**STATISTICAL COMMISSION and**
**ECONOMIC COMMISSION FOR EUROPE**

**CONFERENCE OF EUROPEAN STATISTICIANS**

**Joint ECE/Eurostat Work Session on**
**Statistical Data Confidentiality**
(Skopje, The former Yugoslav Republic of Macedonia,
14-16 March 2001)

**COMMISSION OF THE**
**EUROPEAN COMMUNITIES**

**EUROSTAT**

Working Paper No. 24
English only

Topic II: Impact of new technological developments in software, communications and computing on SDC

# USING MODELING LANGUAGES FOR THE COMPLEMENTARY SUPPRESSION PROBLEM THROUGH NETWORK FLOW MODELS

**Contributed paper**

Submitted by the Universitat Politècnica de Catalunya, Spain[1]

**Abstract:** Several network flows based methods have been suggested in the past for the solution of the complementary suppression problem (CSP). Adding some of those methods to the τ-Argus system is one of the tasks to be performed within the scope of the ongoing IST CASC project. In this paper the authors briefly present how modeling languages can be used for the quick development of algorithm prototypes for CSP. This will permit evaluating and testing different algorithmic options prior to the costly development of an efficient exploitation code. We illustrate the use of modeling languages with a particular network flow based method. This method is implemented using a state-of-the-art modeling language, and some preliminary computational results are presented.

**Key words:** Disclosure protection, complementary cell suppression, linear programming, network optimization, modeling languages.

## I. INTRODUCTION

1. Network flows models have been widely used in the past to avoid the disclosure of sensitive cells of statistical tables see [3] for a comprehensive description). However, the τ-Argus system developed in a previous ESPRIT project lacks such a methodology. Adding it to τ-Argus is one of the tasks to be performed in the recently started IST CASC project. This paper presents the first steps performed to achieve this goal.

2. In particular, we will show that before developing a costly implementation, using a modeling language can be instrumental to see in practice the drawbacks and benefits of a particular approach. We will focus on a heuristic procedure for multiple-cell suppression under a minimum-number-of-suppressions criterion, based on an optimal method for single-cell suppression. This approach is described in [3]. Alternative methods could be attempted and implemented using AMPL in a similar way.

3. An additional benefit of using a modeling language is that trying and developing new algorithms is (usually) a straightforward task. For instance, the behaviour of alternative approaches based on network flows with side constraints [6] or multicommodity flows [2] could be studied, and how can they be applied to multiple dimensional and/or hierarchically related tables.

4. It is worth noting that, unlike network flows based methods that only provide approximate solution, there are other approaches that solve the CSP optimally. For instance in [4] a branch-and-cut

[1] Prepared by Jordi Castro (supported by CICYT Project TAP99-1075-C02-02) and F. Javier Heredia (supported by CICYT Project TAP99-1075-C02-01). This work was suported by the IST-2000-25069 CASC project.

based procedure was presented for the efficient solution of large CSP instances. This methodology is already available in τ-Argus. Network flows based methods can be seen as a complementary tool that will provide the end-user of τ-Argus the option of choosing between different strategies.

5. This paper is organized as follows. Section II briefly describes the simple network flows based method considered in this work. Section III presents the AMPL implementation of the method. Finally, Section IV presents some preliminary results obtained in the solution of a set of 64 randomly generated instances.

## II. A NETWORK FLOWS BASED METHOD FOR THE CSP

6. We will focus on a simple network flows based method (see [3] for a thorough description). The method attempts to minimize the number of secondary suppression cells. When applied to a single primary suppression cell the method is optimum. However, for multiple primary suppression cells the method is applied iteratively, at maximum once for each primary suppression cell, and optimality is not guaranteed (instead, an upper bound to the minimum number of suppressions is obtained).

