BUREAU OF THE CENSUS
STATISTICAL RESEARCH DIVISION
Statistical Research Report Series
No. RR2001/01

# Cell Suppression and Audit Programs used for Economic Magnitude Data

Paul B. Massell
Statistical Research Division
Methodology and Standards Directorate
U.S. Bureau of the Census
Washington D.C. 20233

Report Issued: March 26, 2001

Cell Suppression and Audit Programs used for Economic Magnitude Data
SRD Research Report Series No. RR2001/01
Paul B. Massell

Abstract
Cell suppression and audit programs have been used at the Census Bureau for many years for insuring the confidentiality of establishments that contribute data that are used for building economic magnitude data tables. Since the 1987 Economic Census, the suppression programs were based on network flow models which work well for 2D tables but which have some drawbacks for higher dimensional tables. Linear programming (LP) based models now appear to be a practical option for higher dimensional tables and they do not have these same drawbacks. This paper describes work over the last two years in implementing these LP models, as well as some of the mathematical reasons for preferring the LP based models. Practical aspects of these programs are discussed; e.g., calculating capacities, refinement runs, backtracking, linked tables, frozen cells, and rounded data. A description of earlier work is also included, as are goals for future research.

Key words: statistical disclosure control, confidentiality, cell suppression, audit programs, economic magnitude data, linear programming models, CPLEX

## 1. Introduction

The primary goal of this research report is to provide a description of the cell suppression and audit programs (for three and four dimensional tabular data) that have been created or modified over the period January 2000 to March 2001. We begin with a history of the development of suppression and audit programs at the Census Bureau, focusing on the key algorithmic features. These key features are within the underlying mathematical structure; this structure is often based on one of the well-known methods from the field of operations research, such as network flows or linear programming. We present a table based on a paper by Cox (ref: C2) which evaluates the seminal programs in terms of general desirable features for a suppression program. In our description of recent developments, we focus on how the desired properties of each program can be expressed in terms of the ideas and language of linear programming. We include a fairly detailed description of the routines from the linear programming package CPLEX to show in a concrete way how the key algorithms can be performed in a computationally efficient manner.

It may be helpful to review some of the basic facts and definitions. The distinction between magnitude and frequency tabular data is best seen through an example. Suppose the rows of table are types of retail stores and the columns are cities. If there were exactly five toy stores in the city of Baltimore then a frequency table would have the value '5' for the cell '(toy, Baltimore).' If the annual sales for the 5 stores were x1 > x2 >...> x5 then a magnitude table would have some statistic of the five x(i) values (e.g., the sum) as the

cell value. One major feature of magnitude tables is that the cell values are often dominated by the values of a small number of contributors (e.g., establishments). In order to ensure that accurate estimates of these dominating values (e.g., x1) cannot be made even by the other contributors to the cell value (e.g., x2's), it is necessary to establish a rule for deciding when a cell is too sensitive to be published; such an unpublished cell is called a primary suppression. The Census Bureau currently uses the p% rule to determine primary suppressions; i.e., if it is possible for any contributor to a cell to determine any other contributor's value within p% of its true value, then the cell is suppressed. Complementary suppressions (also called secondary suppressions) are additional cells suppressed to protect the primaries. These are needed since marginal totals are almost always published in economic tables, and the additive relations so generated would often allow a table user to calculate the value of a primary if it were not carefully protected with additional suppressions (ref: WP22).

## 2. History of Suppression and Audit Programs at the Bureau

Suppression programs at the Census Bureau have more than a thirty-year history, and they have undergone an interesting evolution. According to Cox (ref: C2, p.6) the earliest large-scale use of automated suppression programs based on a mathematical theory was for the 1977 U.S. Economic Census. The program was based on combinatorial algorithms for protecting confidential data within a single two-way table. Information loss was defined to be the number of suppressed cells. Disclosure protection in three-way tables was done heuristically by "stacking" constituent two-way tables. Disclosure was based on an (n,k)-rule (a cell was deemed sensitive if the largest n of the contributing establishments accounted for more than k% of the total cell value ref: WP22) and parameters were kept confidential. The suppression module was called INTRA, denoting that suppression was "intra-table." INTRA was used by the Census Bureau for the 1977 and 1982 Economic Censuses, and several surveys. (We now use the p% rule, see above).

