Chapter 4

The NETFLOW Procedure

Chapter Table of Contents

Chapter 4
The NETFLOW Procedure

Overview

Introduction

Constrained network models can be used to describe a wide variety of real-world applications ranging from production, inventory, and distribution problems to financial applications. These problems can be solved with the NETFLOW procedure.

These models are conceptionally easy since they are based on network diagrams that represent the problem pictorially. PROC NETFLOW accepts the network specification in a format that is particularly suited to networks. This not only simplifies problem description but also aids in the interpretation of the solution.

Certain algebraic features of networks are exploited by a specialized version of the Simplex method so that solution times are reduced. Another class of optimization algorithm, the Interior-Point algorithm, has been implemented in PROC NETFLOW and can be used as an alternative to the Simplex algorithm to solve network problems.

Should PROC NETFLOW detect there are no arcs and nodes in the model's data, (that is, there is no network component), it assumes it is dealing with a linear programming (LP) problem. The Interior Point algorithm is automatically elected to perform the optimization. PROC NETFLOW's ability to solve LP problems is an important ability new for the Version 7 release of the SAS System.

Network Models

A network consists of a collection of nodes joined by a collection of arcs. The arcs connect nodes and convey flow of one or more commodities that 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 a 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 the total cost of flow, meets the demand at demand nodes using the supply at supply nodes so that the flow through each arc is on or between the arc's lower flow bound and its capacity, and satisfies the conservation of flow.

One class of network model is the production-inventory-distribution problem. The diagram in Figure 4.1 illustrates this problem. The subscript on the PROD, INVN-

TRY, and SALES nodes indicates the time period. Notice that if you by replicate sections of the model, the notion of time can be included.

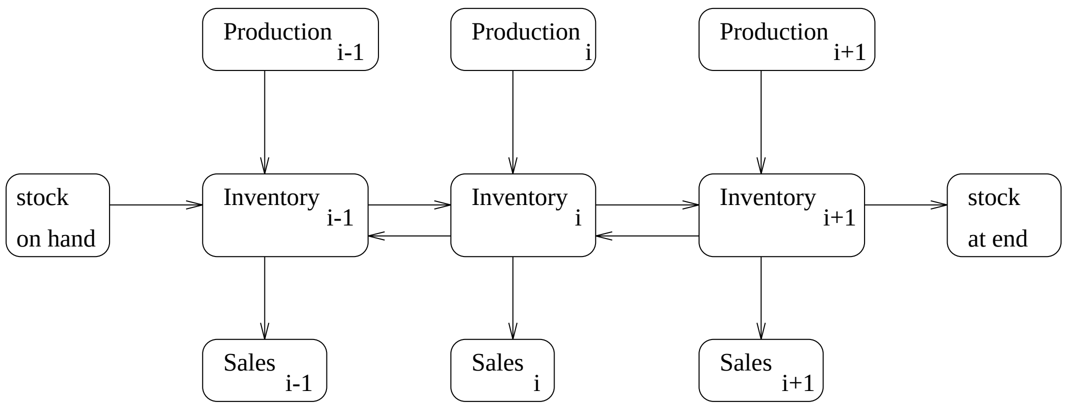


Figure 4.1. Production-Inventory-Distribution Problem

In this type of model, the nodes can represent a wide variety of facilities. Several examples are 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, and 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 can vary widely. Some common types of objectives are:

- to reduce collection or purchase costs of raw materials
- to reduce inventory holding or backorder costs. Warehouses and other storage facilities sometimes have capacities, and there can be limits on the amount of goods that can be placed on backorder
- to decide where facilities should be located and what the capacity of these 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
- to determine the assignment of resources (machines, production capability, workforce) to tasks, schedules, classes, or files
- to determine the optimal distribution of goods or services. This usually means minimizing transportation costs, and reducing time in transit or distances covered

- to find the shortest path from one location to another
- to ensure that demands (for example, production requirements, market demands, contractural obligations) are meet
- to maximize profits from the sale of products or the charge for services
- to maximize production by identifying bottlenecks

Some specific applications are

- car distribution models. These help determine which models and numbers of cars should be manufactured in which factories and where 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. These help determine when to plant and mill forests, schedule production of pulp, paper and wood products, and distribute products for sale or export.
- military applications. The nodes can be theatres, bases, ammunition dumps, logistical suppliers, or radar installations. Some models are used to find the best ways to mobilize personnel and supplies and to evacuate the wounded in the least amount of time.
- communications applications. The nodes can be telephone exchanges, transmission lines, satelite links, and consumers. In a model of an electrical grid, the nodes can be transformers, powerstations, watersheds, reservoirs, 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, these details 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 help to clarify the notion of side constraints.

PROC NETFLOW enables you to specify side constraints. The data for a side constraint consist of coefficients of arcs and coefficients of nonarc variables, a constraint type (that is, $\leq$, =, or $\geq$) and a right-hand-side value (rhs). A nonarc variable has a name, an objective function coefficient analogous to an arc cost, an 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 is met, flow through each arc is on or between the arc's lower flow bound and capacity, the value of each nonarc variable is on or between the nonarc's lower and upper bounds, and the side constraints are satisfied. Note that, since many linear programs have large embedded networks, PROC NETFLOW is an attractive alternative to the LP procedure in many cases.

In order for arcs to be specified 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 can end up as neither flour, bran, nor any useful product, so this waste is drained away via arcs to a waste node.

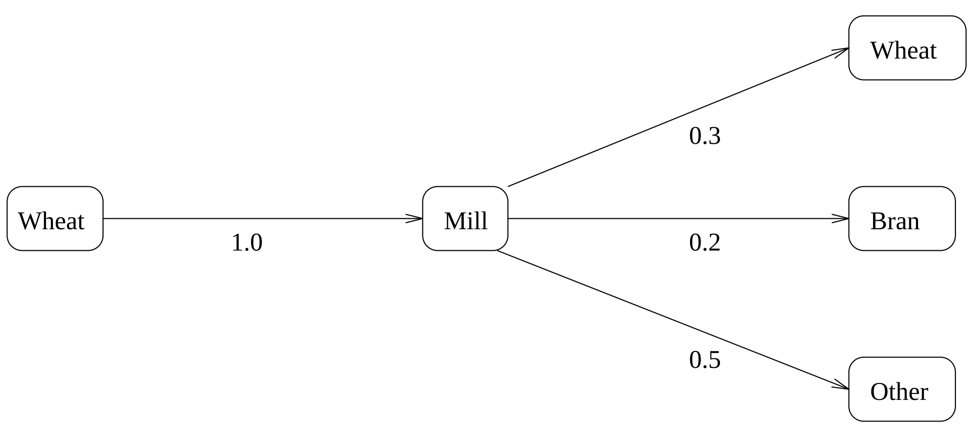


Figure 4.2. Proportional Constraints

Consider the network fragment in Figure 4.2. 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 Wheat_Mill - Mill_Flour = 0.0
0.2 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

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 is 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 is violated.

Blending Constraints

Blending or quality constraints can also influence the recipes or proportions of ingredients that are mixed. For example, different raw materials can 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 4.3 shows an example of this.

