linear program increases. The combination of these two affects the size of the statistical table that can be protected using a particular mathematical solver when protecting statistical tables one cell at a time. We have found that using the CLP solver to implement this LP restricts us to protecting statistical tables with less than or equal to 40 000 cells.

In order to allow us to protect larger tables the LP has been modified to reduce the number of linear programs that are required to be run. The following modifications to the LP have been made to allow it to protect larger statistical tables.

1) A preprocessing optimization is used to identify a subset of the primary suppressed cells ($P$) called the candidate initially exposed primary suppressed cells ($K$). The LP has been modified to only protect members of $K$ as protecting this subset still guarantees to protect all of the primary suppressed cells in the statistical table [20].
2) Protecting the primary suppressed cells in groups as opposed to individually also decreases the number of linear programs that are required to be run. Therefore, a further modification to the LP has been made to protect primary suppressed cells in groups.

The performance of these modifications to the LP was compared by using them to protect the 38 statistical tables described in Section III-A. Three algorithms were compared, all were self-adaptive GAs that invoked the LP as their fitness function. These algorithms were allowed to run for up to 12 h. The algorithms protect all primary cells ($P$) one at a time (GA-one-P), protect candidate initially exposed primary suppressed cells ($K$) one at a time (GA-one-K), and protect candidate initially exposed primary suppressed cells ($K$) in 40 groups (GA-group-K).

### *B. Analysis*

Of the 38 tables that were protected 22 were best or equally best protected by GA-group-K, 12 by GA-one-K, and seven by GA-one-P. The Wilcoxon's signed ranks test indicated, with 98.2% confidence, that on average GA-one-K produced lower cost suppression patterns than GA-one-P. It also indicated, with 96.8% confidence, that on average GA-group-K produced lower cost suppression patterns than GA-one-P. There was no significant difference in the performance of GA-one-K and GA-group-K.

Of the ten tables with 40 000 cells eight were best protected by GA-group-K, one by GA-one-K and one by GA-one-P. When considering only these tables the mean rankings given by Friedman's ANOVA of the cost (information loss) were 1.22 (GA-group-K), 2.22 (GA-one-K), and 2.56 (GA-one-P). The Wilcoxon's signed ranks test indicated that on average GA-group-K was better than GA-one-K (98.7% confidence) and GA-one-P (99.3% confidence).

Protecting only the candidate initially exposed primary cells (K) instead of all the primary cells (P) improves the performance of the self-adaptive GA because of the following.

1) Only the initially exposed primary cells (I) need to be protected and I is a subset of the candidate initially exposed primary cells (K). Protecting all the primary cells (P) may needlessly run LPs to protect consequentially exposed primary cells (C) which may lead to overprotection and they are guaranteed to be protected anyway if the initially exposed primary cells (I) are protected.
2) Less LPs are needed to be run to find the fitness of each permutation of cells as K is a subset of P. Hence, finding the fitness of each permutation is quicker and so in a given time more calls to the fitness function can be made which in turn leads to a greater search of the fitness landscape.

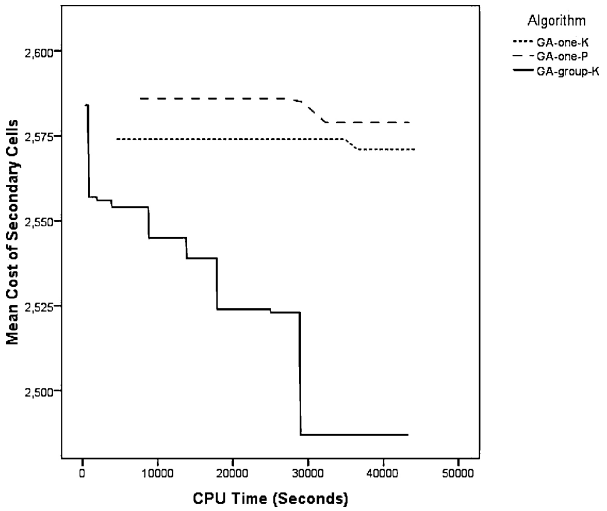


Fig. 2. Suppression pattern cost by CPU time (s) for the three algorithms.