Cox in 1987 proposed using mathematical networks for complementary cell suppression. Networks are mathematical models based on flow of a quantity along the arcs of a graph. There is a natural interpretation of a 2D table as a network. A cell suppression program based on networks was written by Bob Hemmig and used for the 1987 U.S. Economic Census. Another suppression program, also based on networks, was written by Bob Jewett and used for the 1992 Economic Census. Since 1992, Jewett (ref: J) extended the program in many ways and it has been used ever since at the Census Bureau. The computational module of this program, viz. the network flow algorithm, is a Fortran subroutine called MCF, written by Professor Klingman at the University of Texas circa 1980 (MCF=Minimal Cost Flow).

One of the key ideas used in the network-based suppression programs is the idea of capacity. The capacity of a cell is a measure of how much the cell can contribute to the protection of a specified primary cell. Jewett states that Cox first came up with this idea in 1991. In the simplest cases, the capacity of a cell is equal to its value. The extension of the basic idea of capacity to the complicated situations that arise in practice was due to Jewett (ref: J, p. II-G, p.1-11). See below for more facts about capacity.

The MCF subroutine and the entire network flow suppression program based on it, are fast

computationally. For 2D tables, the network method is known, under certain conditions on the distributions of contributors to the cell values, to create a suppression pattern that is not undersuppressed, i.e., a pattern with a sufficient number of complements to ensure that the primaries have (at least) the desired amount of protection for each of the contributors (ref:C1). Even when these conditions are not met, there is often at most a small amount of undersuppression in practice. See section below on capacity for more on these conditions. Therefore, for 2D tables, there is a fairly widespread acceptance of the network-based suppression programs. However, for 3D tables, it is known that use of network flow methods is a rough heuristic, so rough that the possibility of undersuppression at the cell level exists and has actually occurred (although often there is only a small amount of it). It is known that LP based suppression always provides adequate protection at the cell level for standard tables. For this reason, Jim Fagan and Laura Zayatz wrote, in 1991, (J: p. III-A-1) an LP based suppression subroutine callable from Jewett's suppression program; it calls an LP subroutine called XMP that was acquired from Dr. Kelly of the University of Maryland. This suppression program, was not fast enough for production work and Jewett decided to try another approach for 3D tables (J: p. III-A-1). Jewett decided to write a 3D suppression using an idea that Bob Hemmig has used for the 1987 Economic Census; namely to use the network flow approach for 3D tables even though it had the possibility of undersuppression (ref: J, p.III-A-2)

The first audit program for assessing the results of a cell suppression program was written by Laura Zayatz circa 1992 (ref: J, p.III-A-3). This audit program used the LP program XMP mentioned above to calculate the feasibility interval (or actual protection level) associated with each suppressed cell (either primary or complementary). It then compares the feasibility interval with the desired protection interval to see if the latter is contained in the former. If the desired interval is contained in the actual interval for each suppressed cell, the cell suppression is deemed a success. The main purpose of the audit program is to determine and write out a list of those suppressed cells whose desired protection interval is NOT contained in the actual protection interval. In that case, the audit determines if the actual protection interval does at least have a width at least as large as that of the desired interval; this is called "sliding coverage." Of course, the most serious situation is where there is little or no protection afforded a suppressed cell; such cases either indicate some trivial data processing error in some input or a programming error with the suppression program or possibly an inherent weakness with the methodology of the suppression program that leads to undersuppression.

Statistical agencies from other countries also have done research in the area of suppression methods; e.g., Statistics Canada uses LP based programs (ref: R) and the Federal Statistical Office of Germany has used a program based on the new hypercube method (ref: G).

## 3. Basic Features of the Optimizing Routines for Suppression Programs

There are many features required of a suppression program other than those performed in the optimizing routine. In this section, we restrict our discussion to the optimizing routine; see reference J for a discussion of other parts of the program.

By basic features, we mean the inputs, outputs, and mathematical structure which operate on the inputs to produce the outputs. For the mathematical structure, we will use the ideas and language of linear programming rather than the ideas and language of network flow models. For the latter see reference C1.