### II.1 Protecting a single cell

7. Let's consider a table $[a_{ij}]$, $i=1,\ldots,m$, $j=1,\ldots,n$, $m$ being the number of rows and $n$ the number of columns. Marginal total values are denoted as $a_{m+1,j}$, $a_{i,n+1}$ and $a_{m+1,n+1}$, where

$$a_{i,n+1} = \sum_{j=1}^{n} a_{ij} \qquad i = 1,\ldots,m \tag{1}$$

$$a_{m+1,j} = \sum_{i=1}^{m} a_{ij} \qquad j = 1,\ldots,n \tag{2}$$

$$a_{m+1,n+1} = \sum_{j=1}^{n} a_{m+1,j} = \sum_{i=1}^{m} a_{i,n+1} \tag{3}$$

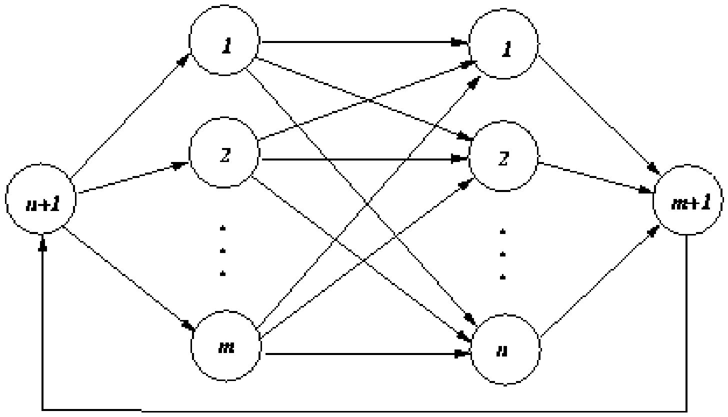
8. The above linear relations can be modeled through a directed graph $G=(V,A)$, $V$ being a set of $m+n+2$ nodes and $A$ a set of $(m+1)(n+1)$ arcs. Figure 1 shows the structure of $G$. There is one node for each row and column of the table (nodes $1\ldots m$ and $1\ldots n$ of Figure 1), plus two additional nodes for the row and column totals (nodes $n+1$ and m+1 of Figure 1, respectively). Cells $a_{ij}$, $1\leq i \leq m$, $1\leq j \leq n$ are related to arcs $(i,j)$. Row totals cells $a_{i,n+1}$, $1\leq i \leq m$ are related to arcs $(n+1,i)$, while column totals cells $a_{m+1,j}$, $1\leq j \leq n$ correspond to arcs $j,m+1$. Total cell $a_{m+1,n+1}$ is related to arc $(m+1,n+1)$.

9. Given a set $S=\{(i_1,j_1),\ldots,(i_p,j_p)\}$ of $p$ suppression cells, one can optimally protect under the minimum-number-of-suppressions criterion a particular target cell $(i_t,j_t)$, $1\leq t \leq p$ solving a network flows problem. This problem is defined over the graph $G'=(V,A')$, where $A'= \{(i,j),(j,i) \mid (i,j) \subseteq A\}$ (i.e., we have in $A'$ a forward and reverse arc for each arc of Figure 1). For each arc in $A'$ we consider one variable, denoted as $x_{ij}^{+}$ if the origin arc corresponds either to a row node or to the column total node $m+1$ (i.e., $(i,j)$, $i=1,2,\ldots,m+1$, $j=1,2,\ldots,n+1$) and $x_{ij}^{-}$ if the origin arc corresponds either to column node or to the row total node $n+1$ (i.e., $(i,j)$, $i=1,2,\ldots,n+1$, $j=1,2,\ldots,m+1$) . Default lower and upper bounds are respectively 0 and 1 for the $2(m+1)(n+1)$ variables. Node injections are zero. Since bounds and injections are integer, it is guaranteed that the solution of any defined network flows problem will be integer as well [1] (in this case, it will be a pattern of 0-1 flows). Moreover, since we consider zero injections at nodes, the optimal solution of any problem defined over this network will be either 0 for all the variables, or -should we forced a positive flow- a 1 flow for some arcs forming a cycle $C$ in $A'$.

