Scaling up a Hybrid Genetic/Linear Programming Algorithm for Statistical Disclosure Control

M.C. Serpell
Department of Computer Science, University of the West of England, Coldharbour Lane, Bristol, BS16 1QY, UK
martin2.serpell@uwe.ac.uk

J.E. Smith
Department of Computer Science, University of the West of England, Coldharbour Lane, Bristol, BS16 1QY, UK
james.smith@uwe.ac.uk

A.R. Clark
Department of Mathematics and Statistics, University of the West of England, Coldharbour Lane, Bristol, BS16 1QY, UK
alistair.clark@uwe.ac.uk

A.T. Staggemeier
Centre for Statistical and Analytic Intelligence, Office for National Statistics, Newport, NP10 8XG, UK
Andrea.Staggemeier@ons.gsi.gov.uk

ABSTRACT

This paper looks at the real world problem of statistical disclosure control. National Statistics Agencies are required to publish detailed statistics and simultaneously guarantee the confidentiality of the contributors. When published statistical tables contain magnitude data such as turnover or health statistics the preferred method is to suppress the values of cells which may reveal confidential information. However suppressing these 'primary' cells alone will not guarantee protection due the presence of margin (row/column) totals and therefore other 'secondary' cells must also be suppressed. A previously developed algorithm that hybridizes linear programming with a genetic algorithm has been shown to protect tables with up to 40,000 cells, however Statistical Agencies are often required to protect tables with over 100,000 cells. This algorithm's performance highly depended on the choice of mutation operator so firstly this dependency was removed. As the algorithm is unable to protect larger tables due to the time it takes for its fitness function (a linear program) to execute a series of modifications have been applied. These modifications significantly reduced its execution time which in turn greatly extend the capabilities of the hybrid algorithm to the point that it can now protect tables with up to one million cells.

Categories and Subject Descriptors

G.1.6 [**Mathematics of Computing**]: Optimisation—*Constrained optimization*; G.2.3 [**Mathematics of Computing**]: Optimisation—*Applications*

GECCO'11, July 12–16, 2011, Dublin, Ireland.

General Terms

Algorithms, Performance, Economics, Experimentation, Security

Keywords

Statistical Disclosure Control, Cell Suppression Problem, Genetic Algorithms, Mathematical Programming

1. INTRODUCTION

National Statistics Agencies publish useful statistical reports relating to their nation, however they must ensure that the confidentiality of those who contributed the data to these reports is not compromised. Protecting this confidentiality is known as Statistical Disclosure Control and it contains many different methods. One of which is Cell Suppression where table cells that break confidentiality are suppressed (not published), these cells are referred to as primary suppressed cells. In order to guarantee their protection other cells must be suppressed to create a suppression pattern, these are referred to as secondary suppressed cells. The problem of creating suppression patterns that both protect all the primary suppressed cells and minimise the amount of information that is lost (information loss) from the statistical table is known as the cell suppression problem. In the ideal situation the statistical agency would be able to create an optimal cell suppression pattern that not only suppresses the minimum amount of information but also guarantees to protect all confidential information in the statistical table for any table size. This however has been shown to be NP-Hard [13]. The use of non-mathematical programming techniques is limited as they do not guarantee to adequately protect the confidential information in the statistical table.

The protection of 2-dimensional statistical tables has been well catered for. Fischetti and Salazar [8] and Castro [4] developed a integer programming (IP) network flow model that can optimally protect 2-dimensional non-hierarchical tables with up to 500,000 cells and near optimally protect 2-dimensional hierarchical tables with up to 250,000 cells (hierarchical tables contain sub-totals within the body of the table - they have a far more complex structure). Protecting

statistical tables with more than two dimensions has been more problematic. An IP model initially proposed by Kelly *et al.* [13] and reformulated by Fischetti and Salazar [8] has been used to optimally protect 2^+-dimensional hierarchical tables with 10,000 cells however this technique can be unreliable due to the limitations of the mathematical solver. A modular approach [6] that partitions tables prior to their protection more reliably protects 2^+-dimensional hierarchical tables with 10,000 cells but the price for doing this is slight over-suppression.

1.1 The Incremental LP Model

Typically the statistical data that is collected is published as 2-dimensional tables of aggregates, however these are themselves generated from a multi-dimensional dataset. For example the BRES reports published by the Office for National Statistics (ONS) in the UK are published as a set of 2-dimensional statistical tables. However the source BRES data, from which these tables are generated, is a 4-dimensional hierarchical table of aggregates indexed by industry type (SIC), geography, part/full-time employment and public/ private employment with over 1,000,000 cells. Therefore any protection applied to these 2-dimensional tables individually could be compromised by the fact the cells in the tables are linked. Protecting the source data as one large multi-dimensional table would guarantee protection but this would be computationally difficult.