Suppression subroutines for unrounded data.
Inputs:
1. Primary cell inputs; cell location and protection required
   (a single value for protection means we assume the upper and lower protection required are equal; i.e., the uncertainty interval for the value v should contain [v - prot, v + prot]).
2. Table structure information; row relations, column relations, etc.
   (note: this information is used to set up the constraint matrix used in the LP problem)
3. Array of capacities; a measure of the maximum protection every cell can provide for the designated primary
4. Array of costs; a measure of how much information is lost from the table when (later) a set of cells is selected as complements (and is therefore not published)

LP variables created:
For each cell in the table we define two variables; one which denotes an increase from the given cell value; the other a decrease. Two variables are needed because it is convenient to state this LP problem in terms of non-negative variables and we want to allow the solution value for a given cell to represent either an increase or a decrease of the cell value but not both (in network flow language, either positive or negative flow is allowed).

LP Constraint matrix created:
The constraint matrix contains, in a compressed form, information about the set of all additive relations that express the fact that the sum over changes in the cells of a given shaft equal zero. This is simply a fancy way of saying that we are searching for feasible solutions in which the new cell values satisfy the same additive relations as the original cells.

Bounds for variables:
There are two ways to implement this. For cases where one knows the desired protection can be found, it is easiest to do the following. Let lb(i)=lower bound(i); ub(i) =upper bound(i) for the ith LP variable. For the primary cell's positive increase variable set lb=ub=protection; for its negative change variable set lb=ub=0. Viewed in terms of the original table, this forces the primary cell to have a new value = (original value + protection). Thus this positive change variable for the primary cell is the "forcing function" for this problem, where "forcing function" is a term often used in applied mathematics for describing dynamic systems. In cases where one may not be able to find the full protection (this may happen when there are frozen cells, see below), it is better to set lb=0, ub=protection, for the positive increase variable and use a very negative cost coefficient for this variable. In this way, the solution will return the maximum achievable fraction of the desired protection. For cells i other than the primary: lb=0, ub=capacity(i); where capacity is computed in a prior routine. These bounds apply to both LP variables associated with each cell i. See

discussion below on the calculation of capacity.

Cost coefficients:
The cost function is the sum over all variables of the cost coefficients times the variable value; i.e. sum over c(i) * x(i). Here each variable value represents a change in cell value. The cost coefficients are inputs to the LP problem. They often equal the value of the cell associated with the variable. In this case, when we try to minimize the cost function during the LP optimization step, we are trying to find a set of cell changes that takes place in cells with small values.

Optimization and the solution:
For each primary we perform a single optimization, viz. a cost minimization. The actual minimum value of the cost function is not usually the most important part of the solution; what is most important is the set of cells associated with the set of variables which have a non-zero solution value. These are the cells that form the complementary suppression pattern for the given primary.

Outputs:
If either variable (the increase or the decrease) associated with a cell has a non-zero solution value we flag that cell with a 'C' for complement unless it already has a 'C' or 'P' flag. This information is returned to the routine which called the optimizing routine.

## 4. Basic Features of Audit Programs

Unlike the suppression program, the audit program can be written as a single routine which calls some general purpose LP package. The basic features of the 3D and 4D audit programs written by Laura Zayatz are listed below.

Note that these programs were designed to run on positive, additive, unrounded data and our description applies just to that case. More complicated cases are now handled by recently developed audit programs; for those see the section on major changes below.
Inputs:
1. Table structure information:
   These includes table size, row relations, column relations, etc.
2. The value of each cell in the table:
   Note that even the values of the suppressed cells are supplied.
3. A suppression flag, either P (primary) or C (complementary) for each suppressed cell.
4. The desired protection (value) for each suppressed cell.

LP variables:
There are several differences in the way LP variables are defined for the audit program in contrast to the suppression routines. Firstly, there are variables defined only for the suppressed cells ( not for all cells as for suppression routines). For the suppressed cells, only one variable is defined (not two as for suppression

routines). This single variable designates a possible value for the suppressed cell (not a change in value as for suppression routines). It is easy to see that there are, in general, many fewer LP variables defined for a given table in the audit programs (of the type described here) than in suppression routines.