10. It can be shown that the target cell $(i_t,j_t)$ will be protected if the optimal solution of the network flows problem is a nontrivial cycle $C$ in $A'$ such that $(i_t,j_t) \subseteq C$. By nontrivial cycles we mean cycles with a number of arcs ≥4, thus avoiding spurious solutions as $x^{+}_{it,jt}=x^{-}_{it,jt} = 1$ and 0 for the remaining variables. This can be guaranteed by imposing an upper bound of 0 to variable $x^{-}_{it,jt}$. The rest of $x^{+}_{ij}$ or $x^{-}_{ij}$ variables of the cycle correspond to $(i,j)$ cells that need to be suppressed to protect cell $(i_t,j_t)$. Then, in order to

minimize the number of suppressions, cells of *S* have to be used whenever possible before including additional complementary ones. This is guaranteed if the following cost vector is used during the optimization of the network flows problem:

**Figure 1: Representation of a table as a directed graph**



(4) $$c_{ij}^{+} = \begin{cases} -(C+1) & \text{if} \quad (i, j) = (i_t, j_t) \\ 1 & \text{if} \quad (i, j) \in S, (i, j) \neq (i_t, j_t) \\ |S| & \text{if} \quad \text{(i, j)} \notin \text{S,} \end{cases}$$

(5) $$c_{ij}^{-} = \begin{cases} 1 & \text{if} \quad (i, j) \in S \\ |S| & \text{if} \quad \text{(i, j)} \notin \text{S,} \end{cases}$$

where |*S*| is the cardinality of *S*, and

(6) $$C = \sum_{(i,j)\neq(i_t,j_t)} (c_{ij}^{+} + c_{ij}^{-}) = 2[((m+1)(n+1) - |S|)|S| + 2(|S|-1)$$

11. This cost vector forces to set $x^{+}_{it,jt}$ = 1 due to its large negative cost. Variables associated with cells in *S* will also be used before adding any complementary suppression cell. The solution of this problem provides an optimal solution for the protection of the single cell ($i_t$,$j_t$) under the minimum-number-of-suppressions criterion.

## II.2 Protecting multiple cells

12. The protection of multiple cells can be accomplished by iteratively applying the procedure described in previous section. This heuristic procedure will provide an upper bound instead of the optimum number of secondary suppressions, as noted in [3]. Figure 2 shows the main steps of a naive algorithm based on this idea. The parameters of the algorithm are a table and a set of primary suppressions denoted by $S_0$, possibly ordered by descending importance. *SP*, the set of protected cells, is initially equal to the empty set. At each iteration of the algorithm we choose the next target cell of $S_0$ not yet protected (line 3). The protection of the target cell is performed through the network flows problem of previous subsection (line 4). The set *S* to be used for the definition of the costs (4) and (5) is $S_k$, which is modified at each iteration of the algorithm. The cycle of the optimal solution corresponds to those cells that were protected by the network flows problem. Some of them will already be in $S_k$. The rest are complementary suppressions, which are added to $S_{k+1}$ and the set of protected cells *SP*. The algorithm iterates until $S_k$ is equal to *SP*, i.e., all the primary suppressions of $S_0$ are protected. Note that, at most, one network flows problem needs to be performed for each primary suppression cell of $S_0$. Although this algorithm could be improved with additional refinements, its current form will be enough to show how a modeling language can be used for a quick implementation.

**Figure 2: Multiple-cell protection heuristic procedure.**

```
Algorithm Multiple-cell protection (Table, $S_0$):
1. $SP := \emptyset, k := 0$;
2. While $S_k \neq SP$ do
    3. Find next target cell $(i_t,j_t) \in S_0$ such that $(i_t,j_t) \notin SP$;
    4. Solve network flow problem of subsection 2.1 for $S := S_k$;
    5. Obtain cycle $C$ of the optimal solution;
    6. $SP := SP \cup C$;
    7. $S_{k+1} := S_k \cup C$;
    8. $k := k+1$;
9. end_while
End_algorithm
```

## III. THE AMPL MODEL

13. We chose the AMPL modeling language [5] for the implementation of the single and multiple-cell suppression problems of subsections II.1 and II.2. This is one of the most versatile modeling languages currently available, permitting, among other features, implementing iterative procedures (as that of Figure 2 and solving the resulting optimization problems through a high variety of packages including own routines). Moreover, there are specialized servers in Internet (e.g., http://www-neos.mcs.anl.gov/neos) that freely permit the remote solution of problems formulated in the AMPL language [8].

14. Briefly, to implement the CSP algorithms of previous sections through AMPL we need:

- A .dat file with the data of the particular instance to be solved (table, and set of primary suppression cells).
- A .run file with the description of the iterative algorithm, i.e., the multiple-cell protection heuristic of Figure 2.
- A .mod file with the description of the network flows problems of subsection 2.2 to be solved at each iteration.