For the smaller statistical tables (<30 000 cells) grouping the cells in K prior to protecting them may not lead to lower cost suppression patterns as it reduces the number of points on the fitness landscape. However, for the larger statistical tables grouping the cells in K prior to protecting them is better than protecting them one at a time as grouping limits the number of LPs required. A fixed number of groups means a fixed number of LPs to find the fitness of each permutation. Which again means more permutations can be examined in a given time (i.e., more searching of the fitness landscape).

It is the ability to search more of the fitness landscape that gives grouping the advantage for statistical tables with more than 30 000 cells. This can be clearly seen if we plot the suppression pattern cost by the CPU time required (Fig. 2) and by the number of calls to the fitness function (Fig. 3) for one of the 40 000 cell statistical tables. All three algorithms were still actively searching their fitness landscape when they terminated after their 12 h time limit. Fig. 2 shows that the algorithms that protected primary suppressed cells one at a time were only able to find one better solution in the time given whereas the algorithm that protected primary suppressed cells in groups found many better solutions. Fig. 3 shows why this is the case. The algorithms that protected primary suppressed cells one at a time made less than 150 calls to the fitness function in the allotted time whereas the algorithm that protected primary suppressed cells in groups made approximately 2000 calls to the fitness function in the allotted time.

### *C. Conclusion*

The grouping of candidate primary suppressed cells prior to protecting them significantly reduces the time taken to execute the fitness function. This, in turn, allows the genetic algorithm to better search its fitness landscape which in turn, on average, leads to lower cost suppression patterns. This has been found to be true for statistical tables with more than 30 000 cells.

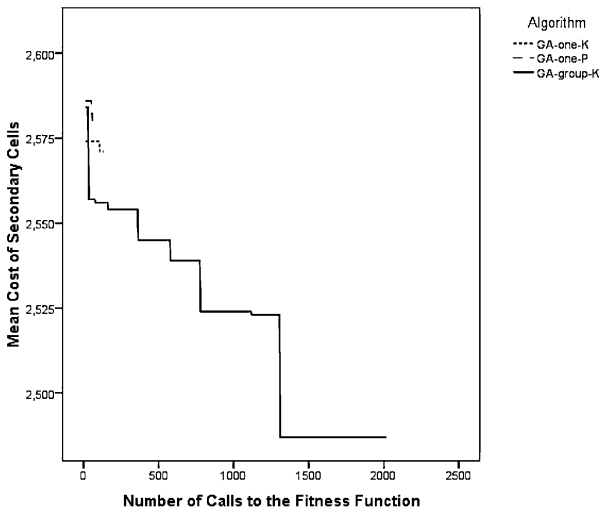


Fig. 3. Suppression pattern cost by the number of calls to the fitness function for the three algorithms.

## VI. Comparing the Genetic Algorithm Search Against Other Algorithms

### A. Procedure

In this section, we compare the performance of GA-group-K against that of the existing methods provided by the statistical disclosure control tool, $\tau$-Argus. The version of $\tau$-Argus used, for this comparison, was 3.2.0 build 6 (2004). This is the version of $\tau$-Argus that was available at ONS, where there was a compatible mathematical solver, when the comparison was carried out. Later versions of $\tau$-Argus have various improvements but these have not affected the cell suppression heuristics [13]. $\tau$-Argus was used to protect the 30 non-hierarchical magnitude statistical tables that were created using the randomized data generator as the microdata files, required by $\tau$-Argus for input, were not available to us for the other eight statistical tables. $\tau$-Argus provides the algorithms Hypercube, Modular, Network, Optimal, and Marginal for cell suppression. Each of these algorithms was used to protect the 30 statistical tables. As the Network algorithm failed to protect any of the statistical tables it has not been included in the results table. GA-group-K was allowed up to 12 h to execute, Modular and Optimal algorithms took between 5 min and 24 h to execute. Hypercube and Marginal took less than 5 min to execute. The costs (information loss) of suppressing the secondary cells are presented in Table V.