The constraint matrix:
The constraint matrix reflects the same additive relations that were expressed for the suppression routines. However, for audit programs, for a given shaft, the values of all the unsuppressed cells are combined on the right-hand-side of the shaft equation; this combination forms a component of a constant vector (often denoted 'b') that is input to the LP solver (see below).

Bounds for variables:
The bounds for variables are also simpler for the audit program. Each possible value
is bounded below by zero (recall positivity assumption above) and bounded above by the grand total for the table. Note that these bounds remain constant for each of the optimizations that are performed.

Cost coefficients:
The only LP quantity that changes as one changes the cell being considered, is the cost coefficient associated with that particular suppressed cell. The coefficient is zero for all variables other than the one being optimized.

Optimization and the solutions:
For each suppressed cell given in the input file, the audit program performs two optimizations, one to find the minimum, the other the maximum value of the suppressed cell. Thus the upper and lower limits are determined for each suppressed cell in the table. (Actually the audit programs save time by simply finding if the range of each variable contains the (desired) protection interval [value(i) - prot(i) , value(i) + prot(i)]; this can be determined by examining the value of all variables in a solution vector, not simply the variable being optimized.)

The outputs:
For each suppressed cell value i one forms the [min val(i), max val(i)]. The interval so formed, often called the "feasibility interval" or "suppression interval" (ref: WdW2, p. 35), is compared with the (desired) protection interval to determine if the latter interval is contained in the former. If so, it is clear that the given suppression pattern is adequate to guarantee the desired protection, at least at the cell level. If not, the program checks to see if there is "sliding protection", i.e. the width of the feasibility interval is at least as large as the width of the protection interval and the feasibility interval contains the cell value. The program also determines when there is inadequate upper or lower protection or no protection at all, and these troublesome cases are written to a separate file. A summary of results, including the number of primary and complementary cells, as well as the number of these achieving full or sliding protection is written to a file. If protection is not adequate for some primaries, one may try to improve the suppression pattern (either with a program or by hand) to achieve adequate protection.

## 5. Desirable Advanced Features for Suppression Programs

In most aspects, the early INTRA program was inferior to current suppression programs. However, it did find a suppression pattern for all the primaries simultaneously. This is interesting because, in theory, the optimal suppression pattern is achievable, in general, only by treating all the primaries simultaneously (ref: C2). However, in practice, even for moderate sized tables, such problems are computationally infeasible. Thus the network and LP programs are sequential in nature, i.e., they find a pattern for the primaries in sequence. Complementary suppressions are added to those found for previous primaries by simply adding flags to cells as they are identified as complements in a dynamic datafile.

Protection at the enterprise (or establishment or contributor) level refers to the fact that a given enterprise may contribute in a major way to two or more cells in some row, column, or other shaft. If the program ignores that fact, it is likely to overstate the amount of protection provided by a cell to a primary when they have common contributors. Since about 1992, the Bureau programs have built a high degree of protection at the enterprise level into the programs. This protection is built into the program by using a notion of capacity that uses identifiers for the two leading contributors to each cell (ref: J, p.II-G-4)

It is easily shown that network flow models can be used to model the additive structure of a 2D table perfectly. Based on this result, it can be proven that suppression for 2D tables provides adequate protection at least at the cell level. The situation for LP programs is even better; it provides adequate protection at the cell level for all dimensions. Of course, the confidentiality requirement is protection at the enterprise level; protection at the cell is a necessary but not sufficient requirement for enterprise level protection. However, for the special case in which all contributors contribute to at most one interior cell in each shaft, the two notions of protection are equivalent. The drawback of LP methods relative to network models is only one of speed.

One major computational consideration is the size and dimension of a table that can be processed. Because of this constraint, overlapping tables, i.e., tables with common cells, are often not treated as a single higher dimensional structure. Instead, we treat the tables with common cells as "linked tables." This treatment however necessitates the time-consuming process of "backtracking," in which protection required for a primary may be modified if that primary is needed to provide protection for some other cell. See below for more details.