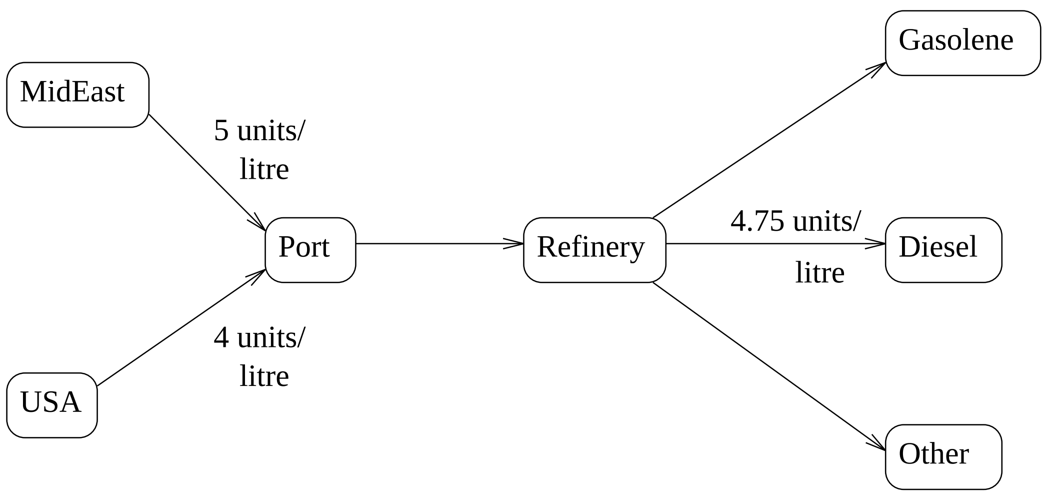


Figure 4.3. Blending Constraints

The arcs MidEast_Port and USA_Port convey crude oil from the two sources. The arc Port_Refinery represents refining while the arcs Refinery_Gasolene and Refinery_Diesel carry the gas and diesel produced. The proportional constraints

0.4 Port_Refinery - Refinery_Gasolene = 0.0
0.2 Port_Refinery - Refinery_Diesel = 0.0

capture the restrictions for producing gasolene 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

$$5 \text{ MidEast\_Port} + 4 \text{ USA\_Port} - 4.75 \text{ Port\_Refinery} \leq 0.0$$

Since Port_Refinery = MidEast_Port + USA_Port, flow conservation allows this constraint to be simplified to

$$1 \text{ MidEast\_Port} - 3 \text{ USA\_Port} \leq 0.0$$

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. The preceding constraint is simplified because you assume that the sulphur concentration of diesel is proportional to the sulphur concentration of the crude mix. If this is not the case, the relation

$$0.2 \text{ Port\_Refinery} = \text{Refinery\_Diesel}$$

is used to obtain

$$5 \text{ MidEast\_Port} + 4 \text{ USA\_Port} - 4.75 \ (\ 1.0/0.2 \text{ Refinery\_Diesel} \ ) \leq 0.0$$

which equals

$$5 \text{ MidEast\_Port} + 4 \text{ USA\_Port} - 23.75 \text{ Refinery\_Diesel} \leq 0.0$$

An example similar to this Oil Industry problem is solved in the “Introductory Example” section on page 197.

Multicommodity Problems

Side constraints are also used in models in which there are capacities on transportation or some other shared resource, or there are limits on overall production or demand in multicommodity, multidivisional or multiperiod problems. Each commodity, division or period can 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. This method is more desirable than doing separate *local* optimizations for individual commodity, process, or time networks and then trying to establish relationships between each when determining an overall policy if the *global* constraint is not satisfied. Of course, to make models more realistic, side constraints may be necessary in the local problems.

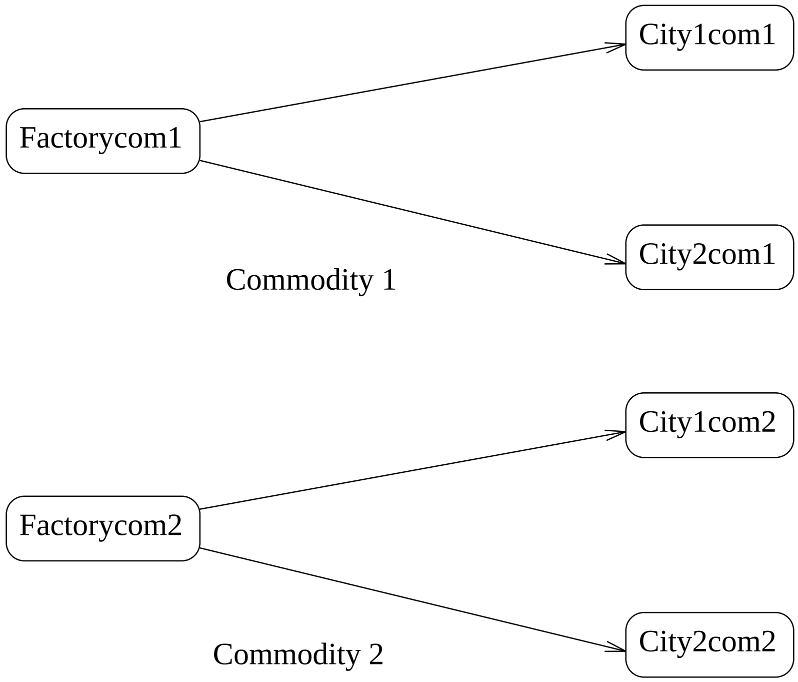


Figure 4.4. Multicommodity Problem

Figure 4.4 shows two network fragments. They represent identical production and distribution sites but of two different commodities. Suffix *com1* represents commodity 1 and suffix *com2* represents commodity 2. The nodes Factorycom1 and Factorycom2 model the same factory, and nodes City1com1 and City1com2 model the same location, city1. Similarly, City2com1 and City2com2 are the same location, city2. Suppose that commodity 1 occupies 2 cubic meters, commodity 2 occupies 3 cubic meters, the truck dispatched to city1 has a capacity of 200 cubic meters, and the truck dispatched to city2 has a capacity of 250 cubic meters. How much of each commodity can be loaded onto each truck? The side constraints for this case are

$$2\ \text{Factorycom1\_City1com1} + 3\ \text{Factorycom2\_City1com2} \leq 200$$
$$2\ \text{Factorycom1\_City2com1} + 3\ \text{Factorycom2\_City2com2} \leq 250$$

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 a method of capturing some aspect of the problem that you would not normally think of as part of a network model. For example, in a multiprocess multiproduct model (Figure 4.5), there might be subnetworks for each process and each product. The subnetworks can 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.

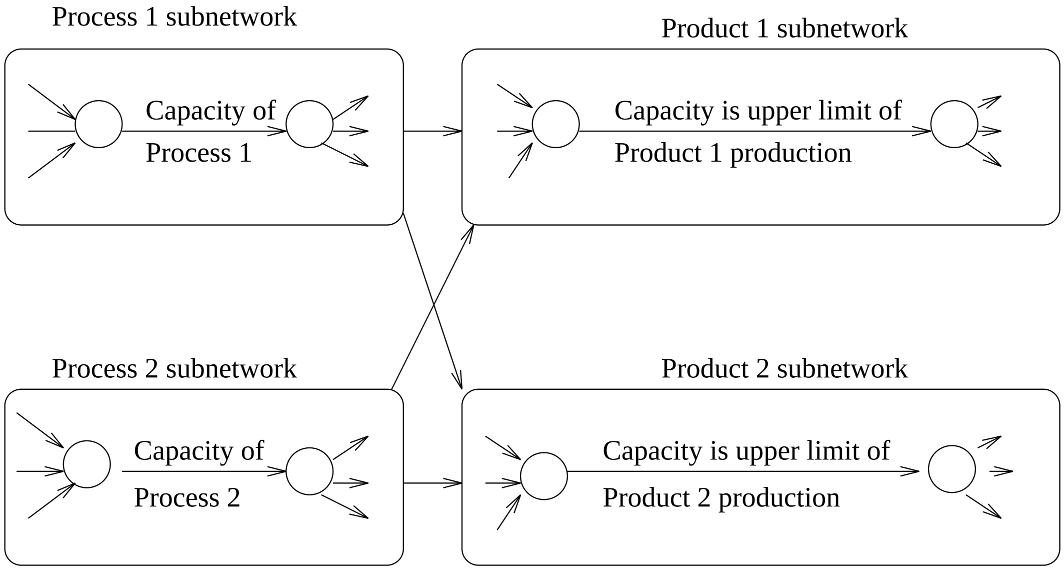


Figure 4.5. Multiprocess, Multiproduct Example

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 multiprocess multiproduct model, you might start with the network depicted in Figure 4.5. 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 can be single nodes, but in subsequent studies they can be expanded to include greater detail.

The NETFLOW procedure accepts the side constraints in the same dense and sparse formats that the LP procedure provides. Although PROC LP can solve network problems, the NETFLOW procedure generally solves network flow problems more efficiently than PROC LP.

Advantages of Network Models over LP Models

Many linear programming problems have large embedded network structures. Such problems often result when modeling manufacturing processes, transportation or distribution networks, or resource allocation, or when deciding where to locate facilities. Often, some commodity is to be moved from place to place, so the more natural formulation in many applications is that of a constrained network rather than a linear program.