TABLE V
Cost of Suppressing the Secondary Cells for 30 of the Statistical Tables Using Different Algorithms

| Table Type | GA-Group-K | Hypercube | Modular | Optimal | Marginal |
|---|---|---|---|---|---|
| 200×5 | 1199 | 2377 | 999 | **997** | 39 580 |
| sens = 0.02 | 232 | 951 | **227** | – | 22 560 |
| zeros = 0.05 | **1276** | 1608 | – | – | 33 639 |
| | 1686 | 3131 | **1287** | – | 20 545 |
| | **1775** | 3299 | – | – | 37 165 |
| 200×5 | **550** | 2535 | – | – | 18 616 |
| sens = 0.10 | 853 | 1696 | **597** | - | 23 753 |
| zeros = 0.25 | 285 | 1316 | **64** | – | 39 550 |
| | **254** | 1368 | – | – | 21 273 |
| | 1193 | 1199 | **850** | – | 20 417 |
| 200×50 | **2955** | – | – | – | 54 280 |
| sens = 0.02 | **2939** | – | 86 984* | – | 54 923 |
| zeros = 0.05 | 2228 | – | **393*** | 2169 | 55 769 |
| | 3436 | – | **790** | – | 56 712 |
| | 2089 | – | **418** | – | 56 778 |
| 200×50 | **2820** | – | – | – | 54 566 |
| sens = 0.10 | **2161** | – | 2189 | – | 36 891 |
| zeros = 0.25 | **1970** | – | 2522 | – | 54 525 |
| | **2532** | – | – | – | 56 895 |
| | **2352** | – | - | – | 36 956 |
| 4000×10 | **2730** | – | – | – | 8033 |
| sens = 0.02 | **2908** | – | – | – | 8542 |
| zeros = 0.05 | **2838** | – | – | – | 8433 |
| | **2605** | – | – | – | 8292 |
| | **2572** | – | – | – | 8343 |
| 4000×10 | **2371** | – | – | – | 8411 |
| sens = 0.10 | **2715** | – | – | – | 8704 |
| zeros = 0.25 | **2646** | – | – | – | 8620 |
| | **2432** | – | – | – | 9039 |
| | **2436** | – | – | – | 8643 |

"–" indicates that the algorithm failed to protect the statistical table.
* indicates an anomaly in the results provided by $\tau$-Argus. The lowest cost of suppressing the secondary cells have been highlighted in bold.

TABLE VI
Number of Statistical Tables Protected by Each Algorithm by Table Size

| | Table Size | | |
|---|---|---|---|
| Algorithm | 200×5 | 200×50 | 4000×10 |
| GA-Group-K | 10 | 10 | 10 |
| Hypercube | 10 | 0 | 0 |
| Modular | 6 | 6 | 0 |
| Network | 0 | 0 | 0 |
| Optimal | 1 | 1 | 0 |
| Marginal | 10 | 10 | 10 |

### B. Analysis

The number of statistical tables protected by each algorithm, by table size, is shown in Table VI.

This shows that the only algorithms to protect all 30 statistical tables were the GA-group-K and Marginal. However, Marginal is the method of last resort as it works by suppressing margin (row and column) totals and this removes a much larger amount of information from the table than would suppressing non-totals. Hypercube protected all of the 200×5 statistical tables, but failed to protect the 200×50 and 4000×10 statistical tables. The Hypercube algorithm can theoretically handle much larger statistical tables than 200×5 and the limitation here is put down to this being an early implementation of the algorithm. Modular protected 60% of the 200×5 and 200×50 statistical tables but none of the 4000×10 statistical tables. This is expected as Modular partitions statistical tables prior to using the Optimal algorithm to protect them. The Optimal algorithm should find the optimal suppression pattern as it is an implementation of the Fischetti and Salazar [10], [11] MIP, however this is limited in the size of statistical table that can be protected as the optimal solution is NP-hard to find.

Unfortunately, the Network algorithm failed to protect any of the 30 statistical tables. The Network algorithm is, however, limited to finding suppression patterns for 2-D statistical tables only.

