A Tutorial on the NETFLOW Procedure in SAS/OR

Trevor D. Kearney
SAS Institute Inc.
Cary, North Carolina

Abstract

Many problems encountered in industry can be formulated and solved as constrained network models. PROC NETFLOW in SAS/OR software (versions 6.03, 6.04 and 6.06) finds optimal solutions to such models. This tutorial will cover the following topics:

- network modeling techniques,
- the performance of PROC NETFLOW,
- new, undocumented or changed options in versions 6.04 and 6.06.

Introduction

Constrained network models can be used to describe a wide variety of real-world applications ranging from product inventory and distribution problems to financial applications. These models are conceptionally easy since they are based on network diagrams that represent the problem pictorially. This not only simplifies problem description but also aids in the interpretation of the solution. These problems can be solved with PROC NETFLOW in the SAS/OR software product.

A network consists of a collection of nodes and a collection of arcs. The arcs connect nodes and convey flow of one or more commodities which are supplied at supply nodes and demanded at demand nodes in the network. Each arc has a cost per unit of flow, a flow capacity and lower flow bound associated with it. An important concept in network modeling is conservation of flow. Conservation of flow means that the total flow in arcs directed toward a node, plus the supply at the node, minus the demand at the node, equals the total flow in arcs directed away from the node.

A network and its associated data can be described in SAS data sets. PROC NETFLOW uses this description and finds the flow through each arc in the network that minimizes total cost of flow, meets the demand at demand nodes using the supply at supply nodes, so that the flow through each arc on or between the arc's lower flow bound and its capacity and satisfies the conservation of flow.

Network Modeling Techniques

There are many problems that can be represented using network models. One class of model is the production-inventory-distribution problem. The diagram in figure 1 illustrates an example. The subscript on the PROD, INVTNTRY, and SALES nodes indicate the time period. Notice that by replicating sections of the model the notion of time can be included. In this type of model, the nodes may represent a wide variety of facilities. Several which come to mind are: suppliers, spot markets, importers, farmers, manufacturers, factories, parts of plant, production lines, waste disposal facilities, workstations, warehouses, coolstores, depots, wholesalers, export markets, ports, rail junctions, airports, road intersections, cities, regions, shops, customers, consumers. The diversity of this selection demonstrates the richness of potential applications of this model.
Depending upon the interpretation of the nodes, the objectives of the modeling exercise may vary widely. Some common types of objectives are:

- Reduce collection or purchase costs of raw materials.
- Reduce inventory holding or backorder costs. Warehouses and other storage facilities sometimes have capacities and there may be limits on the amount of goods that can be placed on backorder.
- Decide where facilities should be located, what their capacity should be. Network models have been used to help decide where factories, hospitals, ambulance and fire stations, oil and water wells and schools should be sited.
- Determine the assignment of resources (machines, production capability, workforce) to tasks, schedules, classes or files can be a major objective.
- Determine the optimal distribution of goods or services. This usually means minimizing transportation costs, reducing time in transit or distances covered.
- Find the shortest path from one location to another.
- Ensure demands (e.g., production requirements, market demands, contractual obligations) are met.
- Maximize profits from the sale of products or the charge for services.
- Maximize production by identifying bottlenecks.

Some specific applications are:

- A car distribution model used to determine which models and the number of cars that should be manufactured in which factory and then to distribute cars from these factories to zones in the United States in order to meet customer demand at least cost.
- Models in the timber industry are common. These help determine when to plant and mill forests, schedule production of pulp, paper and wood products, distribute for sale or export.

- In military applications, the nodes may be theatres, bases, ammunition dumps, logistical suppliers or radar installations. Some models are used to find the best way to mobilise personnel and supplies and how to evacuate the wounded in the least amount of time.

- In communications applications, the nodes may be telephone exchanges, transmission lines, satellite links and consumers. In a model of an electrical grid, the nodes may be transformers, powerstations, watersheds, reserviors, dams and consumers. Of concern might be the effect of high loads or outages.

Side Constraints

Often all the details of a problem cannot be specified in a network model alone. In many of these cases this detail can be represented by the addition of side constraints to the model. Side constraints are a linear function of arc variables (variables containing flow through an arc) and nonarc variables (variables that are not part of the network). This enhancement to the basic network model allows for very general problems. In fact, any linear program can be represented with network models having these types of side constraints. The examples that follow will help to clarify the notion of side constraint.