An incremental linear programming (LP) heuristic model that protects cells one at a time was developed by Kelly et al [13] and has been shown to reliably protect 2^+-dimensional hierarchical tables with up to 40,000 cells. This model however will not scale and over-suppresses, that is to say it suppresses more cells than it needs to. Let us consider a statistical table containing n cells each with a cell value a_i where $i = 1, ..., n$ and m constraints that describe the relationships between the cells. The cell value a_i is known to lie within the bounds lb_i to ub_i and the lower and upper protection limits (lpl_i and upl_i) are provided by the National Statistics Agency. In the incremental LP the relative lower and upper bounds, $LB_i = a_i - lb_i$ and $UB_i = ub_i - a_i$, are used instead of lb_i and ub_i. The information loss associated with suppressing a cell is given by w_i, however c_i is used in this model where $c_i = 0$ if cell i is already suppressed and $c_i = w_i$ if cell i is not suppressed. M is an array that associates cell i with each constraint equation j (row or column), $j = 1, ..., m$. If a cell i is in a constraint equation j then $M_{ij} = 1$, otherwise $M_{ij} = 0$. The variables y_i^- and y_i^+ are the lower and upper uncertainties provided by cell i. P is the set of primary suppressed cells, these are the cells that must be protected to guarantee confidentiality. For each $p \in P$ a suppression pattern that safeguards the upper and lower protection levels is found. After the suppression pattern for the upper protection level is found for p any cell $i \notin P$ that has a value $y_i^- + y_i^+ > 0$ and $a_i > 0$ has it's weighting c_i set to 0. This procedure is also carried out for the lower protection levels. After this process has been repeated for all p the cells $i \notin P$ that have had their weightings c_i set to 0 become the secondary suppressed cells.

This model solves the Cell Suppression Problem for each primary suppressed cell in turn. Clark and Smith [5] found that the order in which each primary suppressed cell was protected significantly affected the quality of the suppression pattern produced. To overcome this problem they used a Genetic Algorithm (GA) to find a "best" permutation with which to protect the primary suppressed cells using this model. 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 incremental LP model which as Equation 1 shows necessarily works by minimising 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 incremental LP model followed by the cost calculation.

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

subject to

$$M(y^+ - y^-) = 0 \quad (2)$$
$$0 \le y_i^+ \le UB_i \quad for\ i = 1, ..., n \quad (3)$$
$$0 \le y_i^- \le LB_i \quad for\ i = 1, ..., n \quad (4)$$

for each $p \in Pin$ *turn* :
upper protection level

$$y_p^- = 0 \quad (5)$$
$$y_p^+ = upl_p - a_p \quad (6)$$

lower protection level

$$y_p^- = a_p - lpl_p \quad (7)$$
$$y_p^+ = 0 \quad (8)$$

Figure 1: The Incremental LP Model of Kelly et al (1992) for the Cell Suppression Problem. This is used as part of the fitness function.

1.2 Improvements to Previously Reported GA

The decision of what values to assign to the parameters that control a GA have a great impact on its performance. The self-adaption of mutation parameters has been proved successful in the continuous domain [3] [15] and for binary combinatorial problems [2] [9] [14] [17]. Removing the need for the GA user to find optimal settings for the mutation operator and/or rate saves them a considerable amount time. It can also reduce the risk of poor performance arising from inappropriate settings when extensive operator tuning is not possible or practical, and so can improve performance over a wider range of problem instances. The GA associated with this class of problem has many possible mutation operators to choose from, each with an associated parameter. The choice of mutation operator has previously been shown to be critical and problem dependent [16]. The self-adaption of mutation operator and/or rate has already been proved successful for permutation representations [16]. An added benefit of using self-adaptation is that it reduces the chances that the GA will get stuck in a local optima. To demonstrate this comparison of the performance of a GA that self-adapts

the mutation operator and four that use the fixed mutation operators, swap, insert, scramble and inversion [7], was carried out. Fig. 2 shows the change in suppression pattern costs as the genetic algorithms search for the 'best' suppression pattern for a 2-dimensional statistical table with one hierarchical dimension and 748 cells of which 105 are primary suppressed cells.

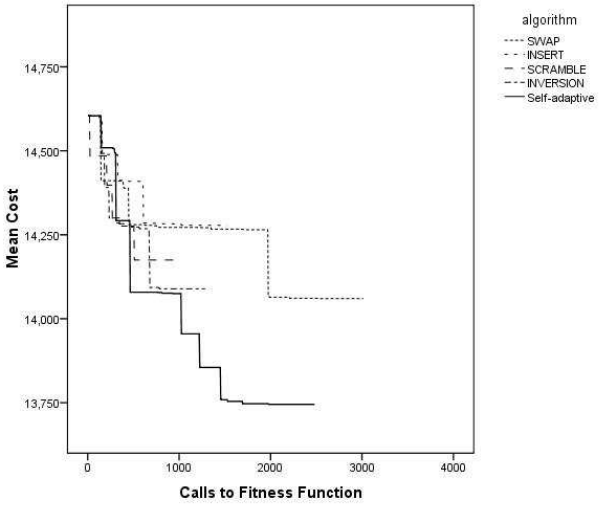

Figure 2: Suppression pattern cost by the number of calls to the fitness function for the five different genetic algorithms.

