Some Considerations in the Use of Linear Networks to Suppress Tabular Data

Errol Rowe*
Bureau of the Census, Washington, D.C. 20233

ABSTRACT The Census Bureau publishes the results of several surveys in tabular form. However, occasionally it is necessary to suppress certain items, i.e., table cells, of information in order to protect unique or easily identifiable survey respondents. Because of the additivity of the tables, one must select *complementary suppressions* for such tables in order to prevent security breach. The purpose of this paper is to describe linear programming strategies for determining the complementary suppressions.

INTRODUCTION

The Census Bureau publishes the results of several surveys in tabular form. However, occasionally it is necessary to suppress certain items, i.e., table cells, of information in order to protect unique or easily identifiable survey respondents. For example, suppose the table lists the average yearly income of each chemical manufacturer in the state of Delaware. Then, in order to maintain the anonymity of the DuPont chemical company, we would have to withhold its average income; the DuPont company is by far the most dominant chemical company in Delaware, hence its average income is not comparable to any other chemical manufacturers in that state.

Most of the Census Bureau's tables are additive, i.e., the last entry of each row or column is the sum of the other entries in that row or column. Because of this additivity, each row or column with a suppression must contain at least one other suppressed entry. For example, consider Table I in the appendix and suppose some entry, $a_{i,j}$ say, is to be suppressed. Then

$$\begin{aligned} a_{i,j} &= a_{i,n} - \sum_{k=1}^{j-1} a_{i,k} - \sum_{k=j+1}^{n-1} a_{i,k} \\ &= a_{m,j} - \sum_{k=1}^{i-1} a_{k,j} - \sum_{k=i+1}^{m-1} a_{k,j}. \end{aligned}$$

Hence, we must protect the primary suppressed cells - i.e., table cells corresponding to unique or easily identifiable respondents, with *complementary suppressions* - i.e., table cells which protect the primary suppressed cells. The purpose of this paper is to describe linear programming strategies for determining the complementary suppressions.

The usefulness of the suppression scheme will depend on how much information remains in the tables after the complementary suppressions have been determined. For some users, usefulness will depend on how many entries of the table remain after suppression; whereas, for others, usefulness will depend on the total value of the entries of the table that remain after suppression. We will consider primarily the latter situation as the first situation can often be interpreted as a special case of the second.

We describe below a strategy for obtaining a solution of the tabular cell suppression problem using network flow methodology. We also describe a method for restructuring the problem to handle negative flows. And finally, we consider various methods to correct for over suppression.

TABLES AND NETWORKS

Consider Table I in the appendix which, for argument's sake, depicts (n - 1) products produced in (m - 1) counties. Notice that each such table can be represented in network form as indicated in Network I. The $a_{i,j}$ represent the cost of shipping one unit of a commodity across the respective arc.

Thus each two dimensional table has a natural network representation. Suppose now that table entry $a_{i,j}$ is an *initial suppression*, i.e., it is considered too sensitive to be released. Then any closed path in the network containing entry $a_{i,j}$ forms a protection for $a_{i,j}$ in the sense that if none of the table entries corresponding to arcs on the cycle are released, then entry $a_{i,j}$ cannot be determined; for example, suppose table entry $a_{1,1}$ is an initial suppression. Then the closed path containing arcs $a_{1,1}, a_{1,n}, a_{2,n}, a_{2,1}$ provides adequate protection for the initial suppression $a_{1,1}$ since its

value cannot be determined from the resulting unsuppressed terms in the corresponding row or column.

Our objective is to place each initial suppression in a table in a closed path while at the same time minimizing the sum of the complementary suppressions; i.e., the sum of the $a_{i,j}$'s corresponding to arcs on the closed path which are not initial suppressions.

Consider Table II - (a) and its corresponding network. We have used asterisks to indicate initial suppressions. To find the complementary suppressions, we use a minimum cost flow algorithm. Thus, suppose each arc has capacity one - i.e., only one unit can be shipped across each arc, and that the flow may occur in either direction. Also, since we wish to minimize the sum of the complementary suppressions, we let the $a_{i,j}$'s represent the cost of shipping one unit across the arc corresponding to $a_{i,j}$. However, the cost of shipping one unit across an arc corresponding to a initially suppressed cell is set to zero; this encourages the algorithm to choose such cells. Let G and D represent the endpoints of the initial suppressed arc labeled 95. The problem is to find the least cost of shipping one unit through arc GD and back to node G again. The route which minimizes the sum of the complementary cells is indicated in Network II - (b).

We must now consider the problem of finding the minimum cost of shipping one unit from node D, through arc DC, and either back to node D or to any node on the previously determined route GDBAG. See Table II - (c) and Network II - (c) for the solution. We have labled table cells corresponding to traversed arcs which are not initial suppressions by the letter 'c' for complementary suppression. Clearly all initial suppressions now have a complementary suppression; however, the table has now been over suppressed - it would have been cheaper to suppress only the 1000 in the column marginal.