Of the 30 tables used in this comparison, 21 were best protected by GA-group-K, eight by Modular, and one by Optimal. Of the 20 tables with less than or equal to 10 000 cells 11 were best protected by GA-group-K, eight by Modular, and one by Optimal. The difference between the costs obtained by GA-group-K and Modular was not statistically significant. For the ten statistical tables that were protected by both Hypercube and GA-group-K, GA-group-K produced lower cost suppression patterns for all ten tables. For the ten statistical tables that were protected by both Marginal and Hypercube, Hypercube produced lower cost suppression patterns for all ten tables.

There are two anomalies in Table V where unexpected suppression pattern costs were reported. The reason for these anomalies is most likely due to the mathematical solver being pushed beyond its working bounds. For three of the $200 \times 50$ statistical tables the Modular algorithm performed very well with secondary cell costs of 393.0, 790.0, and 419.0. The reason for this is that no matter what permutation of the primary suppressed cells is used the LP model sometimes produces relatively poor results when compared with algorithms that protect all the primary suppressed cells at once. Solving this problem is ongoing work.

### C. Conclusion

GA-group-K has shown itself to be a reliable technique for protecting statistical tables when compared against current "state-of-the-art techniques." In all cases, it produced lower cost suppression patterns than the Hypercube and Marginal algorithms. Although it only produced lower cost suppression patterns than the Modular algorithm for three out of the 12 statistical tables, the later cannot protect the larger statistical tables and was only able to protect 60% of the smaller statistical tables. This indicates that in the future the GA-group-K algorithm will have an important role to play in the protection of published statistical tables.

## VII. Protecting Larger Statistical Tables

### A. Procedure

We have seen that for larger statistical tables the lower cost suppression patterns are achieved by protecting candidate initially exposed primary suppressed cells ($K$) in groups. Unfortunately, this approach cannot be directly applied to larger statistical tables. The problem lies with the preprocessing stage that identifies the members of $K$. Part of this preprocessing stage requires the running of two linear programs for each of the primary suppressed cells. To get around this problem a different subset of primary suppressed cells that we call $K_u$ are used, where $K_u \subseteq K$, these are the candidate initially exposed primary suppressed cells that can be identified using an unpicking algorithm. As an unpicking algorithm does not require the use of a mathematical solver it can handle large statistical tables: for example it took only 42 s to unpick a one million cell statistical table. Tests on a large variety of statistical tables showed that in approximately 99% of cases $K = K_u$. In the cases where $K$ was larger than $K_u$ it was so by, on average, only one or two primary cells. Therefore, the self-adaptive GA part of the algorithm uses a surrogate fitness function which is the linear programming model modified to protect members of $K_u$ in groups. Once the "best" permutation of the groups has been identified (i.e., after running the GA) they are again protected followed by protecting an extra group comprising the subset $P \setminus K_u$. This final step ensures that all primary suppressed cells are protected, for this the maximum number of iterations allowed for each LP was increased to ensure the full protection of each of the statistical tables. This approach was tested on 14 artificial 3-D statistical tables of varying size and was shown to have successfully protected them. In order to make the tables as realistic as possible a Poisson distribution was used to assign the number of contributors to each table cell and $-1/\log r$ was used to generate each contributor's contribution, where $r$ is a random number [0..1]. Other factors like the proportion of zero valued cell and primary cells were randomly assigned. Six of the statistical tables were hierarchical. The size and dimensions of the statistical tables are given in Table VII. The algorithm was allowed to run for up to 24 h for each statistical table being protected and the number of groups was reduced to 20.

The algorithm successfully protected all 14 statistical tables. However, even though the number of groups was reduced to