Although the problem of over-suppression of the incremental LP model can be reduced by the use of a GA the problem of scalability remains. To protect a statistical table using the incremental LP model two LPs need to be solved for each primary cell, one to guarantee the fulfilment of the lower protection level requirement and one to guarantee the fulfilment of the upper protection level requirement. As the size of the tables increases so does the number of primary cells that need to be protected and hence so does the number of LPs that need to be solved. Also as the table size grows so too does the amount of time it takes to solve each of the LPs. Hence, combining these two effects, we can see that the time taken to run the incremental LP model grows polynomially with the size of the table. It is this rapid growth in execution time that limits the size of statistical table that the incremental LP model can protect. This problem becomes so acute, as the tables become larger, that the only way to protect them is to reformulate the model.

The purpose of this paper is to describe how the incremental LP model was reformulated to allow it to protect statistical tables with up to one million cells. Section 2 describes the experimental framework and datasets used for this study. Section 3 describes the series of reformulations that were applied to the incremental LP model to make it applicable to solving real world problems. Section 4 contains our conclusions and suggested future work.

2. METHODOLOGY

The algorithms developed in this paper were implemented in the C++ language using the open source COIN-OR framework and in particular the CLP solver [1]. Experiments were run on a 2.33GHz PC running Windows XP. The datasets used in this work, sections 3.1 to 3.6, were artificial multi-dimensional statistical tables of varying size produced using a dataset generator provided by ONS. In order to make these datasets as realistic as possible the generator used a Poisson distribution to assign the number of contributors to each table cell and $-1/\log r$ to generate each contributor's contribution, where r is a random number [0..1]. Other factors like the proportion of zero valued cells and primary cells were randomly assigned. Using the ONS dataset generator allowed the large number of datasets required for statistical analysis to be created. It also removed the problem of real datasets being allowed off-site. The comparison in section 3.7 used datasets provided by ONS. The results from these experiments were analysed using the statistical package SPSS. The Friedman's ANOVA test has been used to see if there is a significant difference in the performance of the different algorithms. The Wilcoxon's Signed Rank test was used when a comparison of the performance of just two of the algorithms was required.

3. REFORMULATING THE MODEL

3.1 Protect a Subset of the Primary Cells

It has long been anecdotally known that some primary suppressed cells (P) are sufficiently protected by other primary suppressed cells in a statistical table prior to it being protected and that they can therefore be removed from the Cell Suppression Problem. The minimum set of primary suppressed cells that need to be protected were identified and named as *initially exposed primary suppressed cells* (I). For practical reasons the set of *candidate initially exposed primary suppressed cells* (K) are used instead of I, as they easier to identify. As $\mid I \mid \leq \mid K \mid \leq \mid P \mid$ is always true and $\mid I \mid \ll \mid P \mid$ is often true (in practice we have examined tables with one million cells and $|P| > 10,000$ but $|K| = 0$) it is more efficient to protect K instead of P. By protecting members of K instead of P the number of LPs that need to be executed is reduced and this in turn allows statistical tables to be protected quicker and/or larger tables to be protected.

3.2 Add Grouping to the Model

The incremental LP model protects candidate initially exposed primary suppressed cells (K) one at a time and this limits the size of the statistical table that can be protected due to the number of LPs that need to be solved. This limitation however can be reduced if instead of protecting these cells one at a time they are protected in groups. To protect these cells in groups the incremental LP model must be reformulated. If G is the set of groups that partition K and $g \in G$ then all the cells in g must have their upper protection levels protected together and their lower protection levels protected together. When grouping is applied to the incremental LP model the grouped LP model shown in Fig. 3 is obtained.

In order to place cells in K into groups the following rule must be adhered to. Let P be the set of primary suppressed cells and K be the set of candidate initially exposed primary suppressed cells. Let J be the set of constraint equations describing the rows and columns of a statistical table and $j \in J$ be the set of cells in a particular row or column j. Let G be the set of groups that contain all the cells in K and

$$min \ \sum_{i=1}^{n} c_i(y_i^+ + y_i^-) \tag{9}$$

subject to

$$M(y^+ - y^-) = 0 \tag{10}$$
$$0 \leq y_i^+ \leq UB_i \quad for\ i = 1, ..., n \tag{11}$$
$$0 \leq y_i^- \leq LB_i \quad for\ i = 1, ..., n \tag{12}$$

for each $g \in G$ *in turn* :
upper protection level

$$y_p^- = 0 \quad \forall p \in g \tag{13}$$
$$y_p^+ \geq upl_p - a_p \quad \forall p \in g \tag{14}$$

lower protection level

$$y_p^- \geq a_p - lpl_p \quad \forall p \in g \tag{15}$$
$$y_p^+ = 0 \quad \forall p \in g \tag{16}$$

Figure 3: The Grouped LP Model.