This example demonstrates how over suppression may occur if the costs in the network are based solely on the table entries. At junction D, the algorithm chooses the arc labled 53 over the arc labled 42 because $53 + 716 < 1000$. Indeed, at each junction the standard minimum cost flow algorithms choose the arc which is on the least expensive route from that junction to the destination - in this case node G. This example demonstrates the fundamental drawback of standard flow algorithms to solve the tabular suppression problem; i.e., they work on only one initial suppression at a time. From observing Network II - (d), one may ask if it is not reasonable to consider some scheme which tries to suppress adjacent initial suppressions in the same flow cycle.

The above example points out the need for more creative cost assignment schemes when using network programming to solve the tabular suppresion problem. One suggestion is to assign each initially suppressed arc a value so low that the algorithm is almost forced to choose that arc. For example, suppose in the above example we assign the value -1716 to each of the initially suppressed arcs. Since $-1716 + 1000 < 53 + 716$ we obtain the optimum solution to our problem. See Table II - (d) and Network II - (d).

Forcing the linear program to choose an initially suppressed arc makes sense for the following reason. If the program chooses an arc that is not an initial suppression over one that is, then nothing is gained because ultimately the algorithm must return to the bypassed arc in order to close off that initial suppression. And since the program must eventually determine a flow across the initial suppression, it often makes sense to do so at the first opportunity. One may ask if instead of 1000 we had a much larger number across arc CA, wouldn't we just be forcing another form of over suppression? No, since arc CA would be compared with all other paths from C to A. For example, consider Table III - (a), Table III - (b) and Network III - (b). This solution is optimal.

Of course the above method is not guaranteed to obtain the optimum solution for all tables. Indeed, to obtain the optimal solution requires an integer programming method which is quite costly. Nevertheless, as demonstrated by the simple example above, the technique of adding negative values to the initial suppressions coupled with various cleanup routines to remove over suppressions often works better than the linear programming approaches that assign non-negative values to arcs corresponding to initial suppressions.

Labeling of Initial Suppressions

Consider Tables IV and V. By choosing a number, alpha say, small enough and assigning it to the arcs labeled *50 - that is, the initial suppressions - we obtain an optimum solution for Table IV. The optimum is obtained by making the two arcs labled 28 complementary. However, using the same value for alpha in the network corresponding to Table V, our program arrived at the same flow pattern for Table V, a *solution* which is clearly not optimal. The optimum for Table V is obtained by making the four tens and two sevens complemen-

tary.

Upon replacing alpha with a greater negative value, beta say, we obtained the reverse situation; i.e., the optimum solution for Table V and an approximate solution for Table IV with the flow pattern corresponding to the optimum solution for Table V. Eventually, we were able to find a number, gamma say, between alpha and beta which resulted in obtaining the optimal solution for both tables. This suggests that it might be possible to perform some initial analysis on the tables to determine a *best* value for the initial suppressions. We are reluctant to give actual values for alpha, beta and gamma because the program is still in development and the values for this example tend to change with each modification. Nevertheless, at the time of this writing, alpha = -30, beta = -5 and gamma = -27.

What about Negative cycles?

By allowing the costs of arcs representing initial suppressions to be negative, we may encounter the situation in which the distance routine in the network flow algorithm encounters a negative cycle; i.e., a closed path in which the sum of the cost of its arcs is negative. Once such a cycle is encountered, we consider each of its arcs suppressed and re-assign them the value zero. In so doing we make the cost of any path containing any arcs of the cycle equal to the sum of the arcs external to the cycle. This essentially restructures the network by making the cycle equivalent to a single node.

Preventing Over Suppression

There are a number of methods of correcting for over suppression. We have found the most useful, and unfortunately the most costly, to be one which is done before any complementary suppressions have been sellected. Consider Network III - (b). Suppose alpha is the negative value to be assigned to the initial suppressions. Then the two shortest disjoint paths from node G to F are GDCF and GEBF. Thus, for the suppression problem, any closed path containing nodes G and F should not not contain arc GF. By finding the two shortest disjoint paths from G to F, we have obtained a cycle containing G and F which does not use the direct path GF; i.e., path GDCFBEG. Thus arc GF should not be a part of any optimum suppression cycle. This implies that we should assign the capacity of arc GF to be zero.

By examining the networks we see that the above procedure should be considered for each row against each column, each row against the marginal total and each column against the marginal total. That is, it is only necessary to perform the procedure between nodes that share a common arc.

As mentioned earlier, the above procedure could be quite costly for very large tables. However, with the advent of inexpensive parallel processing computers, this could easily be overcome with a procedure to handle all rows against all columns at once.