PROC NETFLOW allows you to specify side constraints. The data for a side constraint consists of coefficients of arcs and coefficients of nonarc variables, a constraint type (i.e. \(\leq\) or \(\geq\)) and a right hand side value (rhs). A nonarc variable has a name, objective function coefficient, analogous to an arc cost, upper bound, analogous to an arc capacity and a lower bound, analogous to an arc lower flow bound. PROC NETFLOW finds the flow through the network and the values of any nonarc variables that minimize the total cost of the solution. Flow conservation will be met, flow through each arc will be on or between the arc's lower flow bound and capacity, the value of each nonarc variable will be on or between the nonarc's lower and upper bounds, and the side constraints will be satisfied. Note that since many linear programs have large embedded networks PROC NETFLOW is an attractive alternative to PROC LP in many cases.

To specify arcs in side constraints they must be named. By default, PROC NETFLOW names arcs using the names of the nodes at the head and tail of the arc. An arc is named with its tail node name followed by an "_" followed by the name of its head node name. For example, an arc from node from to node to is called from.to.

Proportional Constraints

Side constraints in network models fall into several categories that have special structure. They are frequently used when the flow through an arc must be proportional to the flow through another arc. Such constraints are called proportional constraints and are useful in models where production is subject to refining or modification into different materials. The amount of each output, or any waste, evaporation or reduction can be specified as a proportion of input.

Typically the arcs near the supply nodes carry raw materials and the arcs near the demand nodes carry refined products. For example, in a model of the milling industry, the flow through some arcs may represent quantities of wheat. After the wheat is processed, the flow through other arcs might be flour. For others it might be bran. The side constraints model the relationship between the amount of flour or bran produced as a proportion of the amount of wheat milled. Some of the wheat may end up as neither flour, bran or any useful product, so this waste is drained away via arcs to a waste node. Consider the network fragment in figure 2. The arc Wheat.Mill conveys the wheat milled. The cost of flow on this arc is the milling cost. The capacity of this arc is the capacity of the mill. The lower flow bound on this arc is the minimum quantity that must be milled for the mill to operate economically. The constraints

\[
0.3 \text{ wheat.mill - mill.four} = 0.0 \\
0.2 \text{ wheat.mill - mill.bran} = 0.0
\]

force every unit of wheat that is milled to produce 0.3 units of flour and 0.2 units of bran. Note that it is not necessary to specify the constraint.

99
Fig. 2 Proportional Constraint Example

0.5 * wheat * mill - mill * other = 0.0

since flow conservation implies that any flow that does not traverse through mill * flour or mill * bran must be conveyed through mill * other. And, computationally it is better if this constraint is not specified, since there will be one less side constraint, and fewer problems with numerical precision. Notice that the sum of the proportions must equal 1.0 exactly, otherwise flow conservation will be violated.

Blending Constraints

Blending or quality constraints may also influence the recipes or proportions of ingredient that are mixed. For example, different raw materials may have different properties. In an application of the oil industry, the amount of products that are obtained could be different for each type of crude oil. Furthermore, fuel might have a minimum octane requirement, or limited sulphur or lead content, so that a blending of crudes is needed to produce the product.

The network fragment in figure 3 shows an example of this. The arcs MidEast * Port and USA * Port convey crude oil from the two sources. The arc Port * Refinery represents refining while the arcs Refinery * Gasoline and Refinery * Diesel carry the gas and diesel produced. The proportional constraints

0.4 * Port * Refinery - Refinery * Gasoline = 0.0
0.2 * Port * Refinery - Refinery * Diesel = 0.0

capture the restrictions for producing gasoline and diesel from crude. Suppose that if only crude from the Middle East is used, the resulting diesel would contain 5 units of sulphur per litre. If only crude from the USA is used, the resulting diesel would contain 4 units of sulphur per litre. Diesel can have at most 4.75 units of sulphur per litre. Some crude from the USA must be used if Middle East crude is used in order to meet the 4.75 sulphur per litre limit. The side constraint to model this requirement is

5 * MidEast * Port + 4 * USA * Port - 4.75 * Port * Refinery ≤ 0.0 .

Since Port * Refinery = MidEast * Port + USA * Port, flow conservation allows this constraint can be simplified to

1 * MidEast * Port - 3 * USA * Port ≤ 0.0 .