$g \in G$ be a subset of K. Then to ensure that the upper protection levels can be satisfied without causing unnecessary over-protection Equation 17 must be satisfied.

$$\sum_{i \in (j \cap g)} (upl_i - a_i) \leq \frac{\sum_{i \in (j \cap P)} (upl_i - a_i)}{2} \tag{17}$$

To ensure that the lower protection levels can be satisfied without causing unnecessary over-protection Equation 18 must be satisfied.

$$\sum_{i \in (j \cap g)} (a_i - lpl_i) \leq \frac{\sum_{i \in (j \cap P)} (a_i - lpl_i)}{2} \tag{18}$$

Together Equation 17 and Equation 18 ensure that, when there is more than one primary cell in a row or column, only half or less of the uncertainty required to protect the primary cells in that row or column can be attributed to a given group. To ensure that there is no unnecessary over-protection Equation 19 must be satisfied.

$$\forall j \in J, \forall g \in G \cdot (\mid j \cap g \mid \leq 1) \vee (UR \wedge LR) \tag{19}$$

Where UR is the upper protection rule given by Equation 17 and LR is the lower protection rule given by Equation 18. Equation 19 ensures that each group, g, contains either zero or one members of K from row or column j or only those members of K (in j) whose required uncertainty can be offset by the remaining members of P (in j) that are not in g.

The candidate initially exposed primary suppressed cells (K) were placed in groups in increasing protection level order as this has previously been shown to be a good order when protecting cells one at a time [5]. Tests have shown that for the smaller statistical tables ($< 30,000$ cells) grouping the cells in K prior to protecting them did not necessarily lead to lower cost suppression patterns as it reduces the number of points on the fitness landscape. However for the larger statistical tables grouping them prior to protecting them is better than protecting them one at a time. This is because as the size of the statistical table increases so does the number of LP's that need to be run to find the fitness of a single permutation of the cells in K. Grouping limits the number of LP's required. A fixed number of groups mean a fixed number of LP's to find the fitness of each permutation. This means that, in general, the fitness of each permutation can be found quicker which 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 $> 30,000$ cells.

3.3 Notes on Generalising and Landscapes

A common GA approach would be to use a rapid surrogate fitness function [12] or to use fitness inheritance [11] [18]. However for the cell suppression problem these approaches can't be used because of the problem of infeasibility. To guarantee feasible solutions a mathematical model needs to be used and these common GA approaches do not deal well with the constraints that form an integral part of the mathematical model, so the computationally expensive mathematical model must still form part of the fitness function. Therefore the solution used here is to reduce the size of the fitness landscape. Obviously the way this is done is important therefore the method chosen and described in 3.2 is designed to reduce over-protection and so ensure that the set of points sampled by the grouping landscape is among the better points on the full landscape.

The integer programming (IP) model for cell suppression [13] "sees" the full landscape of 2^n different suppression patterns. If there are $|P|$ primaries then the incremental version of the IP (all incremental versions are greedy constructive heuristic) samples $|P|!$ of the solutions, there are no guarantees that this subset contains the global optimum. The linear relaxation, incremental LP model, which is the one used in this paper, also sees $|P|!$ solutions but these may not be the same subset as the incremental IP model. The grouping function with $|G|$ groups only samples $|G|!$ points, but these are still all within the original space. However these may not be within either of the two subsets identified above. Also there are$|G|^{|P|}/|G|!$ different groupings to choose from. If we had time we could examine all of the possible groupings, however this is unlikely and so we need a principled way of sampling from these or of picking one from these.

Grouping is definitely faster in searching the fitness landscape as evaluating each point on the landscape only requires $2|G|$ LPs to be run rather than $2|P|$. Without grouping there are more basins of attraction and therefore less chance of sampling each in the initial population. Also on average every step means sampling more points (since the neighbourhoods are bigger) and more scope for the basins to be deeper, so even if the initial population for the incremental LP model had a point in the "optimal" basin it still might not have time to reach the bottom. However there is a trade-off between doing a better search in a smaller sample versus the likelihood that the smaller sample will contain "good" solutions. Equations 17 - 19 show the formulation of fast grouping rules that ensure the grouping is in fact biased away from poor solutions, therefore allowing a better search of a smaller sample of "good" solutions.

3.4 Using a Surrogate Fitness Function

We have seen that for larger statistical tables the lower cost suppression patterns are achieved by protecting cells in K in groups. Unfortunately this approach cannot be directly applied to larger statistical tables, those with $> 40,000$ cells. The problem lies with the pre-processing stage that identifies the members of K. Part of this pre-processing 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 seconds 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 incremental LP model modified to protect members of K_u in groups. Once the 'best' permutation of the groups has been identified they are again protected followed by protecting an extra set of groups comprising the subset $P \setminus K_u$, this final step ensures that all primary suppressed cells are protected. Using this approach 3-dimensional non-hierarchical tables with up to 209,300 cells and 3-dimensional hierarchical tables with up to 142,200 cells were successfully protected.