According to Cox (ref: C2, p.5) there are three measures of information loss that are commonly used in the complementary cell suppression literature. They are:

- (S): minimize the number of suppressed cells
- (V): minimize the total value of suppressed cells
- (B): minimize the sum of logarithms of the values of suppressed cells

To implement the associated cost functions exactly it is necessary to use binary variables which are available only in integer programming (ILP) problems. If binary variables y(i) are available, we would set c(i)=1 for all i, and cost = sum (c(i)*y(i)) where y(i)=0 if 'i' is not suppressed, y(i)=1 if it is. In a continuous

variable LP problem, the best approximation we can make is cost= sum(c(i)*x(i)) where x(i) = 0 if 'i' is not suppressed; x(i)=abs(change in cell value) if 'i' has non-zero flow for some solution. This is what we call in the table below the "continuous approximation" of (S). The same idea applies to (V), (B), and similar functions.

Table of Suppression Program Properties

| | INTRA | Network Based | LP Based |
|---|---|---|---|
| Simultaneous suppression | Yes | No | No |
| Protection at enterprise level | No | Some | Some |
| Guarantees adequate supp. for 2D tables at cell level | No | Yes | Yes |
| Guarantees adequate supp. for 3D tables at cell level | No | No | Yes |
| Treats linked tables (Global suppression) | No | Yes | Yes |
| Cost function (see above for defns.) | S,V, or B | cont. approx. of S,V,or B | cont. approx. of S,V, or B |

## 6. Practical Considerations that Complicate Suppression Programs

### Calculating capacities

The notion of capacity for a cell was developed by Larry Cox and Bob Jewett (J: p.II-G-3). Cox suggested that one compute the required protection for the primary cell by itself, and then compute the required protection if the primary suppression were combined with another cell. The capacity of this other cell to protect the primary is defined to be the decrease in the amount of required protection. Actually implementing this general procedure for all possible cases that arise in practice has proven to be complicated (ref: J, section II-G). Capacities for a given shaft (e.g., row, column) may not be additive. The network-based programs are able to work well with this lack of additivity; currently the LP based programs recompute marginals to force additivity. The calculation of capacities is fairly simple when a given contributor appears in at most one interior cell in any shaft. However, deciding how to calculate capacities for the many types of cases in which a given contributor appears in two or more interior cells for a given shaft, can be difficult; for details, see (ref: J, II-G) or the actual Fortran code.

### Refinement Runs

For a given primary suppose a set of complements has been selected by the suppression program. These

cells may have been selected as complements when the costs were equal to the cell values. However, it often useful to run the suppression program a second time in which one starts with a set of candidate complements equal to the set of complements chosen on the first run. The set of complements found in second call will be a subset of those selected on the first call and will still provide adequate protection. Usually one wants to eliminate some of the small cells; this can be done by assigning a higher cost to such cells than to the large cells. For example, if the cost for the first run were defined as the cell value, the cost for the second run could be defined as the negative of the cost for the first run.

Backtracking and linked tables.

Cells often appear in more than one basic table but it is often not computationally feasible to create a higher dimensional table in which the basic tables can be imbedded. However, whenever a cell becomes part of the suppression pattern for a basic table, it is important to ensure that it has adequate protection in all the basic tables in which it appears. For example, in certain tables, a primary may need more protection than that which is necessary to simply protect its contributors. This can occur when the given primary is used to complement another primary that needed a larger amount of protection (J: p. II-A-4, p. II-G-5). For example, if a primary initially requires 60 units of protection (i.e., uncertainty) to protect its own contributors and it later requires 100 units of uncertainty (in its role as a complement) to help protect some other primary, then its new higher level of required uncertainty (100 units) needs to be reflected in the suppression pattern that it generates. That is, its "own" suppression pattern needs to be redone starting with a larger value for "protection" (100 units). This redoing of its suppressed pattern to reflect a larger "protection" is the key idea involved in backtracking for the cell suppression problem. Backtracking sometimes accounts for a significant fraction of the processing time for suppression runs (J: p. II-A-4). It improves the likelihood that a given cell is fully protected "globally" (i.e., across linked tables) but it does not guarantee it.