TABLE VII
Fourteen Large Statistical Tables

| Dimensions (Including Margin Totals) | Number of Cells | Number of Primary Cells | Size of Ku |
|---|---|---|---|
| 100 × 27 × 18 sens = 0.30, zeros = 0.49 | 48 600 | 14 807 | 263 |
| 100 × 21 × 24 sens = 0.21, zeros = 0.12 | 50 400 | 10 555 | 645 |
| 100 × 5 × 106(H) sens = 0.17, zeros = 0.41 | 53 000 | 8763 | 5558 |
| 100 × 112(H) × 4 sens = 0.08, zeros = 0.54 | 56 000 | 4544 | 4047 |
| 100 × 7 × 83 sens = 0.20, zeros = 0.43 | 58 100 | 11 517 | 4430 |
| 100 × 20 × 31 sens = 0.21, zeros = 0.13 | 62 000 | 13 164 | 678 |
| 100 × 6 × 112(H) sens = 0.03, zeros = 0.51 | 67 200 | 1972 | 1933 |
| 100 × 20 × 36 sens = 0.21, zeros = 0.13 | 72 000 | 15 422 | 734 |
| 100 × 76 × 10 sens = 0.18, zeros = 0.26 | 76 000 | 13 429 | 3629 |
| 100 × 4 × 212(H) sens = 0.21, zeros = 0.46 | 84 800 | 10 265 | 8859 |
| 100 × 11 × 90(H) sens = 0.19, zeros = 0.47 | 99 000 | 18 978 | 10 069 |
| 50 × 50 × 40 sens = 0.22, zeros = 0.13 | 100 000 | 21 850 | 259 |
| 100 × 158(H) × 9 sens = 0.20, zeros = 0.50 | 142 200 | 28 032 | 14 172 |
| 100 × 91 × 23 sens = 0.08, zeros = 0.36 | 209 300 | 16 008 | 3441 |

(H) indicates that the dimension is hierarchical.

TABLE VIII
NUMBER OF CALLS MADE TO THE FITNESS FUNCTION AND THE PERCENTAGE IMPROVEMENT IN THE SUPPRESSION PATTERN COST, BY THE GA, FOR THE 14 LARGE STATISTICAL TABLES

| Dimensions (Including Margin Totals) | Number of Cells | Number of Fitness Function Calls | Percentage Improvement Surrogate |
|---|---|---|---|
| 100 × 27 × 18 | 48 600 | 35 | 15.12 |
| 100 × 21 × 24 | 50 400 | 30 | 0.0 |
| 100 × 5 × 106(H) | 53 000 | 40 | 0.0159 |
| 100 × 112(H) × 4 | 56 000 | 60 | 10.79 |
| 100 × 7 × 83 | 58 100 | 35 | 10.5 |
| 100 × 20 × 31 | 62 000 | 25 | 0.0 |
| 100 × 6 × 112(H) | 67 200 | 25 | 2.46 |
| 100 × 20 × 36 | 72 000 | 20 | 0.0 |
| 100 × 76 × 10 | 76 000 | 20 | 0.0 |
| 100 × 4 × 212(H) | 84 800 | 40 | 0.0 |
| 100 × 11 × 90(H) | 99 000 | 25 | 0.36 |
| 50 × 50 × 40 | 100 000 | 15 | 0.0 |
| 100 × 158(H) × 9 | 142 200 | 20 | 0.0 |
| 100 × 91 × 23 | 209 300 | 195 | 0.15 |

(H) indicates that the dimension is hierarchical.

20 and it was allowed to run for up to 24 h it was only able to make a limited number of calls to the fitness function which in turn limited the improvement that it could make to the suppression patterns (see Table VIII).

### B. Analysis

The limited number of calls to the fitness function is because as the table has grown in size each fitness function call has taken longer to execute. The results shown in Table VIII, however, are hard to interpret. The relationship between the table size and the number of calls to the fitness function was not statistically significant. As the number of calls to the fitness function was limited, the algorithm was unable to search the fitness landscape thoroughly. This leads to only 7 out of the 14 statistical tables having their suppression patterns improved. The best improvement from the surrogate fitness function is 15.12% (see Table VIII). It is reasonable to expect that more improvements would have been seen given either more time or computational power.

### C. Conclusion

The introduction of a surrogate fitness function has allowed the algorithm to successfully protect 3-D non-hierarchical statistical tables with up to 209 300 cells and 3-D hierarchical statistical tables with up to 142 200 cells using the CLP mathematical solver from the open source COIN-OR framework. Although this a very good achievement in the field of cell suppression in statistical disclosure control the algorithm is still not able to effectively search its fitness landscape, for these very large statistical tables. This implies that further improvements to the algorithm should be obtained by simply speeding up each call to the fitness function and this can be easily achieved by using a more powerful commercial mathematical solver.