3.5 Reducing the Number of Groups

An experiment was carried out to determine whether reducing the number of groups that the primary cells are assigned to will allow larger tables to be protected. The algorithm described in Section 3.4 was modified to put the primary cells in K_u into two groups and to give it a larger time limit of 50 hours. This algorithm was then used to protect four different 3-dimensional non-hierarchical statistical tables, each with approximately 250,000 cells, see Table 1.

Dimensions (incl. margin totals)	Number of Cells	Number of Primary Cells	Size of K_u
100 × 33 × 78	257,400	44,617	960
100 × 25 × 96	240,000	59,853	1,135
100 × 73 × 35	255,500	24,981	1,797
100 × 66 × 38	250,800	86,250	403

Table 1: The four statistical tables with approximately 250,000 cells.

Due to the grouping rules the cells in K_u were placed into three groups for two of the tables and four groups for two of the tables. Each of the four tables were successfully protected, however the algorithm over-ran its allotted time of 50 hours. During a run time of approximately 120 hours 16 calls were made to the fitness function meaning that it took approximately 7.5 hours to protect each permutation of groups for each statistical table. This has shown that reducing the number of groups that the primary cells are assigned to does allow larger tables to be protected. However the time taken to protect these tables is excessive and the benefits of landscape search and self-adaption have been lost. To get around this problem further time saving is required.

As part of our pre-processing an 'unpicker' algorithm is used to find exposed primary cells, E_u. For the four statistical tables shown in Table 1 the number of primary cells that failed the lower protection level requirement only, upper protection level requirement only and both were recorded, see Table 2. From Table 2 we can see that most of the exposed

Dimensions (incl. margin totals)	Lower only	Upper only	Both
100 × 33 × 78	0	887	76
100 × 25 × 96	0	1,086	51
100 × 73 × 35	7	1,070	733
100 × 66 × 38	0	403	0

Table 2: Number of primary cells failing protection.

cells, E_u, failed the upper protection level requirement. Of those members of E_u that failed the lower protection level requirement the vast majority also failed the upper protection level requirement. If, in the surrogate function, we only protected those members of E_u that failed the upper protection level requirement then many of the members of E_u that failed the lower protection level requirement would also be protected. This would still leave many of the statistical table's under-protected, but as this is a surrogate fitness function it is only the rank order that is of concern. Once the 'best' order has been found full protection is then applied to the statistical table as before. This modification to the surrogate fitness function greatly reduces the number of LPs that need to be solved and hence significantly reduces the time required to protect the tables.

To find the 'best' permutation we currently run a GA with a pool size of ten. However when there are three or less groups it is more efficient to examine all permutations of the groups of cells in K_u. Therefore to find the 'best' permutation this strategy is now followed. The combination of these changes reduced the time required to protect the four statistical tables containing approximately 250,000 cells to approximately 60 hours and it allowed for the protection of much larger tables. The reformulated model was used to protect a range of larger and more complex statistical tables. The size, dimensions and the number of primary cells has been recorded in Table 3. The largest table protected was a 3-dimensional non-hierarchical table with one million cells. The largest 3-dimensional hierarchical table that was successfully protected had 532,400 cells. The largest 4-dimensional non-hierarchical table that was successfully protected had 264,600 cells. As the table complexity increases then the size of the table that can be protected decreases.

3.6 Speeding up the Model

There are circumstances in which we know that margin totals (and sub-totals) in a published statistical table will always need to be suppressed. For constraint equations that contain at least one primary suppressed cell and have a margin total (or sub-total) that is not a primary suppressed cell. Let J be the set of cells in a constraint equation and x_i be the nominal value of each of the cells in J. Let m be the margin total or sub-total in J. Let p be the primary cell

Dimensions (including margin totals)	Number of Cells	Number of Primary Cells
51(H) × 77(H) × 40	157,080	16,224
88(H) × 50 × 40	176,000	24,076
100 × 66 × 38	250,800	86,250
21 × 35 × 60 × 6	264,600	44,934
489 × 1026	501,714	68,445
243 × 2132	518,076	109,855
100 × 242(H) × 22(H)	532,400	59,416
100 × 56 × 97	543,200	164,239
90 × 90 × 90	729,000	164,927
100 × 100 × 100	1,000,000	227,590

Table 3: The statistical tables with up to one million cells. (H) indicates that the dimension is hierarchical.

with the largest value of $max(lpl_p, upl_p)$ in the set $J \setminus m$, where lpl_p is the lower protection level and upl_p is the upper protection level for cell p. Then m must be suppressed if Equation 20 is true.