If, for example, 120 units of crude from the Middle East is used, then at least 40 units of crude from the USA must be used. We have simplified the above by assuming that the sulphur concentration of diesel is proportional to the sulphur concentration of the crude mix. If this is not the case, the relation
Multi-commodity Problems

Side constraints are also used in models in which there are capacities on transportation or some other shared resource, or limits on overall production or demand in multi-commodity, multi-divisional or multi-period problems. Each commodity, division or period may have a separate network coupled to one main system by the side constraints. Side constraints are used to combine the outputs of subdivisions of a problem (either commodities, outputs in distinct time periods or different process streams) to meet overall demands, or to limit overall production or expenditures. Figure 4 shows two network fragments. They represent identical production and distribution sites but of two different commodities. Suffix \texttt{com1} represents commodity 1 and suffix \texttt{com2} represents commodity 2. The nodes \texttt{Factorycom1} and \texttt{Factorycom2} model the same factory and nodes \texttt{City1com1} and \texttt{City1com2} model the same location, city1. Similarly, \texttt{City2com1} and \texttt{City2com2} are the same location, city2. Suppose that commodity 1 occupies 2 cubic metres, commodity 2 occupies 3 cubic metres and the truck despatched to city1 has a capacity of 200 cubic metres, the truck despatched to city2 has a capacity of 250 cubic metres. How much of each commodity can be loaded onto each truck? The side constraints for this case are:

\begin{align*}
&2 \texttt{ Factorycom1.City1com1} + 3 \texttt{ Factorycom2.City1com2} \leq 200 \\
&2 \texttt{ Factorycom1.City2com1} + 3 \texttt{ Factorycom2.City2com2} \leq 250 
\end{align*}

Large Modeling Strategy

In many cases, the flow through an arc might actually represent the flow or movement of a commodity from place to place or from time period to time period. However, sometimes an arc is included in the network as
Fig. 4 Multi-Commodity Example

A method of capturing some aspect of the problem that one would not normally think of as part of a network model. For example, in a multi-process multi-product model, there might be subnetworks for each process and each product. The subnetworks may be joined together by a set of arcs that have flows that represent the amount of product \( j \) produced by process \( i \). To model an upper limit constraint on the total amount of product \( j \) that can be produced, direct all arcs carrying product \( j \) to a single node and from there through a single arc. The capacity of this arc is the upper limit of product \( j \) production. It is preferable to model this structure in the network rather than to include it in the side constraints because the efficiency of the optimizer is affected less by a reasonable increase in the size of the network. It is often a good strategy when starting a project to use a small network formulation and then to use that model as a framework upon which to add detail. For example, in the multi-process multi-product model, you might start with the network depicted in figure 5. Then, for example, the process subnetwork can be enhanced to include the distribution of products. Other phases of the operation could be included by adding more subnetworks. Initially, these subnetworks may be single nodes, but in subsequent studies they can be expanded to include greater detail.
Fig. 5 Multi-Process, Multi-Product Example

Process 1 subnetwork

Product 1 subnetwork

Process 2 subnetwork

Product 2 subnetwork
More on Nonarc Variables

In some models, nonarc variables are used in constraints to absorb excess resources or supply needed resources. Then, either the excess resource could be used or the needed resource can be supplied to another component of the model.

For example, consider a multi-commodity problem of making television sets that have either 19 or 25 inch screens. In their manufacture, 3 and 4 chips respectively are used. Production occurs at 2 factories during March and April. The supplier of chips can supply only 2600 chips to factory 1 and 3750 chips to factory 2 each month. The names of arcs are in the form 'Prod..n..m', where n is the factory number, s is the screen size, and m is the month. For example, Prod1..25..Apr is the arc that conveys the number of 25 inch TVs produced in factory 1 during April. You might have to determine similar systematic naming schemes for your application. As described, the constraints are:

\[
\begin{align*}
3 \text{ Prod1.19.Mar} + 4 \text{ Prod1.25.Mar} & \leq 2600 \\
3 \text{ Prod2.19.Mar} + 4 \text{ Prod2.25.Mar} & \leq 3750 \\
3 \text{ Prod1.19.Apr} + 4 \text{ Prod1.25.Apr} & \leq 2600 \\
3 \text{ Prod2.19.Apr} + 4 \text{ Prod2.25.Apr} & \leq 3750 \\
\end{align*}
\]

