ABSTRACT

This paper provides a comprehensive introduction to PROC NETFLOW, which is one of the operations research procedures in SAS/OR. This paper describes how PERSCOM uses SAS/OR’s PROC NETFLOW to calculate the optimal distribution of Army officers to jobs.

INTRODUCING THE PROBLEM

The Army employs about 40,000 commissioned officers to fill about 45,000 jobs. The Army has jobs in five pay grades, thirty career fields, and forty geographic regions. Army officers are qualified to work in at least two career fields, in two grades, and in any region. This paper shows how PERSCOM uses SAS/OR to calculate the optimal distribution of officers to fill these jobs. That is, PROC NETFLOW calculates the plan that PERSCOM uses to distribute Army officers to the right job, at the right place, at the right time.

MODEL THE PROBLEM

Operations research analysts call the problem of matching people to jobs the assignment problem. We model the assignment problem with a supply and demand network. Figure 1 is a small portion of the assignment problem under consideration. We represent the available supply of officers on the left and the required demand of jobs on the right. We call these supply and demand points nodes. In figure 2 we connect the supply nodes to the demand nodes with lines called arcs. We want to know how much supply should flow through each arc so as to fill as much of the demand as possible.

Each node on the LEFT represents a supply of officers having the same grade and career fields. The positive number to the left of each node is the number of officers having that combination of grade and career fields. For example, in figure 1 supply node 3.21.49 represents a supply of 25 officers at grade 3 who are qualified in career fields 21 and 49. A double zero in the second career field means that an officer has no second career field.

Each node on the RIGHT represents a demand of jobs requiring a particular grade and career field. The negative number to the right of each node is the number of jobs requiring that grade and career field combination. For example, the demand node at the top right of figure 1 indicates that there are 92 jobs at grade 3 in career field 21 in Europe.

The node at the BOTTOM LEFT of figure 1 is called the super supply node. This node represents a large supply of fake officers available to fill any job. We use this supply of fake officers to insure that there is enough supply to meet the demand. Now, PROC NETFLOW will create this super supply if there is not enough supply to meet the demand. However, we want to control the amount of super supply in our problem so we create our own super supply. PROC NETFLOW simply does not use the excess supply we provide.

Each arc (from a supply node to a demand node) represents a feasible assignment of officers to jobs. Note that an arc exists only where there is a match by grade and career field. For nodes where officers are qualified in two career fields, two arcs depart those nodes. Furthermore, since officers can fill a job at one grade above their current grade, we have arcs for these matches as well. For example, the officers from the supply node 3.21.49 can be assigned to work in jobs requiring a 3.21, 3.49, 4.21, or 4.49 in any of the regions. Note on figure 2 that each arc can be uniquely identified by its from and to nodes. Also note that there is an arc from the super supply to every demand node.

DEFINE & REFINE THE PROBLEM

PROC NETFLOW calculates a solution to this problem by determining the flows through each arc so as to meet the demand at every node without exceeding the supply at any node. We define the best solution as the one that leaves the fewest jobs vacant. That is, our objective is to minimize the use of the officers from the super supply.
In order to accomplish this objective, we must develop a mechanism to force the assignment model to use all of the actual officers before it uses any from the super supply. Network programming uses a costing structure to model this type of situation. PROC NETFLOW’s optimization procedure charges a cost for every unit of flow that moves through the arc from the supply to the demand. Thus, we simply must insure that the cost of an assignment from the available officers is always less expensive than an assignment from the super supply. With this costing structure, PROC NETFLOW will use the available (inexpensive) officers before it uses the super supply as it calculates the best (optimal) solution.

CREATE THE SAS DATA SETS

PROC NETFLOW requires two input SAS data sets: the nodedata and the arndata. PROC NETFLOW produces an output SAS data set called the arcout that contains the solution to the problem. The following paragraphs discuss these three data sets. (The underscored variables are special ones that PROC NETFLOW looks for by default.)

The nodedata data set contains an observation for every node in the network flow model. There are only two required variables, _node_ and _supdem_. _Node_ must be a character variable that uniquely identifies a node. _Supdem_ must be a numeric variable that gives either the supply or demand on each node. A node that provides supply to the network has a positive _supdem_ whereas a node that requires demand has a negative _supdem_. Creating the nodedata is fairly straightforward. The SAS programmer must uniquely identify each node and assign it a positive supply or a negative demand.

The arndata data set contains an observation for every arc in the network flow model. There are only three important variables: _from_, _to_, and _cost_. The _from_ and _to_ are required character variables whose values must identify a _node_ variable in the nodedata. This means that every arc’s _from_ and _to_ values (beginning and ending points) must also be the value of a _node_ in the nodedata. This makes intuitive sense if you consider that an arc without a node is meaningless. Furthermore, each arc must have a unique _from_ to combination. _Cost_ is a required numeric variable. PROC NETFLOW charges this cost for each unit that flows through the arc. Creating the arndata is not so straightforward. The SAS programmer must use the DATA step to match the supply nodes to the demand nodes and to assign a cost to each arc. Aster and Seidman explain a useful technique called match-crossing data sets in their book, Professional SAS Programming Secrets, that accomplishes this task.

Although not required, the numeric _capac_ variable is often useful. _Capac_ gives a maximum allowable flow through the arc. For example, an analyst could set _capac=0 to effectively eliminate an arc from the problem without having to delete the arc from the SAS data set. In our example, we use _capac_ to limit the number of super supply officers that can flow to some demand nodes. This means that we can limit the number of vacant jobs at demand nodes by setting an upper bound (_capac_) on the amount of flow from the super supply to those demand nodes.