Using a network diagram to visualize a problem makes it possible to capture the important relationships in an easily understood picture form. The network diagram aids the communication between model builder and model user, making it easier to comprehend how the model is structured, how it can be changed, and how results can be interpreted.

If a network structure is embedded in a linear program, the problem is a network programming problem with side constraints (NPSC). When the network part of the problem is large compared to the nonnetwork part, especially if the number of side constraints is small, it is worthwhile to exploit this structure in the solution process. This is what PROC NETFLOW does. It uses a variant of the Revised Primal Simplex algorithm that exploits the network structure to reduce solution time.

Mathematical Description of NPSC

If a network programming problem with side constraints has n nodes, a arcs, g nonarc variables, and k side constraints, then the formal statement of the problem solved by PROC NETFLOW is

$$min\{c^T x + d^T z\}$$

subject to

$$Fx = b$$
$$Hx + Qz \geq, =, \leq r$$
$$l \leq x \leq u$$
$$m \leq z \leq v$$

where

c is the a x 1 objective function coefficient of arc variables vector (the cost vector)

x is the a x 1 arc variable value vector (the flow vector)

d is the g x 1 objective function coefficient of nonarc variables vector

z is the g x 1 nonarc variable value vector

F is the n x a node-arc incidence matrix of the network, where

$F_{i,j} = -1$	if arc j is directed toward node i
$F_{i,j} = 1$	if arc j is directed from node i
$F_{i,j} = 0$	otherwise

b is the n x 1 node supply/demand vector, where

$b_i = s$	if node i has supply capability of s units of flow
$b_i = -d$	if node i has demand d of units of flow
$b_i = 0$	if node i is a transshipment node

H is the k x a side constraint coefficient matrix for arc variables, where $H_{i,j}$ is the coefficient of arc j in the ith side constraint

Q is the k x g side constraint coefficient matrix for nonarc variables, where $Q_{i,j}$ is the coefficient of nonarc j in the ith side constraint

r is the k x 1 side constraint right-hand-side vector

l is the a x 1 arc lower flow bound vector

u is the a x 1 arc capacity vector

m is the g x 1 nonarc variable value lower bound vector

v is the g x 1 nonarc variable value upper bound vector

Flow Conservation Constraints

The constraints $Fx = b$ are referred to as the nodal flow conservation constraints. These constraints algebraically state that the sum of the flow through arcs directed toward a node plus that node's supply, if any, equals the sum of the flow through arcs directed away from that node plus that node's demand, if any. The flow conservation constraints are implicit in the network model and should not be specified explicitly in side constraint data when using PROC NETFLOW. The constrained problems most amenable to being solved by the NETFLOW procedure are those that, after the removal of the flow conservation constraints, have very few constraints. PROC NETFLOW is superior to linear programming optimizers when the network part of the problem is significantly larger than the nonnetwork part.

The NETFLOW procedure can also be used to solve an unconstrained network problem, that is, one in which H, Q, d, r, and z do not exist.

Nonarc Variables

If the constrained problem to be solved has no nonarc variables, then Q, d, and z do not exist. However, nonarc variables can be used to simplify side constraints. For example, if a sum of flows appears in many constraints, it may be worthwhile to equate this expression with a nonarc variable and use this in the other constraints. By assigning a nonarc variable a nonzero objective function, it is then possible to incur a cost for using resources above some lowest feasible limit. Similarly, a profit (a negative objective function coefficient value) can be made if all available resources are not used.

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

For example, consider a multicommodity 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 factory1 and 3750 chips to factory2 each month. The names of arcs are in the form Prod*n*_*s*_*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

$$3\ \text{Prod1\_19\_Mar} + 4\ \text{Prod1\_25\_Mar} \leq 2600$$
$$3\ \text{Prod2\_19\_Mar} + 4\ \text{Prod2\_25\_Mar} \leq 3750$$
$$3\ \text{Prod1\_19\_Apr} + 4\ \text{Prod1\_25\_Apr} \leq 2600$$
$$3\ \text{Prod2\_19\_Apr} + 4\ \text{Prod2\_25\_Apr} \leq 3750$$

If there are chips that could be obtained for use in March but not used for production in March, 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, the model becomes

$$3\ \text{Prod1\_19\_Mar} + 4\ \text{Prod1\_25\_Mar} + \text{F1\_Unused\_Mar} = 2600$$
$$3\ \text{Prod2\_19\_Mar} + 4\ \text{Prod2\_25\_Mar} + \text{F2\_Unused\_Mar} = 3750$$
$$3\ \text{Prod1\_19\_Apr} + 4\ \text{Prod1\_25\_Apr} - \text{F1\_Kept\_Since\_Mar} = 2600$$
$$3\ \text{Prod2\_19\_Apr} + 4\ \text{Prod2\_25\_Apr} - \text{F2\_Kept\_Since\_Mar} = 3750$$
$$\text{F1\_Unused\_Mar} + \text{F2\_Unused\_Mar}\ \text{(continued)}$$
$$- \text{F1\_Kept\_Since\_Mar} - \text{F2\_Kept\_Since\_Mar} \geq 0.0$$

where F1_Kept_Since_Mar is the number of chips used during April at factory1 that were obtained in March at either factory1 or factory2 and F2_Kept_Since_Mar is the number of chips used during April at factory2 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.

See Example 4.4 through Example 4.8 for a series of examples that use this TV problem. The use of nonarc variables as descriped previously is illustrated.

Warm Starts

If you have a problem that has already been partially solved and is to be solved further to obtain a better, optimal solution, information describing the solution now available may be used as an initial solution. This is called *warm starting* the optimization, and the supplied solution data are called the *warm start*.

Some data can be changed between the time when a warm start is created and when it is used as a warm start for a subsequent PROC NETFLOW run. Elements in the arc variable cost vector, the nonarc variable objective function coefficient vector, and sometimes capacities, upper value bounds, and side constraint data can be changed between PROC NETFLOW calls. See the “Warm Starts” section on page 292. Also, see Example 4.4 through Example 4.8 (the TV problem) for a series of examples that show the use of warm starts.

Getting Started

To solve network programming problems with side constraints using PROC NETFLOW, you save a representation of the network and the side constraints in three SAS data sets. These data sets are then passed to PROC NETFLOW for solution. There are various forms that a problem's data can take. You can use any one or a combination of several of these forms.

The NODEDATA=data set contains the names of the supply and demand nodes and the supply or demand associated with each. These are the elements in the column vector **b** in problem (NPSC).

The ARCDATA= data set contains information about the variables of the problem. Usually these are arcs, but there can be data related to nonarc variables in the ARCDATA= data set as well.

An arc is identified by the names of its tail node (where it originates) and head node (where it is directed). Each observation can be used to identify an arc in the network and, optionally, the cost per flow unit across the arc, the arc's capacity, lower flow bound, and name. These data are associated with the matrix F and the vectors c, l, and u in problem (NPSC).

Note: although F is a node-arc incidence matrix, it is specified in the ARCDATA= data set by arc definitions.

In addition, the ARCDATA= data set can be used to specify information about nonarc variables, including objective function coefficients, lower and upper value bounds, and names. These data are the elements of the vectors d, m, and v in problem (NPSC). Data for an arc or nonarc variable can be given in more than one observation.

Supply and demand data also can be specified in the ARCDATA= data set. In such a case, the NODEDATA= data set may not be needed.

The CONDATA= data set describes the side constraints and their right-hand-sides. These data are elements of the matrices H and Q and the vector r. Constraint types are also specified in the CONDATA= data set. You can include in this data set upper bound values or capacities, lower flow or value bounds, and costs or objective function coefficients. It is possible to give all information about some or all nonarc variables in the CONDATA= data set.

An arc is identified in this data set by its name. If you specify an arc's name in the ARCDATA= data set, then this name is used to associate data in the CONDATA= data set with that arc. Each arc also has a default name that is the name of the tail and head node of the arc concatenated together and separated by an underscore character; tail_head, for example.

If you use the dense side constraint input format (described in the "CONDATA= Data Set" section on page 267) and want to use these default arc names, these arc names are names of SAS variables in the VAR list of the CONDATA= data set.