If there are chips that could be obtained for use in March but not used for production in March then why not keep these unused chips until April? Furthermore, if the March excess chips at factory 1 could be used either at factory 1 or factory 2 in April then the model becomes:

\[
\begin{align*}
3 \text{ Prod1.19.Mar} + 4 \text{ Prod1.25.Mar} + F1.\text{Unused.Mar} & = 2600 \\
3 \text{ Prod2.19.Mar} + 4 \text{ Prod2.25.Mar} + F2.\text{Unused.Mar} & = 3750 \\
3 \text{ Prod1.19.Apr} + 4 \text{ Prod1.25.Apr} - F1.\text{Kept.Since.Mar} & = 2600 \\
3 \text{ Prod2.19.Apr} + 4 \text{ Prod2.25.Apr} - F2.\text{Kept.Since.Mar} & = 3750 \\
F1.\text{Unused.Mar} + F2.\text{Unused.Mar} - F1.\text{Kept.Since.Mar} - F2.\text{Kept.Since.Mar} & \geq 0.0 \\
\end{align*}
\]

where F1.Kept.Since.Mar is the number of chips used during April at factory 1 that were obtained in March at either factory 1 or factory 2 and F2.Kept.Since.Mar is the number of chips used during April at factory 2 that were obtained in March. The last constraint ensures that the number of chips used during April that were obtained in March does not exceed the number of chips not used in March. There may be a cost to hold chips in inventory. This can be modeled having a positive objective function coefficient for the nonarc variables F1.Kept.Since.Mar and F2.Kept.Since.Mar. Moreover, nonarc variable upper bounds represent an upper limit on the number of chips that can be held in inventory between March and April.

Conclusion

In this section we have exposed the reader to a variety of potential applications and modeling techniques. We have demonstrated that the variety and flexibility of the network modeling techniques makes this approach valuable in many settings.

The Performance of PROC NETFLOW

PROC NETFLOW was designed to solve large problems fast. Problems encountered in industry can be enormous, often by virtue that they model multi-commodity, multi-process and multi-time period scenarios. Network models with thousands of arcs, nodes and side constraints are common. Because of this, options that tune the optimiser and control memory management in PROC NETFLOW can be very important in reducing execution time. Several hints that can improve performance are discussed in this section.

The BYTES and COREFACTOR parameters of the PROC NETFLOW statement can be used to control the type of memory management scheme used. The BYTES parameter is especially important as this indicates the amount of work memory NETFLOW can use. If too small a value is specified, PROC NETFLOW may have to do data transfer between core and disk. This may not have been necessary if a larger BYTES value had been specified. Data transfers can be very time consuming. Too large a BYTES parameter can also slow the NETFLOW procedure on some machines. Experimentation for a given problem type and machine type can lead to satisfactory results. Specifying the MEMREP option in the PROC NETFLOW
statement cause messages to appear on the SAS log informing you whether the problem was solved using in-core or out-of-core routines.

In general, to decrease execution time:

- have names for arcs only if the arc name appears in the CONDATA=SAS data set.
- use NAMECTRL=1 or NAMECTRL=3 if applicable.
- use default arc names from_to.
- move observations that indicate constraint type to the top of the CONDATA= data set,
- limit the length of node, arc, nonarc variable and row names.
- use warm starts whenever possible.
- use the SAME NONARC DATA option described below.

Optimization times are also significantly affected by the choice of pricing strategy. Experiment with the options that control pricing strategies: try PRICETYPEx=Q and PRICETYPEx=NOQ; try changing QSIZEx; experiment with the BIGMx and TWOPHASEx options. See the SAS/OR User's Guide for details on the use of these options. BIGMx and TWOPHASEx are new options described in the next section.

Undocumented or Changed options in versions 6.04 and 6.06

Changes to existing options, PROC NETFLOW statement

SCALE is no longer a simple option that indicates whether side constraint rows are to be scaled, so that the absolute value of the largest coefficient in any row of the side constraint coefficient matrix is near to 1. PROC NETFLOW can now also scale nonarc variable columns so that the absolute value of the largest constraint coefficient of a nonarc variable is near to 1. Specify SCALE=ROW or CON or CONSTRAINT, SCALE=COL or COLUMN or NONARC, SCALE=BOTH, or SCALE=NONE. The default is SCALE=BOTH.