$$(x_m < x_p + lpl_p) \; or \; (x_m < x_p + upl_p) \tag{20}$$

This test provides an easy means of finding margin totals (or sub-totals) that must be suppressed in a published statistical table. These preselected margin totals are temporarily treated as primary suppressed cells. Calculating the required upper protection limit for these preselected margin totals however would allow a check to be carried out to see if the grand margin total needs to be suppressed. This would apply also to the suppression of grand sub-totals. The required lower and upper protection levels are calculated as follows.

$$lpl_m = lpl_p - \sum_{i \in J \setminus p \setminus m} x_i \tag{21}$$

$$upl_m = upl_p - \sum_{i \in J \setminus p \setminus m} x_i \tag{22}$$

Where J is the set of cells in a constraint equation, x_i is the nominal value of each of the cells in J, m is the margin total or sub-total in J, p is the primary cell with the largest value of $max(lpl_p, upl_p)$ in the set $J \setminus m$, lpl_p is the lower protection level and upl_p is the upper protection level for cell p. This process can be repeated until no new cells can be suppressed. The newly suppressed margin totals are then turned back into ordinary cells but their cell weightings, w_i, are reduced to zero ensuring that they will be selected as secondary suppressed cells when the table is protected. It is this modified statistical table that is then protected. If the cells were left as primary cells then testing has shown that this would increase the execution time of the LPs.

Similarly it is also possible to preselect some cells with large values to be secondary suppressed cells. Again imagine a constraint equation with at least one primary cell that requires a large protection level and an unsuppressed margin total, in such a case though the margin total does not need to be suppressed. Let p be the primary cell with the largest value of $max(lpl_p, upl_p)$ and m the margin total. The largest non-primary, n, needs to be suppressed if Equation 23 is true.

$$(x_m - x_n < x_p + lpl_p) \; or \; (x_m - x_n < x_p + upl_p) \tag{23}$$

Up until now grouping has mainly been based on the size of the protection level requirement of the cells being placed into the groups, primary cells with small protection level requirements being grouped together and primary cells with large protection level requirements being grouped together. However a better suppression pattern may be generated if cells in the same row/column are protected together, where possible primary cells in the same row/column will be placed in the same group.

The pre-selection of secondary cells and grouping changes to the model were tested against twenty three statistical tables with between 40,000 and 260,000 cells, the results are shown in Table 4. Each algorithm attempted to place all the members of K_u into just two groups. Each algorithm was given a maximum of 50 hours (180,000 seconds) to run although it was allowed to over-run if initialling the parent pool took longer.

Statistical testing showed no significant difference in the suppression pattern costs. This was unexpected as it was thought that grouping cells by row/column would lead to lower cost suppression patterns. However it did identify a significant difference in the time taken to find the 'best' suppression pattern. In particular the combinations that grouped cells by row/column on average found the 'best' suppression pattern fastest, on average finding it 10.61% faster than the equivalent algorithm that grouped cells by the size of their required protection levels. This speed up was due to grouping by row/column producing fewer groups than grouping by required protection levels. This in turn meant that less LPs needed to be solved to find the 'best' suppression pattern. Statistical tests showed that grouping by row/column was on average quicker than grouping by the size of the cells required protection levels (95% confidence when only the margin total was preselected and 95.4% when both the margin total and large cell values were preselected). Hence grouping by row/column should always be used. The pre-selection of margin totals for secondary suppression was found to shorten the time taken to find the 'best' suppression pattern by 1.25%. Similarly the pre-selection of the large valued cells shortened the time by a further 1.38%. Of the forty six comparisons between the times taken to find the 'best' suppression patterns, thirty four were quicker when both the margin totals and large valued cells were preselected and twelve when the margin totals only were preselected. Hence, with 99.4% confidence, pre-selection of both the margin totals and large valued cells for secondary suppression should be used to speed up the algorithm.

3.7 A Comparison With Other Algorithms

The performance of the algorithm was compared against that of Hypercube, Modular, Optimal and Marginal which form part of the statistical disclosure control tool, τ-Argus [10]. Each algorithm was used to protect thirty non-hierarchical 2-dimensional magnitude statistical tables provided by ONS.

The algorithm and Marginal protected all thirty tables, Hypercube ten, Modular twelve and Optimal two. The difference between the suppression pattern costs found by the algorithm and Modular were not statistically significant. The algorithm was significantly better than Hypercube and Marginal.

4. CONCLUSIONS AND FUTURE WORK

The hybrid algorithm developed by Clark and Smith [5] used the incremental LP model as the fitness function for a GA. This reduced the problem of over-suppression of the incremental LP model however it did not improve its scalability. In this paper a series of reformulations have been reported that have allowed the algorithm to successfully protect statistical tables with up to one million cells. This reformulated model is currently being tested on the BRES data described in Section 1. Further improvements in the time required to protect the tables can easily be achieved by using a commercial mathematical solver and a faster PC. However to protect still larger tables will require further reformulation.