If you use the sparse side constraint input format (see the “CONDATA= Data Set” section on page 267) and want to use these default arc names, these arc names are values of the COLUMN list SAS variable of the CONDATA= data set.

The execution of PROC NETFLOW has three stages. In the preliminary (zeroth) stage, the data are read from the NODEDATA= data set, the ARCDATA= data set, and the CONDATA= data set. Error checking is performed, and an initial basic feasible solution is found. If an unconstrained solution warm start is being used, then an initial basic feasible solution is obtained by reading additional data containing that information in the NODEDATA= data set and the ARCDATA= data set. In this case, only constraint data and nonarc variable data are read from the CONDATA= data set.

In the first stage, an optimal solution to the network flow problem neglecting any side constraints is found. The primal and dual solutions for this relaxed problem can be saved in the ARCOUT= data set and the NODEOUT= data set, respectively. These data sets are named in the PROC NETFLOW, RESET, and SAVE statements.

In the second stage, an optimal solution to the network flow problem with side constraints is found. The primal and dual solutions for this side constrained problem are saved in the CONOUT= data set and the DUALOUT= data set, respectively. These data sets are also named in the PROC NETFLOW, RESET, and SAVE statements.

If a constrained solution warm start is being used, PROC NETFLOW does not perform the zeroth and first stages. This warm start can be obtained by reading basis data containing additional information in the NODEDATA= data set (also called the DUALIN= data set) and the ARCDATA= data set.

If warm starts are to be used in future optimizations, the FUTURE1 and FUTURE2 options must be used in addition to specifying names for the data sets that contain the primal and dual solutions in stages one and two. Then, most of the information necessary for restarting problems is available in the output data sets containing the primal and dual solutions of both the relaxed and side constrained network programs.

Introductory Example

Consider the following transshipment problem for an oil company. Crude oil is shipped to refineries where it is processed into gasoline and diesel fuel. The gasoline and diesel fuel are then distributed to service stations. At each stage, there are shipping, processing, and distribution costs. Also, there are lower flow bounds and capacities.

In addition, there are two sets of side constraints. The first set is that two times the crude from the Middle East cannot exceed the throughput of a refinery plus 15 units. (The words “plus 15 units” that finishes the last sentence is used to enable some side constraints in this example to have a nonzero rhs.) The second set of constraints are necessary to model the situation that one unit of crude mix processed at a refinery yields three-fourths of a unit of gasoline and one-fourth of a unit of diesel fuel.

Because there are two products that are not independent in the way in which they flow through the network, a network programming problem with side constraints is an appropriate model for this example (see Figure 4.6). The side constraints are used to model the limitations on the amount of Middle Eastern crude that can be processed by each refinery and the conversion proportions of crude to gasoline and diesel fuel.

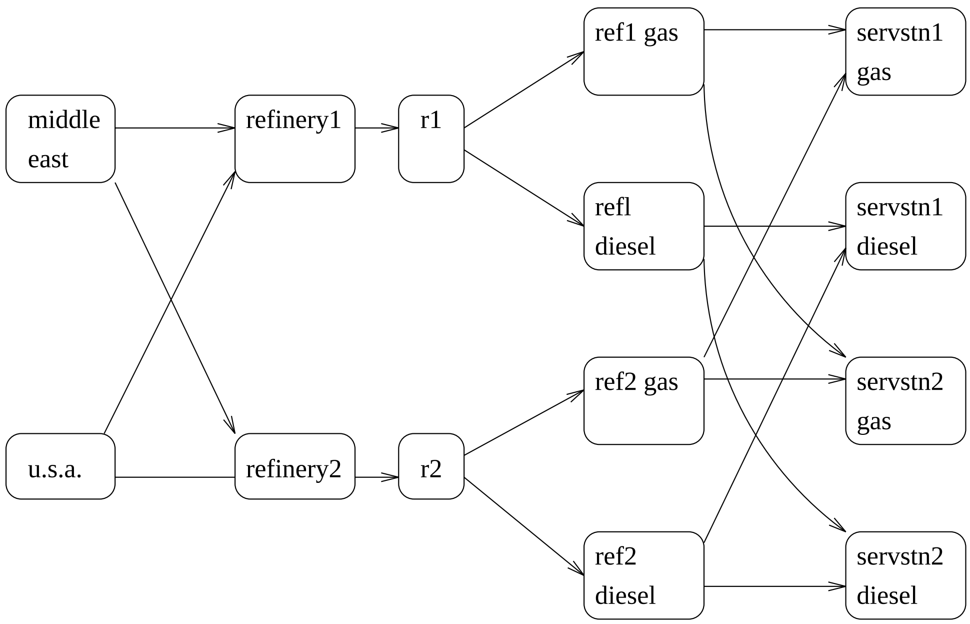


Figure 4.6. Oil Industry Example

To solve this problem with PROC NETFLOW, save a representation of the model in three SAS data sets. In the NODEDATA= data set, you name the supply and demand nodes and give the associated supplies and demands. To distinguish demand nodes from supply nodes, specify demands as negative quantities. For the oil example, the NODEDATA= data set can be saved as follows:

```
title  'Oil Industry Example';
title3 'Setting Up Nodedata = Noded For Proc Netflow';
data noded;
   input   _node_&$15. _sd_;
   datalines;
middle east       100
u.s.a.             80
servstn1 gas      -95
servstn1 diesel   -30
servstn2 gas      -40
servstn2 diesel   -15
;
```

The ARCDATA= data set contains the rest of the information about the network. Each observation in the data set identifies an arc in the network and gives the cost per flow unit across the arc, the capacities of the arc, the lower bound on flow across the arc, and the name of the arc.

```
title3 'Setting Up Arcdata = Arcd1 For Proc Netflow';
data arcd1;
   input _from_&$11. _to_&$15. _cost_ _capac_ _lo_ _name_ $;
   datalines;
middle east    refinery 1        63     95   20    m_e_ref1
middle east    refinery 2        81     80   10    m_e_ref2
u.s.a.         refinery 1        55      .    .    .
u.s.a.         refinery 2        49      .    .    .
refinery 1     r1               200    175   50    thruput1
refinery 2     r2               220    100   35    thruput2
r1             ref1 gas           .    140    .    r1_gas
r1             ref1 diesel        .     75    .    .
r2             ref2 gas           .    100    .    r2_gas
r2             ref2 diesel        .     75    .    .
ref1 gas       servstn1 gas      15     70    .    .
ref1 gas       servstn2 gas      22     60    .    .
ref1 diesel    servstn1 diesel   18      .    .    .
ref1 diesel    servstn2 diesel   17      .    .    .
ref2 gas       servstn1 gas      17     35    5    .
ref2 gas       servstn2 gas      31      .    .    .
ref2 diesel    servstn1 diesel   36      .    .    .
ref2 diesel    servstn2 diesel   23      .    .    .
;
```

Finally, the CONDATA= data set contains the side constraints for the model.

```
title3 'Setting Up Condata = Cond1 For Proc Netflow';
data cond1;
   input m_e_ref1 m_e_ref2 thruput1 r1_gas thruput2 r2_gas
         _type_ $ _rhs_;
   datalines;
-2  .  1 .  . . >= -15
 . -2  . .  1 . GE -15
 .  . -3 4  . . EQ   0
 .  .  . . -3 4  =   0
;
```

Note that the SAS variable names in the CONDATA= data set are the names of arcs given in the ARCDATA= data set. These are the arcs that have nonzero constraint coefficients in side constraints. For example, the proportionality constraint that specifies that one unit of crude at each refinery yields three-fourths of a unit of gasoline and one-fourth of a unit of diesel fuel is given for REFINERY 1 in the third observation and for REFINERY 2 in the last observation. The third observation requires that each unit of flow on arc THRUPUT1 equals three-fourths of a unit of flow on arc R1_GAS. Because all crude processed at REFINERY 1 flows through THRUPUT1 and all gasoline produced at REFINERY 1 flows through R1_GAS, the constraint models the situation. It proceeds similarly for REFINERY 2 in the last observation.