SAVE saves the ARCOUT, CONOUT, NODEOUT, DUALOUT data sets to whatever data sets are named whenever the statement is invoked. Consider:

```
PROC NETFLOW options; lists;
RESET MAXIT1=10 MAXIT2=25
ARCOUT=ARCOUT1 NODEOUT=NODEOUT1
CONOUT=CONOUT1 DUALOUT=DUALOUT1;
RUN;
   /* Stage 1 optimization stops after iteration 10. */
   /* No output data sets are created yet. */
   SAVE ARCOUT=ARCOUT1 NODEOUT=NODEOUT1;
   /* ARCOUT1 and NODEOUT1 are created. */
RESET ARCOUT=ARCOUT2 MAXIT1=999999;
RUN;
   /* The stage 1 optimum is reached. */
   /* ARCOUT2 and NODEOUT2 are created. */
   /* ARCOUT2 is not created as ARCOUT=ARCOUT2 over- */
   /* rides the ARCOUT=ARCOUT2 specified earlier. */
   /* Stage 2 optimization stops after 25 iterations */
   /* as MAXIT2=25 was specified. */
   SAVE CONOUT=CONOUT1;
   /* CONOUT1 is created. */
RESET MAXIT2=999999 DUALOUT=NULL;
RUN;
```
The stage 2 optimum is reached. /*
CONOUT is created. */
No DUALOUT is created as the last NETFLOW or */
RESET DUALOUT=data set specification was */
DUALOUT=NULL. */

The data sets specified in the PROC NETFLOW and RESET statements are created when an optimal solution is found. The data sets specified in SAVE statements are created immediately.

The data sets in the example above are all distinct, but this need not be the case. The only exception to this is that the ARCOOUT= and NODEOUT= data sets, or the CONOUT= and DUALOUT= data sets that are being created at the same time must be distinct. Use the SHOW DATASETS; statement to examine what data sets are current and which have already been created and when.

New options on the PROC NETFLOW statement

SAME_NONARC_DATA or SND If all nonarc variable data is given in the ARCDATA=data set, or the problem has no nonarc variables, the unconstrained warm start can be read more quickly if the option SAME_NONARC_DATA is specified. SAME_NONARC_DATA indicates that any non-constraint nonarc variable data in CONDATA=data set is to be ignored. Only side constraint data in CONDATA=data set will be read.

When using an unconstrained warm start and SAME_NONARC_DATA is not specified, any nonarc variable objective function coefficient, upper and lower bound can be changed. Any nonarc variable data in CONDATA=data set overrides (without WARNING messages) corresponding data in the ARCDATA=data set. You can possibly introduce new nonarc variables to the problem, i.e. nonarc variables that were not in the problem when the warm start was generated.

SAME_NONARC_DATA should be specified if nonarc variable data in CONDATA=data set is to be deliberately ignored. Consider:

PROC NETFLOW options ARCDATA=ARCO NODEDATA=NODEO
CONDATA=CONO
/* this data set has nonarc variable objective */
/* function coefficient data */
RUN;
DATA ARC2;
SET ARC1; /* this data set has nonarc variable obs */
IF _COST_ < 60.0 THEN _COST_ = _COST_ * 1.25; /* some objective coefficients of nonarc */
/* variable might be changed */
PROC NETFLOW options
WARM ARCDATA=ARC2 NODEDATA=NODE1
CONDATA=CONO SAME_NONARC_DATA
/* This data set contains "old" nonarc variables */
/* obj, fn. coefficients. SAME_NONARC_DATA */
/* indicates that the "new" coefs in ARCDATA */
/* are to be used. */
RUN;

MAXARRAYBYTES=m specifies the maximum number of bytes an individual array can occupy. This option is of most use when solving large problems and the amount of available memory is insufficient to store all arrays at once. Specifying MAXARRAYBYTES ensures that arrays that need lots of memory do not consume too much memory at the expense of other arrays.

There is one array which contains information about nodes and the network basis spanning tree description. This tree description enables computations involving the network part of the basis to be
performed very quickly and is the reason why PROC NETFLOW is more suited to solving constrained network problems than PROC LP. It is beneficial that this array be stored in core when possible, otherwise this array must be paged, slowing down the computations. Try not to specify a MAXARRAYBYTES=m value smaller than the amount of memory needed to store the main node array. You will be told what this memory amount is on the SAS log if you specify MEMREP in the PROC NETFLOW statement.