15. Although we won't enter into details, it is worth to include the code of the above files to see how straighforward is an AMPL implementation (see Figures 3-5 of Appendix A). Figure 3 shows the data for a particular instance made of a table 3×4 and 2 primary suppression cells -(1,1) and (2,2). Figures 4 and 5 show all the code required to implement respectively the iterative procedure and the network flows problems. Note that only 72 lines of code were required. This permits easily trying alternative algorithmic options (new heuristics, different costs---with the inclusion of cells values-, alternative solvers, etc.) before developing a final exploitation code.

## IV. COMPUTATIONAL RESULTS

16. We tested the AMPL implementation of the previous section with a set of 64 randomly generated problems. Each instance depends of three parameters $(m,n,p)$, which denote the number of rows, columns and primary suppressions, respectively. The $p$ primary cells were randomly distributed inside the table. The 64 instances were obtained considering all the combinations for $m,n,p \in \{50,100,150,200\}$.

**Table 1: Results for the 64 randomly generated problems**

| $m$ | $n$ | $p$ | $\|SP\|$ | NF | $\text{CPU}_{\text{NF}}$ |
|---|---|---|---|---|---|
| 50 | 50 | 50 | 95 | 33 | 2.04 |
| 50 | 50 | 100 | 135 | 42 | 2.69 |
| 50 | 50 | 150 | 162 | 47 | 3.04 |
| 50 | 50 | 200 | 203 | 71 | 4.54 |
| 50 | 100 | 50 | 90 | 29 | 3.68 |
| 50 | 100 | 100 | 145 | 47 | 6.09 |
| 50 | 100 | 150 | 184 | 62 | 8.14 |
| 50 | 100 | 200 | 233 | 85 | 11.19 |
| 50 | 150 | 50 | 101 | 35 | 6.97 |
| 50 | 150 | 100 | 168 | 61 | 12.19 |
| 50 | 150 | 150 | 219 | 76 | 15.35 |
| 50 | 150 | 200 | 255 | 86 | 17.41 |
| 50 | 200 | 50 | 104 | 35 | 9.42 |
| 50 | 200 | 100 | 172 | 61 | 16.53 |
| 50 | 200 | 150 | 225 | 78 | 21.40 |
| 50 | 200 | 200 | 272 | 94 | 25.91 |
| 100 | 50 | 50 | 101 | 37 | 4.75 |
| 100 | 50 | 100 | 148 | 50 | 6.75 |
| 100 | 50 | 150 | 194 | 65 | 8.65 |
| 100 | 50 | 200 | 234 | 80 | 10.78 |
| 100 | 100 | 50 | 97 | 31 | 8.28 |
| 100 | 100 | 100 | 181 | 63 | 17.16 |
| 100 | 100 | 150 | 226 | 76 | 20.92 |
| 100 | 100 | 200 | 254 | 78 | 21.47 |
| 100 | 150 | 50 | 99 | 30 | 12.66 |
| 100 | 150 | 100 | 183 | 59 | 24.96 |
| 100 | 150 | 150 | 260 | 89 | 37.33 |
| 100 | 150 | 200 | 284 | 86 | 36.88 |
| 100 | 200 | 50 | 108 | 33 | 19.16 |
| 100 | 200 | 100 | 197 | 65 | 37.90 |
| 100 | 200 | 150 | 261 | 89 | 52.16 |
| 100 | 200 | 200 | 310 | 105 | 62.27 |

17. Table 1 shows the results obtained. Executions were performed on a Sun Ultra2 200MHz workstation, of approx. 68 Linpack Mflops, 14.7 Specfp95 and 7.8 Specint95 (this machine is approximately equivalent to a 350 Mhz Pentium PC). Columns $m$, $n$ and $p$ have the meaning above described. Column $|SP|$ gives the final number of (primary plus secondary) suppressions obtained by the heuristic procedure. Column NFP shows the number of network flow subproblems solved. Finally, column $\text{CPU}_{\text{NF}}$ gives the CPU time spent by the code in the solution of the network flows problems. They were solved using the state-of-the-art network solver of the Cplex 6.5 package [7]. The execution time of the overall procedure is not presented because it is meaningless: AMPL is not a compiled language, thus executions are fairly large. In an efficient C implementation the overall execution time should be close to that of the $\text{CPU}_{\text{NF}}$ column. No computational results were presented in [3] for a similar approach, so the results obtained can not be compared with those of previous implementations.