To find the minimum cost flow through the network that satisfies the supplies, demands, and side constraints, invoke PROC NETFLOW as follows:

```
proc netflow
   nodedata=noded        /* the supply and demand data */
   arcdata=arcd1         /* the arc descriptions       */
   condata=cond1         /* the side constraints       */
   conout=solution;      /* the solution data set      */
run;
```

The following messages, that appear on the SAS log, summarize the model as read by PROC NETFLOW and note the progress toward a solution:

```
NOTE: Number of nodes= 14 .
NOTE: Number of supply nodes= 2 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 180 , total demand= 180 .
NOTE: Number of arcs= 18 .
NOTE: Number of iterations performed (neglecting any
      constraints)= 8 .
NOTE: Of these, 0 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= 50600 .
NOTE: Number of <= side constraints= 0 .
NOTE: Number of == side constraints= 2 .
NOTE: Number of >= side constraints= 2 .
NOTE: Number of arc and nonarc variable side constraint
      coefficients= 8 .
NOTE: Number of iterations, optimizing with constraints= 4 .
NOTE: Of these, 0 were degenerate.
NOTE: Optimum reached.
NOTE: Minimal total cost= 50875 .
NOTE: The data set WORK.SOLUTION has 18 observations and 14
      variables.
```

Unlike PROC LP, which displays the solution and other information as output, PROC NETFLOW saves the optimum in output SAS data sets that you specify. For this example, the solution is saved in the SOLUTION data set. It can be displayed with the PRINT procedure as

```
proc print data=solution;
   var _from_ _to_ _cost_ _capac_ _lo_ _name_
       _supply_ _demand_ _flow_ _fcost_ _rcost_;
   sum _fcost_;
   title3 'Constrained Optimum'; run;
```

Constrained Optimum

Obs	_from_	_to_	_cost_	_capac_	_lo_	_name_
1	refinery 1	r1	200	175	50	thruput1
2	refinery 2	r2	220	100	35	thruput2
3	r1	ref1 diesel	0	75	0	
4	r1	ref1 gas	0	140	0	r1_gas
5	r2	ref2 diesel	0	75	0	
6	r2	ref2 gas	0	100	0	r2_gas
7	middle east	refinery 1	63	95	20	m_e_ref1
8	u.s.a.	refinery 1	55	99999999	0	
9	middle east	refinery 2	81	80	10	m_e_ref2
10	u.s.a.	refinery 2	49	99999999	0	
11	ref1 diesel	servstn1 diesel	18	99999999	0	
12	ref2 diesel	servstn1 diesel	36	99999999	0	
13	ref1 gas	servstn1 gas	15	70	0	
14	ref2 gas	servstn1 gas	17	35	5	
15	ref1 diesel	servstn2 diesel	17	99999999	0	
16	ref2 diesel	servstn2 diesel	23	99999999	0	
17	ref1 gas	servstn2 gas	22	60	0	
18	ref2 gas	servstn2 gas	31	99999999	0	

Obs	_SUPPLY_	_DEMAND_	_FLOW_	_FCOST_	_RCOST_
1	.	.	145.00	29000.00	.
2	.	.	35.00	7700.00	29
3	.	.	36.25	0.00	.
4	.	.	108.75	0.00	.
5	.	.	8.75	0.00	.
6	.	.	26.25	0.00	.
7	100	.	80.00	5040.00	.
8	80	.	65.00	3575.00	.
9	100	.	20.00	1620.00	.
10	80	.	15.00	735.00	.
11	.	30	30.00	540.00	.
12	.	30	0.00	0.00	12
13	.	95	68.75	1031.25	.
14	.	95	26.25	446.25	.
15	.	15	6.25	106.25	.
16	.	15	8.75	201.25	.
17	.	40	40.00	880.00	.
18	.	40	0.00	0.00	7
				========	
				50875.00	

Figure 4.7. CONOUT=SOLUTION

Notice that, in CONOUT=SOLUTION (Figure 4.7), the optimal flow through each arc in the network is given in the variable named _FLOW_, and the cost of flow through each arc is given in the variable _FCOST_.

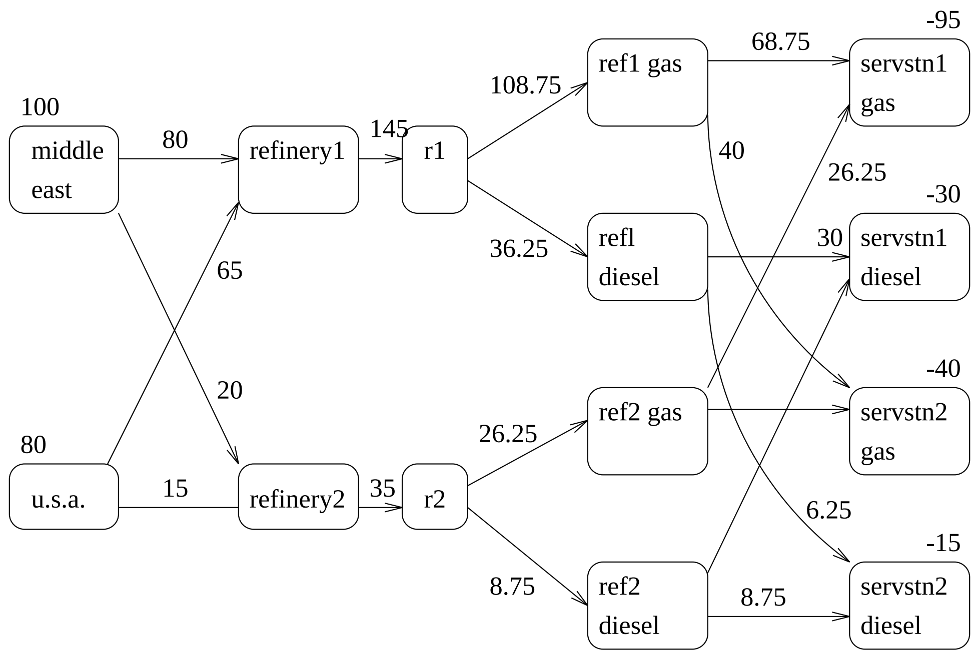


Figure 4.8. Oil Industry solution

Syntax

The following statements are used by PROC NETFLOW.

PROC NETFLOW *options* **;**
CAPACITY *variable* **;**
COEF *variables* **;**
COLUMN *variable* **;**
CONOPT;
COST *variable* **;**
DEMAND *variable* **;**
HEADNODE *variable* **;**
ID *variables* **;**
LO *variable* **;**
NAME *variable* **;**
NODE *variable* **;**
PIVOT;
PRINT *options* **;**
QUIT;
RESET *options* **;**
ROW *variables* **;**
RHS *variables* **;**
RUN;
SAVE *options* **;**
SHOW *options* **;**
SUPDEM *variable* **;**
SUPPLY *variable* **;**
TAILNODE *variable* **;**
TYPE *variable* **;**
VAR *variables* **;**

PROC NETFLOW *options*;	required statement
TAILNODE *variable*; **HEADNODE** *variable*; **COST** *variable*; **CAPACITY** *variable*; **LO** *variable*; **SUPPLY** *variable*; **DEMAND** *variable*; **NAME** *variable*; **ID** *variables*;	optional ARCDATA lists
NODE *variable*; **SUPDEM** *variable*;	optional NODEDATA lists
COLUMN *variable*; **ROW** *variables*; **COEF** *variables*; **TYPE** *variable*; **VAR** *variables*; **RHS** *variable*;	optional CONDATA lists
RESET *options*; **SAVE** *options*; **SHOW** *options*; **PRINT** *options*; **CONOPT**; **RUN**; **PIVOT**; **QUIT**;	optional interactive statements

Interactivity

PROC NETFLOW can be used interactively. You begin by giving the PROC NETFLOW statement, and you must specify the ARCDATA= data set. The CONDATA= data set must also be specified if the problem has side constraints. If necessary, specify the NODEDATA= data set.

The variable lists should be given next. If you have variables in the input data sets that have special names (for example, a variable in the ARCDATA= data set named _TAIL_ that has tail nodes of arcs as values), it may not be necessary to have many or any variable lists.

The CONOPT, PIVOT, PRINT, QUIT, SAVE, SHOW, RESET, and RUN statements follow and can be listed in any order. The CONOPT and QUIT statements can be used only once. The others can be used as many times as needed.