Frozen cells and partial protection

With linked tables, after processing a single table, one may generate a suppression pattern which one wants to "transfer" to other tables which overlap with the first one. By "transfer" we mean that the cells in common that are suppressed in the first table should be suppressed in all other tables in which they appear; this applies likewise for unsuppressed cells. Such cells whose suppression status in a given table is fixed during a suppression run are called "frozen cells." Frozen unsuppressed cells may cause a problem since the suppression program does not allow such cells to be used as complements to protect primaries. However, by so restricting the set of candidate complements, it is possible that the desired amount of protection for a given primary may not be found. In this case, the program needs to determine the maximum amount of protection that is available; one way to find this quantity is being implemented for the LP based programs. It is based on a technique that has been used in the network based programs, viz., associate a very negative cost coefficient to the (positive) change variable for a cell, with an upper bound equal to the protection desired. This cost function will lead to a solution with the largest protection possible that is consistent with the pattern of frozen cells.

Rounded vs. Unrounded data; Recomputing Marginals.

Let us first review the notion of conventional rounding (ref: WdW1, p.103). An example of this for a rounding base of 1 would be as follows: an unrounded value such as 1.750 is rounded up to 2; whereas 1.499 is rounded down to 1. In general, in conventional rounding, each value is rounded to the nearest multiple of the rounding base. The rounding is also done on the marginals and this often leads to non-additivity in the table; i.e., some marginals after rounding may be close to, but not equal to, the sum of the rounded values of interior cells of the shaft. Cell values are often rounded just prior to publication; they may also be rounded earlier in the production process. Such earlier rounding may have a slight effect on the calculation of capacities; but the network-based programs do not require capacity additivity so these rounding effects are not important. The LP based suppression programs do currently recompute marginals to ensure capacity additivity although this feature may not be needed and it may be eliminated. The audit programs for unrounded data also recompute marginals; however, they recompute cell value marginals, not capacity marginals. In contrast, the audit program for rounded data allows for a lack of additivity for cell values and aborts only when the non-additivity is so great that it could not occur due to rounding alone.

## 7. Major Developments with Suppression and Audit Programs since January 2000

i) In 1999, SRD acquired a license for a commercial linear programming (LP) and mixed integer programming (MIP) package called CPLEX. This package, written in C, has routines that are callable from batch programs; it can also be run interactively. This package is considered to be of high quality; it has a presolver for simplifying the LP problem prior to optimization, correct and fast implementations of some common standard LP algorithms (simplex, dual, and barrier) and is computationally fast. Its routines can be called from either a Fortran or C program. The SRD version runs on a SUN computer running Solaris (v2.6) UNIX . The economic directorate has another license for CPLEX; their version runs on a DEC Alpha with UNIX.

ii) The calls to the linear programming subroutine XMP that existed in the earlier suppression and audit programs were replaced by calls to the various LP routines from CPLEX. Specifically the 3D suppression program that was LP based was updated in this way. Both the 3D and 4D audit programs were also updated. The suppression programs that are used for 2D tables and use network flow methods are known to provide a sufficient amount of suppression at the cell level (ref: C1, p. 1458) and the basic algorithmic structure has not been modified in recent years. Use of CPLEX in the 2D programs may be explored later to determine if CPLEX, when run in its "network mode," runs as fast (or even faster) than MCF.

iii) Efforts have begun to speed up the suppression and audit programs by saving some parts of the solution of the LP problem for one suppressed cell that can be used for the next suppressed cell being processed. In algorithmic analysis, a "warm start" is an informal expression that expresses the idea of using part of an earlier solution in the search for a solution to the next in a series of problems; thus, we have implemented a warm start for the suppression program. The high degree of control of the LP solver allowed in the CPLEX package makes this possible. In the suppression problem, the constraints simply express the fact that the sum of cell value changes in any shaft must be zero. These constraints are clearly the same for each

primary being protected. What differs are the bounds for the cells that are candidate complements, because the bounds are determined by the capacities and these do depend on the given primary. The cost coefficients associated with the cells also change with the primary since typically after a cell is chosen as a complement its cost coefficient decreases.

iv) The 4D audit program was generalized to handle linear constraints other than a simple additive relation; e.g., frequently a sum of two columns equals the sum of the remaining columns. This generality was needed to process certain Manufacturers Energy Consumption Survey (MECS) tables. A generalization like this is easily implemented in CPLEX.

v) The 3D audit program was enhanced so that it can handle rounded values and determine if a suppressed cell in the rounded table can be shown to have a unique rounded value. In such cases, the rounding is providing adequate protection for the cell and the unique value rather than a suppression flag should be published for the cell. This program has two other types of generality; it can handle non-simple additive relations (as in iv) and it allows for negative values in designated columns. The LP structure for rounded data involves many more variables; in fact a variable is now required for each cell; the bounds for the variable depends on whether the cell is suppressed or has a published rounded value; e.g. for a suppressed cell the bounds may be lb=0; ub=grand total for table; for a cell with a rounded value v, lb= v-0.5, ub=v+0.5 if rounding is to the nearest unit.