BYPASSDIVIDE or BYPASSDIV or BPD When MAXFLOW is specified, a bypass arc is set up between the SOURCENODE and SINKNODE. This arc will convey flow equal to INFINITY minus the maximal flow through the network. The cost of the bypass arc must be expensive to drive flow through the network, rather than through the bypass arc. However, the cost of the bypass arc must be less than the cost of artificial variables (otherwise these might have nonzero optimal value and a false infeasibility error will result). Also, the cost of the bypass arc must be greater than the eventual total cost of the maximal flow, which may be nonzero if some network arcs have nonzero costs. The cost of the bypass arc will be set to INFINITY/BYPASSDIV. Valid values for BYPASSDIV must be greater than or equal to 1.1.

If there are no nonzero costs of arcs in the MAXFLOW problem, the cost of the bypass arc is set to 1.0 (-1.0 if maximizing) if you do not specify BYPASSDIV. The reduced costs in ARCOUT and CONOUT=data set will correctly reflect the value that would be added to the maximal flow if the capacity of the arc is increased by one unit. If there are nonzero costs, or you specify BYPASSDIV, the reduced costs may be contaminated by the cost of the bypass arc and no economic interpretation can be given to reduced cost values. The default value for BYPASSDIV is 100.0.

New options of the RESET statement

BIGMx or TWOPHASEx x is the character 1 if the options control what happens during optimization of the problem when side constraints are relaxed. x is the character 2 if the options control what happens during optimization considering the side constraints.

BIGMx indicates that the big-M approach to optimization will be used (artificial variables are treated like real arcs, slacks, surpluses and nonarc variables. Artificials have very expensive costs).

TWOPHASEx indicates that the two-phase approach will be used. (At first, artificial variables are the only variables to have nonzero objective function coefficients. An artificial objective function coefficient is 1 and PROC NETFLOW minimizes. When all artificial variables have zero value, phase 2 commences during which arcs and nonarc variables have their real costs and objective function coefficients.

TWOPHASE2 is often better than BIGM2 when the problem has many side constraints.

Before all artificial variables are driven to have zero value, you may toggle between the big-M and two-phase approaches by specifying BIGMx or TWOPHASEx in a RESET statement. NOTWOPHASEx is synonymous with BIGMx, NOBIGMx with TWOPHASEx. The default is BIGMx.

CYClemULT1=c or MINBLOCK1=m or PERTURB1 or NOPERTURB1 In an effort to reduce the number of iterations performed when the problem is highly degenerate, PROC NETFLOW has in stage 1 optimization an algorithm outlined in "On the Solution of Highly Degenerate Linear Programmes" Ryan D.M. Osborne M.R. Mathematical Programming 41 (1988) 385-392.

If the number of consecutive degenerate pivots (those with no progress toward the optimum) are performed equals CYClemULT1 times number of nodes, the arcs that were "blocking" (can leave the basis) are put onto a list. In subsequent iterations, of the arcs that now can leave the basis, the one chosen to leave is an arc on the list of arcs that could have left in the previous iteration. i.e. preference is given to arcs that "block" many iterations.

If the number of blocking arcs is less than MINBLOCK1=m, a list is not kept. Otherwise, if PERTURB1 is specified, the arc flows are perturbed by a random quantity, so arcs on the list that block subsequent iterations are randomly chosen to leave the basis. Although perturbation often pays off, it
is computationally expensive. You can specify NOPERTURB1 to prevent this occurring. The defaults are CYCLEMULT1=0.15 MINBLOCK1=2 NOPERTURB1.

MOREOPT If PROC NETFLOW determines that the problem is infeasible, it will not do any more optimization unless you specify MOREOPT in a RESET statement. At the same time, you may try resetting parameters (particularly zero tolerances) in the hope that the infeasibility was raised incorrectly.

If PROC NETFLOW finds an optimal solution, you might want to do additional optimization to confirm that an optimum has really been reached. MOREOPT turns off all optimality and infeasibility flags that may have been raised. Unless this is done, PROC NETFLOW does not do any optimization when a RUN statement is specified. Consider:

```
proc netflow
   nodedata=noded   /* the supply and demand data */
   arcdata=arcd1    /* the arc descriptions */
   condata=cond1    /* the side constraints */
   conout=solution; /* the solution */
run;
/* NETFLOW states that the problem is infeasible. */
/* You suspect that the zero tolerance is too large */
reset zero2=1.0e-10 moreopt;
run;
/* Netflow will attempt more optimization. */
/* After this, if it reports that the problem is infeasible, */
/* the problem really might be infeasible */
```