## VIII. CONCLUSION AND SUGGESTED FUTURE WORK

The use of a heuristic algorithm that combines a genetic algorithm and linear program has been shown to successfully protect large statistical tables. This algorithm was shown to outperform the combination of local search and linear programming. This algorithm has been shown to perform consistently on a variety of statistical tables of differing sizes. It has been shown to better protect larger tables than all the algorithms it was compared with in this paper. The use of a surrogate fitness function has allowed the algorithm to protect statistical tables with over 200 000 cells. In this paper, this algorithm has only been used to protect 2-D and 3-D hierarchical and non-hierarchical statistical tables; however, it can also protect statistical tables with more than three dimensions.

Future work to improve this algorithm will need to address the large amount of time that is required to execute the fitness function as this limits the ability of the algorithm to search its fitness landscape. Further improvement in the cost of the suppression patterns found using this algorithm may come from using a different grouping strategy. Currently, grouping is done by the cell weighting but the addition of grouping by row and column would add new, previously unseen, points on the fitness landscape.

The datasets used in this paper will be made available for download from http://www.cems.uwe.ac.uk/~jsmith/Statistical_Disclosure_Control.html.

## REFERENCES

[1] M. T. Almeida, G. Schutz, and F. D. Carvalho, "Cell suppression problem: A genetic-based approach," *Comput. Oper. Res.*, vol. 35, no. 5, pp. 1613–1623, 2008.
[2] T. Bäck, "Self adaptation in genetic algorithms," in *Proc. 1st Eur. Conf. Artif. Life: Toward a Practice Autonomous Syst.*, 1992, pp. 263–271.
[3] H.-G. Beyer, *Theory of Evolution Strategies*. Berlin, Germany: Springer, 2001.
[4] J. Castro, "Network flows heuristics for complementary cell suppression: An empirical evaluation and extensions," in *Proc. Inference Control Statist. Databases*, LNCS 2316. 2002, pp. 59–73.
[5] J. Castro, "Quadratic interior-point methods in statistical disclosure control," *Comput. Manage. Sci.*, vol. 2, no. 2, pp. 107–121, 2005.
[6] ILOG SA. (2006). *Cplex Mathematical Programming Optimizer* [Online]. Available: www.cplex.com
[7] COIN-OR. (2006). *COmputational INfrastructure for Operations Research* [Online]. Available: www.coin-or.org
[8] L. Davis, *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991.
[9] M. Fischetti and J. J. Salazar, "Experiments with controlled rounding for statistical disclosure control in tabular data with linear constraints," *J. Official Statist.*, vol. 14, no. 4, pp. 553–565, 1998.
[10] M. Fischetti and J. J. Salazar, "Models and algorithms for the 2-D cell suppression problem in statistical disclosure control," *Math. Program.*, vol. 84, no. 2, pp. 283–312, 1999.
[11] M. Fischetti and J. J. Salaza, "Solving the cell suppression problem on tabular data with linear constraints," *Manage. Sci.*, vol. 47, no. 7, pp. 1008–1027, 2001.
[12] M. Glickman and K. Sycara, "Reasons for premature convergence of self-adaptating mutation rates," in *Proc. CEC*, 2000, pp. 62–69.
[13] A. Hundepool, "Tau-Argus manager," private e-mail, Jul. 16, 2010.
[14] A. Hundepool, J. Domingo-Ferrer, L. Franconi, S. Giessing, R. Lenz, J. Naylor, E. S. Nordholt, G. Seri, and P-P. de Wolfe. (2007). *Handbook on Statistical Disclosure Control* [Online]. Available: http://neon.vb.cbs.nl/casc/SDC_Handbook.pdf
[15] J. P. Kelly, B. L. Golden, and A. A. Assad, "Cell suppression: Disclosure protection for sensitive tabular data," *Networks*, vol. 22, no. 4, pp. 397–417, 1992.