## 8. The CPLEX Linear Programming Package

The basic CPLEX routines:
For initializing a CPLEX environment use:

```
env = CPXopenCPLEXdevelop (status)
```

Note that this works only if the computer is licensed for the Base System (i.e., the interactive system) and Callable Library use.
For creating an LP problem object with no variables or constraints yet defined use:

```
lp = CPXcreateprob ( env, status, probname)
```

For copying various arrays to the problem object after they have been defined in the code:

```
status = CPXcopylp (env,lp, nvar, ncon, objsen, c, b, sense,
                    matbeg, matcnt, matind, matval, lb, ub)
```

In the call to CPXcopylp, key inputs are:

nvar = the number of LP variables;
ncon = the number of constraints;
objsen = the type of optimization problem; 1 to minimize; -1 to maximize
c = an array of size nvar, the cost (i.e., objective) function coefficients;
b = n array of size ncon, the right hand size constants for the constraints;
sense = an array of size ncon, the sense of the inequalities for each constraint;
matbeg, matcnt, matind, matval = 4 arrays that describe the nonzero elements of the constraint matrix (see code or reference manual for details)

lb = an array of size nvar, the lower bound for each variable
ub = an array of size nvar, the upper bound for each variable

For solving the LP problem, there are three choices for the solver,
To invoke the primal simplex algorithm: status = CPXprimopt (env, lp)
To invoke the dual simplex algorithm: status = CPXdualopt (env, lp)
To invoke the barrier algorithm: status = CPXbaropt (env, lp)
(baropt is used for problems with a very large number of variables, say, more than 100,000)
Any of these solvers can be used to solve a moderate sized suppression problem. However, it appears that dualopt may be slightly faster in general (see section 10 below)

For extracting various components of the solution use:
status = CPXsolution (env, lp, termin , objval, x, pi, slack, dj)
In the suppression programs, the part of the solution of greatest interest is the vector x, a vector with the values for each of the LP variables. A variable with a nonzero value means that the associated cell must be suppressed. For the audit program, the objective function value (objval) is the quantity of greatest interest; it represents the maximum or minimum value that the associated cell can achieve; the interval (min, max) defines the feasibility interval for that suppression.

If the current LP problem object will no longer be used, and a completely new object will be defined then one can free the current object with:
status = CPXfreeprob (env, lp)
When CPLEX is no longer needed in the program one calls:
status = CPXcloseCPLEX (env)

Programming note: the Fortran - C interface:
When calling a routine written in C, such as a CPLEX routine, a scalar quantity such as 'env' must be passed as '%val(env)' ; this passes the variable by value, which is what the C routine expects for a scalar quantity. For arrays and character strings, a C routine expects a call by reference (which is what Fortran does by default; i.e. without the %val( ) ). There are also array indexing and ordering differences between Fortran and C; see the SUN Fortran Programmer's Guide for the F77 and F90 compilers; or the CPLEX User's Manual.

## 9. Implementing Warm Starts and Modifications of an LP problem

Instead of defining a completely new problem object for each successive optimization in a given run of the suppression or audit programs, it is more efficient to modify only those parts of the LP problem that are modified between successive calls to an LP solver.

For the suppression problem, the only parts of the problem that are changed between calls to a solver for successive primaries are the bounds for the variables (these are determined by the capacities) and the cost (=objective) function coefficients (these are often defined in terms of the values of the cells

associated with the variables).

status = CPXchgbds (env, lp, bdscnt, bdsindices, lu, bd)

Here bdscnt = number of bounds that are changed; lu, type of bound; bd, new value of bound.

status = CPXchgobj (env, lp, objcnt, objindices, objvalues)

Here objcnt = number of objective function coefficients that are to be changed, objvalues is the array of new values.