18. It is worth to note that these results have been obtained with a naive heuristic procedure, thus it should be possible to improve them. However, although the number of network flow problems solved is fairly less than $p$ -the number of primary cells-, their solution still requires a significant computational effort. Moreover, this effort increases with $p$. Even with alternative network solvers [9], for large instances, this procedure could be very expensive. In this case, a quickly developed AMPL model has been enough to understand the main drawbacks of this approach, avoiding a costly C implementation of the algorithm.

**Table 1 (cont): Results for the 64 randomly generated problems.**

| m | n | P | \|SP\| | NF | $CPU_{NF}$ |
|---|---|---|---|---|---|
| 150 | 50 | 50 | 98 | 33 | 6.50 |
| 150 | 50 | 100 | 160 | 56 | 11.44 |
| 150 | 50 | 150 | 212 | 76 | 15.51 |
| 150 | 50 | 200 | 260 | 91 | 18.48 |
| 150 | 100 | 50 | 99 | 30 | 12.48 |
| 150 | 100 | 100 | 180 | 56 | 23.48 |
| 150 | 100 | 150 | 233 | 75 | 31.91 |
| 150 | 100 | 200 | 278 | 88 | 38.34 |
| 150 | 150 | 50 | 102 | 31 | 20.85 |
| 150 | 150 | 100 | 189 | 62 | 41.43 |
| 150 | 150 | 150 | 264 | 91 | 61.00 |
| 150 | 150 | 200 | 317 | 104 | 73.54 |
| 150 | 200 | 50 | 105 | 32 | 29.16 |
| 150 | 200 | 100 | 196 | 60 | 54.94 |
| 150 | 200 | 150 | 265 | 87 | 80.64 |
| 150 | 200 | 200 | 342 | 119 | 13.29 |
| 200 | 50 | 50 | 97 | 32 | 8.73 |
| 200 | 50 | 100 | 176 | 64 | 17.60 |
| 200 | 50 | 150 | 225 | 78 | 22.12 |
| 200 | 50 | 200 | 279 | 99 | 28.02 |
| 200 | 100 | 50 | 101 | 30 | 18.03 |
| 200 | 100 | 100 | 181 | 60 | 36.06 |
| 200 | 100 | 150 | 254 | 87 | 53.28 |
| 200 | 100 | 200 | 304 | 102 | 66.61 |
| 200 | 150 | 50 | 107 | 34 | 31.74 |
| 200 | 150 | 100 | 204 | 67 | 62.58 |
| 200 | 150 | 150 | 277 | 91 | 85.49 |
| 200 | 150 | 200 | 335 | 112 | 107.62 |
| 200 | 200 | 50 | 110 | 32 | 41.30 |
| 200 | 200 | 100 | 214 | 67 | 86.70 |
| 200 | 200 | 150 | 282 | 88 | 113.93 |
| 200 | 200 | 200 | 354 | 116 | 151.85 |

## V. CONCLUSIONS

19. The main concern of this paper has been to show how modeling languages as AMPL could be used to obtain easy and quick (although not efficient) prototype implementations. As an example, the AMPL code for the heuristic procedure of Cox [3] for CSP has been presented. This prototype implementation has been used to solve a set of randomly generated problems ranging from *m*=50, *n*=50 and 50 primary cells to *m*=200, *n*=200 and 200 primary cells. Developing and testing additional networks based methods, and implementing for τ-Argus those that eventually appear to be efficient, is part of the further work to be done.

### References

[1] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] J. Castro, A specialized interior-point algorithm for multicommodity network flows, *SIAM J. on Optimization* 10(3), pp. 852-877, 2000.

[3] L.H. Cox, Network models for complementary cell suppression, *J. of the American Statistical Association* 90, pp. 1453-1462, 1995.

[4] M. Fischetti and J.J. Salazar, Models and algorithms for the 2-dimensional cell suppression problem in statistical disclosure control, *Mathematical Programming* 84, pp. 283-312, 1999.