The arcout data set contains the optimal solution to the network flow model. It contains the same variables as the arndata and more. _Flow_ is the critical output variable. PROC NETFLOW calculates the _flow_ through each arc so as to minimize the sum total of the _flow_ times _cost_ product for each arc. Furthermore, PROC NETFLOW enforces a conservation of flow constraint at every supply and demand node. First, the flow out of a supply node cannot exceed its available supply. Second, the total flow into each demand node must meet the required demand.

SIDE CONSTRAINTS

Now, the costing structure described above cannot capture all of the real world constraints on the distribution of Army officers world-wide. For example, the Army requires that a certain number of the jobs at every demand node be filled with real officers, that is, that every demand node must get at least a minimum level of real (as opposed to fake) officers. Figure 3 is a network of demand node 4.49.USA. Say for example that we want to insure that at least 200 of the 250 jobs at 4.49.USA are filled with real officers,
i.e., not with officers from the super supply. PROC NETFLOW can handle this situation through what it calls side constraints. Through side constraints, we can tell PROC NETFLOW that the sum of the flow's through these real arcs must be greater than or equal to 200.

The side constraints are contained in a third input data set called the condata. The condata SAS data set (using the sparse condata option) contains four important variables: _type_, _coef_, _row_, & _column_. Most operations researchers are familiar with this sparse constraint matrix format. By the way, in order to use condata, the arcdata must give a unique name to each arc, called _name_. The ability to use side constraint data is a great enhancement to PROC NETFLOW. This means that many real-world problems can be solved with SAS/OR.

Another way to handle this situation is through use of our own super supply. As mentioned above, we could limit the capacity of the arc from the super supply to this demand node by setting its _capac_=50. This method is much more efficient than the side constraint method from both a computational viewpoint and from a code maintenance viewpoint. However, not all side constraint situations can be modeled using the _capac_ variable.

SOLVE THE PROBLEM

PROC NETFLOW solves the assignment problem. That is, PROC NETFLOW calculates the optimal flow of supply to demand through the network so as to fill the demand at the minimum total cost. It uses a variant of the revised primal simplex algorithm that exploits the network structure of the problem. PROC NETFLOW first solves the problem without the side constraints and then it solves the problem with the side constraints. SAS/OR required about two minutes to calculate the solution to the problem without side constraints on a small workstation. This problem has about 78,000 arcs that connect 783 real supply nodes to 2483 demand nodes. PROC NETFLOW required about 1 & 1/2 hours to solve the side constrained problem.

CONCLUSION

Network flow programming is one of the most widely used optimization tech-
Figure 1, Assignment Problem, Node Data

<table>
<thead>
<tr>
<th>supply of officers</th>
<th>demand of jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>supdem</em> <em>node</em></td>
<td><em>node</em> <em>supdem</em></td>
</tr>
<tr>
<td>(+762) 3.21.00</td>
<td>O 3.21.Europe (-92)</td>
</tr>
<tr>
<td>(+025) 3.21.49</td>
<td>O 3.21.USA (-362)</td>
</tr>
<tr>
<td>(+411) 4.21.00</td>
<td>O 3.21.Korea (-33)</td>
</tr>
<tr>
<td>(+044) 4.21.49</td>
<td>O 3.49.Europe (-5)</td>
</tr>
<tr>
<td>(+99,999) #.##</td>
<td>O 3.49.USA (-190)</td>
</tr>
<tr>
<td></td>
<td>O 3.49.Korea (-1)</td>
</tr>
<tr>
<td></td>
<td>O 4.21.Europe (-50)</td>
</tr>
<tr>
<td></td>
<td>O 4.21.USA (-125)</td>
</tr>
<tr>
<td></td>
<td>O 4.21.Korea (-10)</td>
</tr>
<tr>
<td></td>
<td>O 4.49.Europe (-7)</td>
</tr>
<tr>
<td></td>
<td>O 4.49.USA (-250)</td>
</tr>
<tr>
<td></td>
<td>O 4.49.Korea (-2)</td>
</tr>
</tbody>
</table>

1247
Figure 2 Assignment Problem, Node & Arc Data

supply of officers

```
  __supdem__ _node_ :: _from_ _cost_ _to_ :: _node__ _supdem_
```

<table>
<thead>
<tr>
<th>Node</th>
<th>Europe</th>
<th>USA</th>
<th>Korea</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.21.</td>
<td>(-92)</td>
<td>(-362)</td>
<td>(-33)</td>
</tr>
<tr>
<td>3.49.</td>
<td>(-5)</td>
<td>(-190)</td>
<td>(-1)</td>
</tr>
<tr>
<td>4.21.</td>
<td>(-50)</td>
<td>(-125)</td>
<td>(-10)</td>
</tr>
<tr>
<td>4.49.</td>
<td>(-7)</td>
<td>(-250)</td>
<td>(-2)</td>
</tr>
</tbody>
</table>

(+762) 3.21.00
(+025) 3.21.49
(+411) 4.21.00
(+044) 4.21.49
(+99,999) #.##
Figure 3, Side Constraint Example

supply of officers

(+50) 3.11.49  O
(+35) 3.12.49  O
(+40) 3.13.49  O
(+25) 3.14.49  O
(+45) 3.15.49  O
(+50) 4.00.49  O
(+20) 4.18.49  O
(+15) 4.21.49  O
(+15) 4.31.49  O
(+30) 4.42.49  O
(+05) 4.44.49  O
(+9,999) #.##  O

demand of jobs

O 4.49.USA (-250)