Use the RESET or SAVE statement to change the names of the output data sets. With RESET, you can also indicate the reasons why optimization should stop, (for example, you can indicate the maximum number of stage 1 or stage 2 iterations that can be performed). PROC NETFLOW then has a chance to either execute the next statement, or, if the next statement is one that PROC NETFLOW does not recognize (the next PROC or DATA step in the SAS session), do any allowed optimization and finish. If no new statement has been submitted, you are prompted for one. Some options of the RESET statement enable you to control aspects of the Primal Simplex algorithm. Specifying certain values for these options can reduce the time it takes to solve a problem. Note that any of the RESET options can be specified in the PROC NETFLOW statement.

The RUN statement starts or resumes optimization. The PIVOT statement makes PROC NETFLOW perform one Simplex iteration. The QUIT statement immediately stops PROC NETFLOW. The CONOPT statement forces PROC NETFLOW to consider constraints when it next performs optimization. The SAVE statement has options that enable you to name output data sets; information about the current solution is put in these output data sets. Use the SHOW statement if you want to examine the values of options of other statements. Information about the amount of optimization that has been done and the STATUS of the current solution can also be displayed using the SHOW statement.

The PRINT statement instructs PROC NETFLOW to display parts of the problem. PRINT ARCS produces information on all arcs. PRINT SOME_ARCS limits this output to a subset of arcs. There are similar PRINT statements for nonarc variables and constraints:

```
print nonarcs;
print some_nonarcs;
print constraints;
print some_cons;
```

PRINT CON_ARCS enables you to limit constraint information that is obtained to members of a set of arcs that have nonzero constraint coefficients in a set of constraints. PRINT CON_NONARCS is the corresponding statement for nonarc variables.

For example, an interactive PROC NETFLOW run might look something like this:

```
proc netflow
       arcdata=data set
       other options;
   variable list specifications; /* if necessary */
   reset options;
   print options;    /* look at problem                  */
run;                 /* do some optimization             */
   /* suppose that optimization stopped for  */
   /* some reason or you manually stopped it */
   print options;    /* look at the current solution     */
   save options;     /* keep current solution            */
   show options;     /* look at settings                 */
   reset options;    /* change some settings, those that */
                     /* caused optimization to stop      */
   run;              /* do more optimization             */
   print options;    /* look at the optimal solution     */
   save options;     /* keep optimal solution            */
```

If you are interested only in finding the optimal solution, have used SAS variables that have special names in the input data sets, and want to use default setting for everything, then the following statement is all you need:

PROC NETFLOW ARCDATA=*data set options* **;**

Functional Summary

The following tables outline the options available for the NETFLOW procedure classified by function.

Table 4.1. Input Data Set Options

Description	Statement	Option
arcs input data set	NETFLOW	ARCDATA=
nodes input data set	NETFLOW	NODEDATA=
constraint input data set	NETFLOW	CONDATA=

Table 4.2. Options for Networks

Description	Statement	Option
default arc cost	NETFLOW	DEFCOST=
default arc capacity	NETFLOW	DEFCAPACITY=
default arc lower flow bound	NETFLOW	DEFMINFLOW=
network's only supply node	NETFLOW	SOURCE=
SOURCE's supply capability	NETFLOW	SUPPLY=
network's only demand node	NETFLOW	SINK=
SINK's demand	NETFLOW	DEMAND=
excess supply or demand is conveyed through network	NETFLOW	THRUNET
find maximal flow between SOURCE and SINK	NETFLOW	MAXFLOW
cost of bypass arc when solving MAXFLOW problem	NETFLOW	BYPASSDIV=
find shortest path from SOURCE to SINK	NETFLOW	SHORTPATH

Table 4.3. Miscellaneous Options

Description	Statement	Option
infinity value	NETFLOW	INFINITY=
do constraint row and/or nonarc variable column coefficient scaling, or neither	NETFLOW	SCALE=
maximization instead of minimization	NETFLOW	MAXIMIZE
use warm start solution	NETFLOW	WARM
all-artificial starting solution	NETFLOW	ALLART

Table 4.4. Data Set Read Options

Description	Statement	Option
CONDATA has sparse data format	NETFLOW	SPARSECONDATA
default constraint type	NETFLOW	DEFCONTYPE=
special COLUMN variable value	NETFLOW	TYPEOBS=
special COLUMN variable value	NETFLOW	RHSOBS=
is used to interpret arc and nonarc variable names in the CONDATA	NETFLOW	NAMECTRL=
no new nonarc variables	NETFLOW	SAME_NONARC_DATA
no nonarc data in the ARCDATA	NETFLOW	ARCS_ONLY_ARCDATA
data for an arc found in only one obs of ARCDATA	NETFLOW	ARC_SINGLE_OBS
data for an constraint found in only one obs of CONDATA	NETFLOW	CON_SINGLE_OBS
data for a coefficient found once in CONDATA	NETFLOW	NON_REPLIC=
data is grouped, exploited during data read	NETFLOW	GROUPED=

Table 4.5. Problem Size (approx.) Options

Description	Statement	Option
number of nodes	NETFLOW	NNODES=
number of arcs	NETFLOW	NARCS=
number of nonarc variables	NETFLOW	NNAS=
number of coefficients	NETFLOW	NCOEFS=
number of constraints	NETFLOW	NCONS=

Table 4.6. Memory Control Options

Description	Statement	Option
issue memory usage messages to SASLOG	NETFLOW	MEMREP
number of bytes to use for main memory	NETFLOW	BYTES=
proportion of memory used by frequently accessed arrays	NETFLOW	COREFACTOR=
memory allocated for LU factors	NETFLOW	DWIA=
linked list for updated column	NETFLOW	SPARSEP2
use 2-dimensional array instead of LU factors for basis matrix	NETFLOW	INVD_2D
maximum bytes for a single array	NETFLOW	MAXARRAYBYTES=

Table 4.7. Output Data Set Options: RESET

Description	Statement	Option
unconstrained solution data set	RESET	ARCOUT=
unconstrained solution data set	RESET	NODEOUT=
constrained solution data set	RESET	CONOUT=
constrained solution data set	RESET	DUALOUT=

Table 4.8. Stop Optimization Options

Description	Statement	Option
pause after stage 1. Don't start stage 2	RESET	ENDPAUSE1
pause when feasible - stage 1	RESET	FEASIBLEPAUSE1
pause when feasible - stage 2	RESET	FEASIBLEPAUSE2
maximum number of iterations - stage 1	RESET	MAXIT1=
maximum number of iterations - stage 2	RESET	MAXIT2=
negates ENDPAUSE1	RESET	NOENDPAUSE1
negates FEASIBLEPAUSE1	RESET	NOFEASIBLEPAUSE1
negates FEASIBLEPAUSE2	RESET	NOFEASIBLEPAUSE2
pause every PAUSE1 iterations - stage 1	RESET	PAUSE1=
pause every PAUSE2 iterations - stage 2	RESET	PAUSE2=

Table 4.9. Simplex Options

Description	Statement	Option
Use Big M instead of twophase - stage 1	RESET	BIGM1
Use Big M instead of twophase - stage 2	RESET	BIGM2
anti-cycling option	RESET	CYCLEMULT1
interchange first eligible nonkey with leaving key variable	RESET	INTFIRST
invert working basis matrix (2-dim) every INVFREQ= iterations	RESET	INVFREQ=
maximum number of L row operations allowed before refactorization done instead of factor column update	RESET	MAXL=
maximum number fo LU factor column updates, otherwise refactorize	RESET	MAXLUUPDATES=
anticycling option	RESET	MINBLOCK1
first eligible leaving variable used, otherwise best is used	RESET	LRATIO1
first eligible leaving variable used, otherwise best is used	RESET	LRATIO2
negates INTFIRST	RESET	NOINTFIRST
negates LRATIO1	RESET	NOLRATIO1
negates LRATIO2	RESET	NOLRATIO2
negates PERTURB1	RESET	NOPERTURB1
anti-cycling option	RESET	PERTURB1
re-factorize working basis matrix every REFACTFREQ= iterations	RESET	REFACTFREQ=
use twophase instead of Big M - stage 1	RESET	TWOPHASE1
use twophase instead of Big M - stage 2	RESET	TWOPHASE2
pivot element selection parameter	RESET	U=
zero tolerance - stage 1	RESET	ZERO1=
zero tolerance - stage 2	RESET	ZERO2=
zero tolerance - real number comparisons	RESET	ZEROTOL=