Future improvements to this algorithm are likely to come from improving the GAs ability to search the fitness landscape and further reducing the time taken to run the fitness function i.e. the reformulated incremental LP model (grouped LP model). Improvements to the GA may come by better tuning it to this particular problem. Though the choice of mutation operator and rate are selected via self-adaption no effort has been put into selecting the 'best' crossover operator. Future work may include the tuning of this and other GA parameters. Future work on the grouped LP model should further reduce its over-suppression and the time it takes to execute. Reducing its time to execute will allow for a more thorough search of the fitness landscape which in turn should lead to lower cost suppression patterns. Useful knowledge may be gained by examining all the permutations of the groups and it may be productive to consider using a GA to do the grouping.

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

5. ACKNOWLEDGMENTS

This work was funded by the Office of National Statistics (ONS) and the EPSRC Maths CASE project 019/007. The authors wish to thank colleagues at ONS for their useful insights.

6. REFERENCES

[1] Computational infrastructure for operations research, 2006. www.coin-or.org.

[2] T. Bäck. Self adaptation in genetic algorithms. In F. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems: Proceedings of the 1st European Conference on Artificial Life*, pages 263–271. MIT Press, Cambridge, MA, 1992.

[3] H.-G. Beyer. *The Theory of Evolution Strategies*. Springer, Berlin, Heidelberg, New York, 2001.

[4] J. Castro. Network flows heuristics for complementary cell suppression: An empirical evaluation and extensions. In J. Domingo-Ferrer, editor, *Inference Control in Statistical Databases*, volume 2316 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2002.

[5] A. Clark and J. Smith. Improvements to cell suppression in statistical disclosure control. Technical report, University of the West of England, 2006. End-of-Project Report for the Office for National Statistics (ONS).

[6] P.-P. de Wolf. Hitas: A heuristic approach to cell suppression in hierarchical tables. In J. Domingo-Ferrer, editor, *Inference Control in Statistical Databases*, volume 2316 of *Lecture Notes in Computer Science*, pages 81–98. Springer Berlin / Heidelberg, 2002.

[7] A. Eiben and J. Smith. *Introduction to Evolutionary Computation*. Springer, 2003.

[8] M. Fischetti and J. Salazar-González. Models and algorithms for the 2-dimensional cell suppression problem in statistical disclosure control. *Mathematical Programming*, 84(2):283–312, 1999.

[9] M. Glickman and K. Sycara. Reasons for premature convergence of self-adaptating mutation rates. In *2000 Congress on Evolutionary Computation (CEC'2000)*, pages 62–69. IEEE Press, Piscataway, NJ, 2000.

[10] A. Hundpool. τ-argus statistical disclosure control software, 2004. http://neon.vb.cbs.nl/CASC/tau.html.

[11] J. hung Chen, D. E. Goldberg, S. ying Ho, and K. Sastry. Fitness inheritance in multiobjective optimization. 2002.

[12] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 9(1):3–12, 2005.

[13] J. Kelly, B. Golden, and A. Assad. Cell suppression: Disclosure protection for sensitive tabular data. *Networks*, 22(4):397–417, 1992.

[14] M. Preuss and T. Bartz-Beielstein. Sequential parameter optimisation applied to self-adaptation for binary-coded evolutionary algorithms. In L. et al, editor, *Parameter Setting in Evolutionary Algorithms*, pages 91–120. Springer, 2007.

[15] H.-P. Schwefel. *Numerical Optimisation of Computer Models*. Wiley, New York, 1981.

[16] M. Serpell and J. Smith. Self-adaption of mutation operator and probability for permutation representations in genetic algorithms. *Evolutionary Computation*, 18(3):491–514, 2010.

[17] J. Smith and T. Fogarty. Self adaptation of mutation rates in a steady state genetic algorithm. In *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*, pages 318–323. IEEE Press, Piscataway, NJ, 1996.

[18] R. E. Smith, B. A. Dike, and S. A. Stegmann. Fitness inheritance in genetic algorithms. In *Proceedings of the 1995 ACM symposium on Applied computing*, SAC '95, pages 345–350, New York, NY, USA, 1995. ACM.

Table 4a: Suppression Pattern Costs (primary + secondary costs)