[16] M. Preuss and T. Bartz-Beielstein, "Sequential parameter optimization applied to self-adaptation for binary-coded evolutionary algorithms," in *Parameter Setting in Evolutionary Algorithms*, F. J. Lobo, C. F. Lima, and Z. Michalewicz, Eds. Berlin, Germany: Springer, 2007, pp. 91–120.
[17] D. A. Robertson, "Cell suppression at statistics Canada," in *Proc. 2nd Int. Conf. Statist. Confidentiality*, 1995, pp. 107–131.
[18] J. J. Salazar, C. Bycroft, and A. T. Staggemeier, "Controlled rounding implementation," in *Proc. Joint UNECE/Eurostat Work Session Statist. Data Confidentiality*, 2005 [Online]. Available: http://www.unece.org/stats/documents/2005.11.confidentiality.htm WP36.pdf
[19] H.-P. Schwefel, *Numerical Optimisation of Computer Models*. New York: Wiley, 1981.
[20] M. C. Serpell, A. R. Clark, J. Smith, and A. T. Staggemeier, "Preprocessing optimization applied to the classical integer programming model for statistical disclosure control," in *Proc. Privacy Statist. Databases*, 2008, pp. 24–36.
[21] M. C. Serpell and J. Smith, "Self-adaption of mutation operator and probability for permutation representations in genetic algorithms," *J. Evol. Comput.*, vol. 18, no. 3, pp. 491–514, 2010.
[22] J. E. Smith, "On replacement strategies in steady state evolutionary algorithms," *Evol. Comput.*, vol. 15, no. 1, pp. 29–59, 2007.
[23] J. E. Smith, A. R. Clark, and A. T. Staggemeier, "A genetic approach to statistical disclosure control," in *Proc. GECCO*, 2009, pp. 1625–1632.
[24] P. D. Surry and N. J. Radcliffe, "Inoculation to initialize evolutionary search," in *Proc. Evolutionary Computing AISB Workshop*, LNCS 1143. 1996, pp. 269–285.
[25] *Tau-Argus Statistical Disclosure Control Software* [Online]. Available: http://neon.vb.cbs.nl/CASC/TAU.html
[26] L. Willenborg and T. de Waal, *Elements of Statistical Disclosure Control* (Lecture Notes in Statistics). New York: Springer, 2000.

**Jim E. Smith** received his first degree in electrical sciences from Cambridge University, Cambridge, U.K., and received the Ph.D. degree in 1998.

He spent several years in industry before earning his Ph.D. He is currently a Reader in artificial intelligence with the University of the West of England, Bristol, U.K. His current research interests include the theory and application of intelligent systems that adapt their learning strategies in response to experience, and the interface between humans and adaptive intelligent systems.

**Alistair R. Clark** is currently a Reader in operational research (OR) and the Associate Head of the Department for Research and Scholarship, Department of Engineering Design and Mathematics, University of the West of England, Bristol, U.K. He is also interested in free (or nearly-free) software for OR, the mathematical modeling of statistical disclosure control, and the application of OR in emerging economies and developing countries. His current research interests include the development of models and methods for scheduling, making use of integer programming and metaheuristic approaches.

He is a fellow of the U.K. Operational Research Society and is the Editor-in-Chief of the society's practice-oriented journal *OR Insight* (to which he invites practitioner-oriented submissions of various kinds).

**Andrea T. Staggemeier** graduated in mathematics and information technology from the Universidade de Franca, São Paulo, Brazil, and received the M.Sc. degree in production engineering from Universidade Federal de Santa Maria, Santa Maria, Brazil, before coming to the U.K. She received the Ph.D. degree in computer science and mathematics from the University of the West of England, Bristol, U.K.

She taught graduate and masters students in software engineering, algorithms and operational research at the University of the West of England. Since 2005, she has been the Head of the Center for Statistical and Analytic Intelligence, IT Delivery Division, U.K. Office for National Statistics, Newport, U.K. In 2010, she joined a specialist consultancy directing analytic intelligence architecture solutions, and has been engaged with Lloyds Banking Group, London, U.K., and others. She has published/co-authored a number of articles in conference proceedings throughout her career (for further details please visit www.analyticintelligence.co.uk). Currently, she is preparing a book for publication entitled *Architecting for Analytics with SAS*, expected to be out in March 2012.

**Martin C. Serpell** received the Ph.D. degree in 2011.

He is currently a Research Associate with the University of the West of England, Bristol, U.K. He was with the software industry for many years before returning to academia.