Table 4.10. Pricing Options

Description	Statement	Option
calculate dual values every DUALFREQ= iterations after major iterations	RESET	DUALFREQ=
pricing strategy - stage 1	RESET	PRICETYPE1=
pricing strategy - stage 2	RESET	PRICETYPE2=
used when P1SCAN=PARTIAL	RESET	P1NPARTIAL=
how nonbasis variables (PRICETYPE1=NOQ) or queue candidates (PRICETYPE1=Q) are scanned	RESET	P1SCAN=
used when P2SCAN=PARTIAL	RESET	P2NPARTIAL=
how nonbasis variables (PRICETYPE2=NOQ) or queue candidates (PRICETYPE2=Q) are scanned	RESET	P2SCAN=
initial queue size - stage 1	RESET	QSIZE1=
initial queue size - stage 2	RESET	QSIZE2=
used when Q1FILLSCAN=PARTIAL	RESET	Q1FILLNPARTIAL=
how candidates are scanned when filling queue - stage 1	RESET	Q1FILLSCAN=
used when Q2FILLSCAN=PARTIAL	RESET	Q2FILLNPARTIAL=
how candidates are scanned when filling queue - stage 2	RESET	Q2FILLSCAN=
queue size reduction factor	RESET	REDUCEQSIZE1=
queue size reduction factor	RESET	REDUCEQSIZE2=
when the queue are refreshed - stage 1	RESET	REFRESHQ1=
when the queue are refreshed - stage 2	RESET	REFRESHQ2=

Table 4.11. Miscellaneous Options: RESET

Description	Statement	Option
output complete basis information to ARCOUT= and NODEOUT= data sets	RESET	FUTURE1
output complete basis information to CONOUT= and DUALOUT= data sets	RESET	FUTURE2
turn off infeasibility or optimality flags as more optimization is to be done	RESET	MOREOPT
negates FUTURE1	RESET	NOFUTURE1
negates FUTURE2	RESET	NOFUTURE2
negates SCRATCH	RESET	NOSCRATCH
negates ZTOL1	RESET	NOZTOL1
negates ZTOL2	RESET	NOZTOL2
do not do stage 1 optimization. Do stage 2 optimization	RESET	SCRATCH
display this number of similar SAS log messages, suppress the rest	RESET	VERBOSE=
use zero tolerance- stage 1	RESET	ZTOL1
use zero tolerance- stage 2	RESET	ZTOL2

Table 4.12. PRINT Statement Options

Description	Statement	Option
display everything	PRINT	PROBLEM
display arc information	PRINT	ARCS
display nonarc variable information	PRINT	NONARCS
display variables information	PRINT	VARIABLES
display constraint information	PRINT	CONSTRAINTS
display information for some arcs	PRINT	SOME_ARCS
display information for some nonarc variables	PRINT	SOME_NONARCS
display information for some variables	PRINT	SOME_VARIABLES
display information for some constraints	PRINT	SOME_CONS
display information for some constraints associated with some arcs	PRINT	CON_ARCS
display information for some constraints associated with some nonarc variables	PRINT	CON_NONARCS
display information for some constraints associated with some variables	PRINT	CON_VARIABLES

Table 4.13. PRINT Statement Qualifiers

Description	Statement	Option
produce a short report	PRINT	/ SHORT
produce a long report	PRINT	/ LONG
only arcs (nonarc variables) with zero flow (value)	PRINT	/ ZERO
only arcs (nonarc variables) with nonzero flow (value)	PRINT	/ NONZERO
only for basics	PRINT	/ BASIC
only nonbasics	PRINT	/ NONBASIC

Table 4.14. SHOW Statement Options

Description	Statement	Option
show problem, optimization status	SHOW	STATUS
show network model parameters	SHOW	NETSTMT
show data sets that have, will be created	SHOW	DATA SETS
show options that pause optimization	SHOW	PAUSE
show Simplex algorithm options	SHOW	SIMPLEX
show pricing strategy options	SHOW	PRICING
show miscellaneous options	SHOW	MISC

Table 4.15. SHOW Statement Qualifiers

Description	Statement	Option
only if relevant	SHOW	RELEVANT
only stage 1 options when doing stage 1. only stage 2 options when doing stage 2	SHOW	STAGE

Table 4.16. Output Data Set Options: SAVE

Description	Statement	Option
unconstrained solution data set	SAVE	ARCOUT=
unconstrained solution data set	SAVE	NODEOUT=
constrained solution data set	SAVE	CONOUT=
constrained solution data set	SAVE	DUALOUT=

Table 4.17. Interior Point algorithm Options

Description	Statement	Option
use Interior Point algorithm	NETFLOW	INTPOINT
allowed amount of dual infeasibility	RESET	TOLDINF=
allowed amount of primal infeasibility	RESET	TOLPINF=
cut-off tolerance for Cholesky factorization	RESET	CHOLTINYTOL=
density threshold for Cholesky processing	RESET	DENSETHR=
maximum number of Interior Point algorithm iterations	RESET	MAXITERB=
Primal-Dual (Duality) gap tolerance	RESET	PDGAPTOL=
step-length multiplier	RESET	PDSTEPMULT=
preprocessing type	RESET	PRSLTYPE=

PROC NETFLOW Statement

PROC NETFLOW *options* **;**

This statement invokes the procedure. The following options and the options listed with the RESET statement can appear in the PROC NETFLOW statement.

Overview of PROC NETFLOW Options

The options available with the PROC NETFLOW statement are summarized by purpose in Table 4.18.

Table 4.18. Functional Summary, PROC NETFLOW statement

Description	Statement	Option
Input Data Set Options		
arcs input data set	NETFLOW	ARCDATA=
nodes input data set	NETFLOW	NODEDATA=
constraint input data set	NETFLOW	CONDATA=
Output Data Set Options		
unconstrained primal solution data set	NETFLOW	ARCOUT=
unconstrained dual solution data set	NETFLOW	NODEOUT=
constrained primal solution data set	NETFLOW	CONOUT=
constrained dual solution data set	NETFLOW	DUALOUT=

Description	Statement	Option
Options for Networks		
default arc cost	NETFLOW	DEFCOST=
default arc capacity	NETFLOW	DEFCAPACITY=
default arc lower flow bound	NETFLOW	DEFMINFLOW=
network's only supply node	NETFLOW	SOURCE=
SOURCE's supply capability	NETFLOW	SUPPLY=
network's only demand node	NETFLOW	SINK=
SINK's demand	NETFLOW	DEMAND=
excess supply or demand is conveyed through network	NETFLOW	THRUNET
find maximal flow between SOURCE and SINK	NETFLOW	MAXFLOW
cost of bypass arc when solving MAXFLOW problem	NETFLOW	BYPASSDIV=
find shortest path from SOURCE to SINK	NETFLOW	SHORTPATH
Miscellaneous Options		
infinity value	NETFLOW	INFINITY=
do constraint row and/or nonarc variable column coefficient scaling, or neither	NETFLOW	SCALE=
maximization instead of minimization	NETFLOW	MAXIMIZE
use warm start solution	NETFLOW	WARM
all-artificial starting solution	NETFLOW	ALLART
Data Set Read Options		
CONDATA has sparse data format	NETFLOW	SPARSECONDATA
default constraint type	NETFLOW	DEFCONTYPE=
special COLUMN variable value	NETFLOW	TYPEOBS=
special COLUMN variable value	NETFLOW	RHSOBS=
is used to interpret arc and nonarc variable names in the CONDATA	NETFLOW	NAMECTRL=
no new nonarc variables	NETFLOW	SAME_NONARC_DATA
no nonarc data in the ARCDATA	NETFLOW	ARCS_ONLY_ARCDATA
data for an arc found in only one obs of ARCDATA	NETFLOW	ARC_SINGLE_OBS
data for an constraint found in only one obs of CONDATA	NETFLOW	CON_SINGLE_OBS
data for a coefficient found once in CONDATA	NETFLOW	NON_REPLIC=
data is grouped, exploited during data read	NETFLOW	GROUPED=
Problem Size (approx.) Options		
number of nodes	NETFLOW	NNODES=
number of arcs	NETFLOW	NARCS=
number of nonarc variables	NETFLOW	NNAS=
number of coefficients	NETFLOW	NCOEFS=

Description	Statement	Option
number of constraints	NETFLOW	NCONS=
Memory Control Options		
issue memory usage messages to SASLOG	NETFLOW	MEMREP
number of bytes to use for main memory	NETFLOW	BYTES=
proportion of memory used by frequently accessed arrays	NETFLOW	COREFACTOR=
memory allocated for LU factors	NETFLOW	DWIA=
linked list for updated column	NETFLOW	SPARSEP2
use 2-dimensional array instead of LU factor for basis matrix	NETFLOW	INVD_2D
maximum bytes for a single array	NETFLOW	MAXARRAYBYTES=
Interior Point algorithm Options		
use Interior Point algorithm	NETFLOW	INTPOINT

The following options can be specified only in the PROC NETFLOW statement and are relevant to the start of the procedure. Once specified, they cannot be changed.