Dimensions (including margin totals)	Order by Increasing Protection Level			Order by Row/Column	
	No pre-selection	Marginals Only	Marginals and Large	Marginals Only	Marginals and Large
100 × 112(H) × 4 (44,800 cells)	14,200,364	14,634,496	**13,347,366**	13,002,724	13,719,646
100 × 27 × 18 (48,600 cells)	1,206,282	890,307	888,160	**817,250**	820,074
100 × 21 × 24 (50,400 cells)	1,004,490	1,066,570	1,095,378	**978,857**	1,011,579
100 × 5 × 106(H) (53,000 cells)	10,514,077	**8,076,508**	9,254,447	9,707,584	9,721,753
100 × 7 × 83 (58,100 cells)	1,153,866	1,124,224	1,071,203	**1,003,650**	1,118,858
100 × 20 × 31 (62,000 cells)	509,375	403,207	**401,323**	489,424	488,345
100 × 6 × 112(H) (67,200 cells)	**8,664,779**	9,490,609	8,846,818	9,189,121	9,475,840
100 × 20 × 36 (72,000 cells)	524,462	457,476	**450,346**	498,571	494,505
100 × 76 × 10 (76,000 cells)	2,233,823	1,776,019	1,770,470	1,775,372	**1,752,982**
100 × 4 × 212(H) (84,800 cells)	4,286,570	**2,919,503**	5,463,395	7,186,450	6,724,912
100 × 85 × 10 (85,000 cells)	965,846	1,027,987	1,131,146	**924,343**	980,024
100 × 11 × 90(H) (99,000 cells)	1,130,691	1,100,494	1,046,512	**875,815**	1,062,524
50 × 50 × 40 (100,000 cells)	545,253	**464,305**	499,235	467,139	480,445
50 × 50 × 40 (100,000 cells)	398,934	402,845	443,264	406,300	**368,961**
100 × 43 × 27 (116,100 cells)	681,301	729,314	**629,563**	635,362	662,598
100 × 158(H) × 9 (142,200 cells)	**3,013,141**	6,508,584	4,245,042	5,646,202	3,795,041
100 × 30 × 51 (153,000 cells)	1,473,847	1,197,542	1,242,531	1,222,712	**1,176,491**
100 × 91 × 23 (209,300 cells)	1,311,253	1,336,358	**1,283,137**	1,477,550	1,440,611
100 × 25 × 96 (240,000 cells)	1,205,047	905,265	938,788	816,237	**791,300**
100 × 66 × 38 (250,800 cells)	4,020,419	4,885,828	**2,605,882**	5,152,570	4,211,123
100 × 55 × 46 (253,000 cells)	685,567	622,599	628,427	**597,919**	628,164
100 × 73 × 35 (255,500 cells)	1,313,824	893,249	**889,924**	1,007,851	1,001,572
100 × 33 × 78 (257,400 cells)	1,153,866	1,124,224	1,071,203	**1,003,650**	1,118,858

Table 4b: CPU Time (seconds)

Dimensions including margin totals	Order by Increasing Protection Level			Order by Row/Column	
	No pre-selection	Marginals Only	Marginals and Large	Marginals Only	Marginals and Large
100 × 112H × 4 (44,800 cells)	23,326	**19,945**	23,606	37,621	32,963
100 × 27 × 18 (48,600 cells)	75,632	69,084	**68,747**	81,112	78,299
100 × 21 × 24 (50,400 cells)	429,165	421,867	408,658	185,883	**176,021**
100 × 5 × 106H (53,000 cells)	142,507	138,645	133,836	128,252	**120,023**
100 × 7 × 83 (58,100 cells)	337,506	327,831	**286,579**	339,577	327,393
100 × 20 × 31 (62,000 cells)	24,365	25,349	26,626	25,026	**21,051**
100 × 6 × 112H (67,200 cells)	59,645	58,268	56,660	**50,280**	71,035
100 × 20 × 36 (72,000 cells)	51,829	49,626	48,671	**38,253**	40,255
100 × 76 × 10 (76,000 cells)	22,860	21,277	**20,184**	20,752	25,259
100 × 4 × 212H (84,800 cells)	24,338	17,014	**16,194**	21,479	19,499
100 × 85 × 10 (85,000 cells)	79,644	113,044	101,082	**91,298**	95,935
100 × 11 × 90H (99,000 cells)	526,959	554,947	553,654	180,708	**167,982**
50 × 50 × 40 (100,000 cells)	128,930	130,230	128,637	67,651	**65,174**
50 × 50 × 40 (100,000 cells)	64,839	60,923	**58,627**	62,985	62,547
100 × 43 × 27 (116,100 cells)	**187,156**	191,088	192,511	190,340	194,431
100 × 158H × 9 (142,200 cells)	32,985	41,040	37,721	37,426	**32,274**
100 × 30 × 51 (153,000 cells)	626,596	575,495	510,073	**182,036**	191,359
100 × 91 × 23 (209,300 cells)	9,384	8,796	9,964	10,833	**7,609**
100 × 25 × 96 (240,000 cells)	326,008	303,999	252,491	207,350	**195,656**
100 × 66 × 38 (250,800 cells)	14,286	16,901	20,049	**13,195**	18,869
100 × 55 × 46 (253,000 cells)	153,074	131,129	118,503	111,759	**86,965**
100 × 73 × 35 (255,500 cells)	45,805	41,678	41,086	40,967	**39,407**
100 × 33 × 78 (257,400 cells)	337,506	327,831	**286,579**	339,577	327,393

Table 4: The suppression pattern costs (primary + secondary costs) and the required CPU time (seconds) for different combinations of grouping and pre-selection of secondary cells. H indicates that the dimension is hierarchical. These values are for a single run. The lowest values are highlighted in bold.