Conclusion

As we have demonstrated, linear programming techniques can be used to obtain solutions to the two dimensional tabular suppression problem. However, these methods do not guarantee that the solution will be optimal. We have discussed two techniques of refinment which, at least in some cases, will provide better solutions. In the first method, we try to force the network algorithm to choose a path which contains as many initial suppressions as possible. This method involves assigning negative values to the costs of the initial suppressions. What we have not discussed however, is just how to assign those negative values, i.e., how small should these numbers be. Our discussion earlier, involving Tables IV and V suggests that some research must be done to determine a *best* value for the initial suppressions. We believe this best value is a function of the number of the initial suppressions and the distribution of the initial suppressions over the network.

The second method of refinment involves removing arcs from the network that should not be included in *good* solutions. In doing so, we force the algorithm to choose from a better set of possible solutions.

References

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

Luenberger, D.G. (1984), *Linear and NonLinear Programming*, Second Edition, Reading: Addison-Wesley.

Sullivan, C.M. and Zayatz, L. "A Network Flow Disclosure Avoidance System Applied to the Census of Agriculture", *American Statistical Association, 1991 Proceedings.*

*This paper reports the general results of research undertaken by the Census Bureau staff. The views expressed are attributable to the author and do not necessarily reflect those of the Census Bureau.

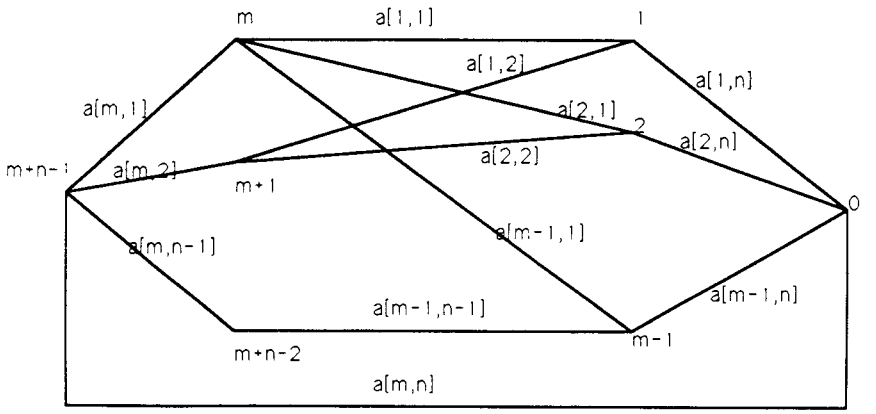
TABLE I

prd(i) = product i

C(i) = county i

	prd(1)	prd(2)	prd(n-1)	total
C(1)	a[1,1]	a[1,2]	a[1,n-1]	a[1,n]
C(2)	a[2,1]	a[2,2]	a[2,n-1]	a[2,n]
C(m-1)	a[m-1,1]	a[m-1,2]	a[m-1,n-1]	a[m-1,n]
total	a[m,1]	a[m,2]	a[m,n-1]	a[m,n]

NETWORK I



Note: for i = 1 to (m - 1), a[i,n] = arc(i,0),

for j = 1 to (n - 1), a[m,j] = arc(m+n-1,m+j-1),

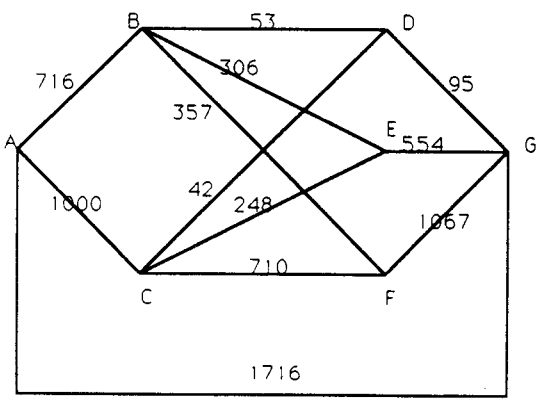
for j = 1 to (m - 1) and k = 1 to (n - 1), a[j,k] = arc(j,m-1+k);

a[m,n] = arc(0,m+n-1)

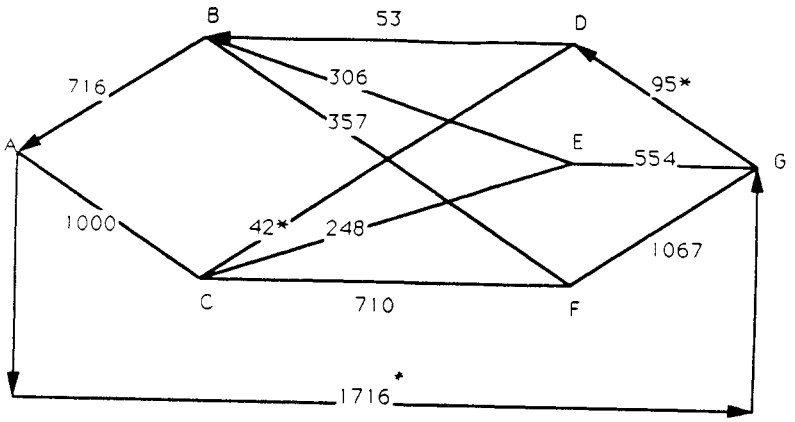
TABLE II - (a)