For the audit program, the bounds do not change as we successively determine the lower and upper limits for each suppressed cell. However, the cost function coefficients do change but in a very simple way. In the audit program there is only one variable for each cell, and when we find the lower and upper limits for a given variable var(i) , we simply define cost(var(i))=1 and
set the cost for all other variables equal to zero. As we cycle through the set of suppressed cells, to find the lower bound we set objsen = 1 (for minimization) and then call:

call CPXchgobjsen (env, lp, objsen)

Then we call a solver and extract objval (see above).
To find the upper bound we set objsen = -1 (for maximization) and then call the solver again.

## 10. Future Research

The main topic that will be explored in coming months is determining the design of the LP based programs that will lead to the shortest run times for production work. Specifically we will explore the following questions. Using the LP heuristic for the cell suppression problem, what is the optimal way to implement it using the CPLEX routines ? This involves determining which of the three CPLEX LP solvers is best under given conditions (e.g., size of the table, number of variables, etc.). What is the best way to implement warm starts ? Under what condition will they return a valid solution ? How much of a time savings do warm starts provide ? Would a CPLEX suppression program in 'network flow mode' be able to run faster than the current MCF based suppression program for large production runs on 2D tables ?

A longer range topic to be explored is determining the practicality of integer linear programming (ILP) based programs for production work. One motivation for using ILP based programs was mentioned above; it is only such programs that allow one to implement the exact version of the B,S,V cost functions. ILP based programs would likely also have less oversuppression than the LP based programs. Recent research (ref: F, F-SG) in this area will be studied for hints as to how to best design such programs.

Acknowledgments:

Thanks to Laura Zayatz for introducing me to the very interesting area of cell suppression theory and software and allowing me to focus my attention on it. Thanks also to Jim Fagan for showing me how to get past all the practical complications to see the basic linear programming structure that underlies these problems. Thanks to Bob Jewett for showing me that doing suppression work for production runs involves many more considerations, both conceptual and computational, than the basic theory would suggest. Thanks to Carol Blatt for numerous interesting discussions on the programming aspects of this software.

References:

C1: Cox, Lawrence H., (1995), "Network Models For Complementary Cell Suppression", JASA, June 1995, v.90, p.1453-1462.

C2: Cox, Lawrence H.,(2000), "A Comprehensive Methodology for Complementary Cell Suppression in Tabular Data" (unpublished)

F: Fagan, James, (1999), "Cell Suppression Formulations - Exact Solution and Heuristics" (unpublished)

F-SG: Fischetti, Matteo; J. J. Salazar Gonzalez, (2000), "Models and Algorithms for Optimizing Cells Suppression in Tabular Data with Linear Constraints", JASA, Sept. 2000, v. 95, p.916-928.

G: Giessing, Sarah, (1999), "Looking for Efficient Automated Secondary Cell Suppression Systems: A Software Comparison," Proceedings of the Conference on Statistical Data Protection, Eurostat, 1999

J: Jewett, Robert, (1993),"Disclosure Analysis for the 1992 Economic Census," (Economic Programming Report)

R: Robertson, D., (1994), "Automated Disclosure Control at Statistics Canada", paper presented at the Second International Seminar on Statistical Confidentiality, Luxembourg

S: Sullivan, Colleen, (1993), "A Comparison of Cell Suppression Methods," Proceedings of the International Conference of Establishment Surveys, June 1993

WdW1: Willenborg, Leon, Ton de Waal, (1996), *Statistical Disclosure Control in Practice*, Lecture Notes in Statistics, v. 111, Springer

WdW2: Willenborg, Leon, Ton de Waal, (2001), *Elements of Statistical Disclosure Control*, Lecture Notes in Statistics, v. 155, Springer

WP22: Statistical Policy Working Paper 22 (1994), Report on Statistical Disclosure Limitation Methodology, Statistical Policy Office, OMB, May 1994

Z: Zayatz, Laura, (1992), "Linear Programming Methodology used for Disclosure Avoidance Purposes at the Census Bureau," ASA Proceedings of Survey Research Methods Section