[5] R. Fourer, D.M. Gay and B.W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Duxbury Press, 1993.

[6] F.J. Heredia and N. Nabona, Numerical implementation and computational results of nonlinear network optimization with linear side constraints,. P. Kall (Ed.*). System Modelling and Optimization. Proceedings of the 15th IFIP Conference*, Springer-Verlag, pp. 301-310, 1991.

[7] ILOG CPLEX, *ILOG CPLEX 6.5 Reference Manual Library*, ILOG, 1999.

[8] J. Czyzyk, M. Mesnier and J. Moré, The NEOS Server, *IEEE Journal on Computational Science and Engineering* 5, pp. 68-75, 1998.

[9] M.G.C. Resende and G. Veiga, An implementation of the dual affine scaling algorithm for minimum cost flow on bipartite uncapacitated networks, *SIAM J. on Optimization* 3, pp. 516-537, 1993.

## APPENDIX. AMPL FILES

**Figure 3: AMPL .dat code for a particular instance.**

```
param m := 3;
param n := 4;
param p := 2;
param:           p_r             p_c :=
1                1               1
2                2               2 ;
param a :
        1       2       3       4       5 :=
1       1       111     172     165     449
2       500     1       9       256     766
3       297     143     212     184     836
4       798     255     393     605     2051 ;
```

**Figure 4: AMPL .run code for the iterative procedure.**

```
model csp.mod;
data csp_instance.dat;
option solver cplex65;
param i_p;
let S0 := {};
for {i in 1..p}{
        let S0:= S0 union {(p_r[i],p_c[i])};
}
let S := S0;
let i_p := 1;
let TARGET := {(p_r[i_p],p_c[i_p])};
let Sprot :={};
let CYCLE :={};
problem Nc: Num_CS, Xpos, Xneg, Row, Column;
repeat
{
   solve Nc;
   let CYCLE := {};
   for {(i,j) in (LINKS)}
   {
      if Xpos[i,j]>0.99 or Xneg[i,j]>0.99 then{
         let CYCLE := CYCLE union {(i,j)};
      }
   }
   let S := S union CYCLE;
   let Sprot := Sprot union CYCLE;
   if card(S) != card(Sprot) then {
        repeat {
           let i_p:= i_p + 1;
        } until (p_r[i_p],p_c[i_p]) not in Sprot;
        let TARGET := {(p_r[i_p],p_c[i_p])};
   } else {
        let TARGET:={};
        let i_p := 0;
   }
} until card(Sprot) == card(S);
```

**Figure 5: AMPL .mod code for the network flows problems.**

```
param m integer >0 ; #number of rows (without totals row)
param n integer >0 ; #number of columns (without totals column)
param p integer >0;  #number of primary suppression cells
param p_r{1..p} integer, >=1, <=m; #row of each primary suppression
param p_c{1..p} integer, >=1, <=n; #column of each primary
suppression
set ROWS := {1..m+1};
set COLUMNS := {1..n+1};
set LINKS :={ROWS, COLUMNS};
param a {LINKS}; #(m+1)*(n+1) table values (including totals)
#
set CYCLE within LINKS;
set TARGET within LINKS;
set S0  within LINKS; #primary suppressed cells
set S within LINKS; #current primary and secondary suppressed cells
set Sprot within S; #protected cells
#
param cneg {(i,j) in LINKS} := if (i,j) in S then 1 else card(S) ;
param cpos {(i,j) in LINKS} :=
   if (i,j) in TARGET then
        -( 2*(card(LINKS)-card(S))*card(S) + 2*(card(S)-1) + 1)
   else if (i,j) in S then
        1
   else # (i,j) not in S
        card(S);
#
# Definition of networks flow problem
#
minimize Num_CS;
node Row {k in ROWS}: net_in = 0;
node Column {k in COLUMNS}: net_in = 0;
arc Xpos {(i,j) in LINKS} >= (if (i,j) in TARGET then 1 else 0),
    <= (if a[i,j]== 0 then 0 else 1),
    from Row[i] to Column[j], obj Num_CS cpos[i,j];
arc Xneg {(i,j) in LINKS} >=0, <= (if (i,j) in TARGET or a[i,j]==0
    then 0 else 1), from Column[j] to Row[i] , obj Num_CS cneg[i,j];
```