53	*42	*95
306	248	554
357	710	1067
716	1000	*1716

NETWORK II - (a)



NETWORK II - (b)



NETWORK II - (c)

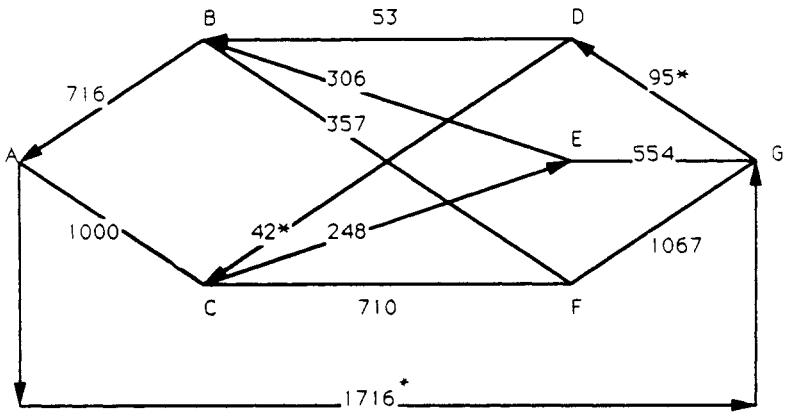


TABLE II - (c)

c 53	*42	*95
c 306	c 248	554
357	710	1067
c 716	1000	*1716

NETWORK II - (d)

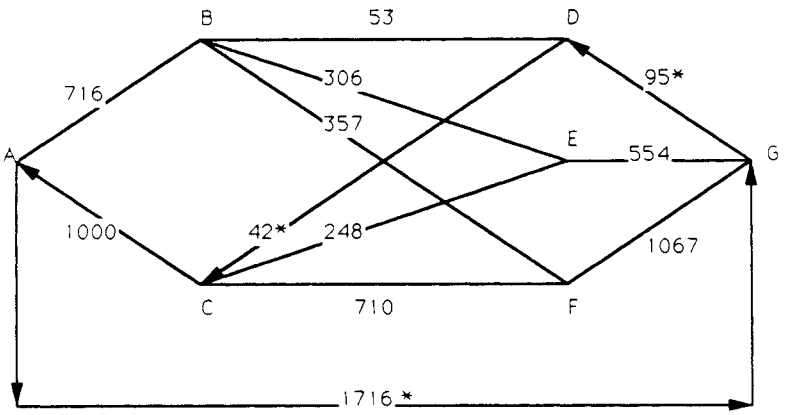


TABLE II - (d)

53	*42	*95
306	248	554
357	710	1067
716	c 1000	*1716

TABLE III - (a)

53	*42	*95
6	248	254
357	1010	1367
416	1300	*1716

NETWORK III - (b)

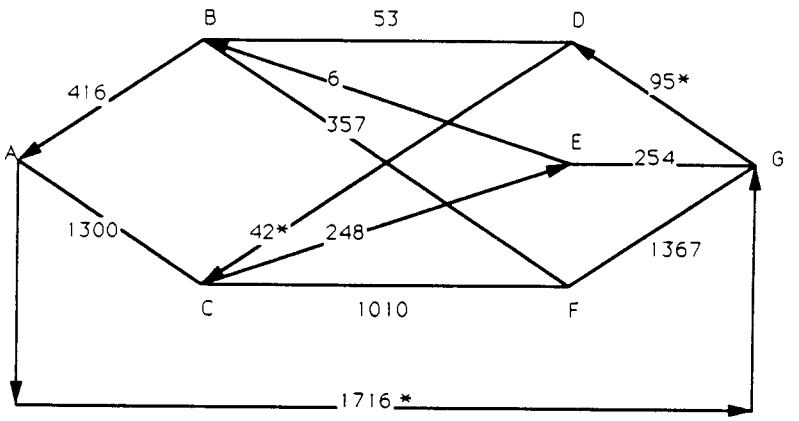


TABLE III - (b)

53	*42	*95
c 6	c 248	254
357	1010	1367
c 416	1300	*1716

TABLE IV

*50	10	100	28	188
10	10	100	100	220
100	100	10	10	220
28	100	10	*50	188
188	220	220	188	816

TABLE V

*50	10	100	28	188
10	7	100	100	217
100	100	7	10	217
28	100	10	*50	188
188	217	217	188	810