Data Set Options

This section briefly describes all the input and output data sets used by PROC NETFLOW. The ARCDATA= data set, NODEDATA= data set, and CONDATA= data set can contain SAS variables that have special names, for instance _CAPAC_, _COST_, and _HEAD_. PROC NETFLOW looks for such variables if you do not give explicit variable list specifications. If a SAS variable with a special name is found and that SAS variable is not in another variable list specification, PROC NETFLOW determines that values of the SAS variable are to be interpreted in a special way. By using SAS variables that have special names, you may not need to have any variable list specifications.

ARCDATA=*SAS-data-set*

names the data set that contains arc and, optionally, nonarc variable information and nodal supply/demand data. The ARCDATA= data set must be specified in all PROC NETFLOW statements.

ARCOUT=*SAS-data-set*
AOUT=*SAS-data-set*

names the output data set that receives all arc and nonarc variable data, including flows or values, and other information concerning the unconstrained optimal solution. The supply and demand information can also be found in the ARCOUT= data set. Once optimization that considers side constraints starts, you are not able to obtain an ARCOUT= data set. Instead, use the CONOUT= data set to get the current solution. See the "ARCOUT= and CONOUT= Data Sets" section on page 276 for more information.

CONDATA=*SAS-data-set*
names the data set that contains the side constraint data. The data set can also contain other data such as arc costs, capacities, lower flow bounds, nonarc variable upper and lower bounds, and objective function coefficients. PROC NETFLOW needs a CONDATA= data set to solve a constrained problem or a Linear Programming problem. See the “CONDATA= Data Set” section on page 267 for more information.

CONOUT=*SAS-data-set*
COUT=*SAS-data-set*
names the output data set that receives an optimal primal solution to the problem obtained by performing optimization that considers the side constraints. See the “ARCOUT= and CONOUT= Data Sets” section on page 276 for more information.

DUALOUT=*SAS-data-set*
DOUT=*SAS-data-set*
names the output data set that receives an optimal dual solution to the problem obtained by performing optimization that considers the side constraints. See the “NODEOUT= and DUALOUT= Data Sets” section on page 277 for more information.

NODEDATA=*SAS-data-set*
DUALIN=*SAS-data-set*
names the data set that contains the node supply and demand specifications. You do not need observations in the NODEDATA= data set for transshipment nodes. (Transshipment nodes neither supply nor demand flow.) All nodes are assumed to be transshipment nodes until supply or demand data indicate otherwise. It is acceptable for some arcs to be directed toward supply nodes or away from demand nodes.

The use of the NODEDATA= data set is optional in the PROC NETFLOW statement provided that, if the NODEDATA= data set is not used, supply and demand details are specified by other means. Other means include using the MAXFLOW or SHORTPATH option, SUPPLY or DEMAND list variables (or both) in the ARCDATA= data set, and the SOURCE=, SUPPLY=, SINK=, or DEMAND= option in the PROC NETFLOW statement.

NODEOUT=*SAS-data-set*
names the output data set that receives all information about nodes (supply and demand and nodal dual variable values) and other information concerning the optimal solution found by the optimizer when neglecting side constraints. Once optimization that considers side constraints starts, you are not able to obtain a NODEOUT= data set. Instead, use the DUALOUT= data set to get the current solution dual information. See the “NODEOUT= and DUALOUT= Data Sets” section on page 277 for a more complete description.

General Options

The following is a list of options you can use with PROC NETFLOW. The options are listed in alphabetical order.

ALLART
indicates that PROC NETFLOW uses an all artificial initial solution (Kennington and Helgason 1980, p. 68) instead of the default *good path* method for determining

an initial solution (Kennington and Helgason 1980, p. 245). The ALLART initial solution is generally not as good; more iterations are usually required before the optimal solution is obtained. However, because less time is used when setting up an ALLART start, it can offset the added expenditure of CPU time in later computations.

ARCS_ONLY_ARCDATA

indicates that data for only arcs are in the ARCDATA= data set. When PROC NETFLOW reads the data in ARCDATA= data set, memory would not be wasted to receive data for nonarc variables. The read might then be performed faster. See the "How to Make the Data Read of PROC NETFLOW More Efficient" section on page 296.

ARC_SINGLE_OBS

indicates that for all arcs and nonarc variables, data for each arc or nonarc variable is found in only one observation of the ARCDATA= data set. When reading the data in ARCDATA= data set, PROC NETFLOW knows that the data in an observation is for an arc or a nonarc variable that has not had data previously read that needs to be checked for consistency. The read might then be performed faster.

If you specify ARC_SINGLE_OBS, PROC NETFLOW automatically works as if netflowgrouped=ARCDATA is also specified.

See the "How to Make the Data Read of PROC NETFLOW More Efficient" section on page 296.

BYPASSDIVIDE=*b*
BYPASSDIV=*b*
BPD=*b*

should be used only when the MAXFLOW option has been specified; that is, PROC NETFLOW is solving a maximal flow problem. PROC NETFLOW prepares to solve maximal flow problems by setting up a bypass arc. This arc is directed from the SOURCE to the SINK and will eventually convey flow equal to INFINITY minus the maximal flow through the network. The cost of the bypass arc must be expensive enough 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 can be nonzero if some network arcs have nonzero costs. The cost of the bypass is set to the value of the INFINITY= option. Valid values for the BYPASSDIV= option 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 the BYPASSDIV= option. The reduced costs in the ARCOUT= data set and the 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 if you specify the BYPASSDIV= option, 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 the BYPASSDIV= option (in the presence of nonzero arc costs) is 100.0.

BYTES=*b*

indicates the size of the main working memory (in bytes) that PROC NETFLOW will allocate. The default value for the BYTES= option is near to the number of bytes of the largest contiguous memory that can be allocated for this purpose. The working memory is used to store all the arrays and buffers used by PROC NETFLOW. If this memory has a size smaller than what is required to store all arrays and buffers, PROC NETFLOW uses various schemes that page information between memory and disk.

PROC NETFLOW uses more memory than the main working memory. The additional memory requirements cannot be determined at the time when the main working memory is allocated. For example, every time an output data set is created, some additional memory is required. Do not specify a value for the BYTES= option equal to the size of available memory.

CON_SINGLE_OBS

improves how the CONDATA= data set is read. How it works depends on whether the CONDATA has a dense or sparse format.

If CONDATA has the dense format, specifying CON_SINGLE_OBS indicates that, for each constraint, data for each can be found in only one observation of CONDATA.

If CONDATA has a sparse format, and data for each arc and nonarc variablecan be found in only one observation of CONDATA, then specify the CON_SINGLE_OBS option. If there are n SAS variables in the ROW and COEF list, then each arc or nonarc can have at most n constraint coefficients in the model. See the “How to Make the Data Read of PROC NETFLOW More Efficient” section on page 296.

COREFACTOR=*c*
CF=*c*

enables you to specify the maximum proportion of memory to be used by the arrays frequently accessed by PROC NETFLOW. PROC NETFLOW strives to maintain all information required during optimization in core. If the amount of available memory is not great enough to store the arrays completely in core, either initially or as memory requirements grow, PROC NETFLOW can change the memory management scheme it uses. Large problems can still be solved. When necessary, PROC NETFLOW transfers data from random access memory (RAM) or core that can be accessed quickly but is of limited size to slower access large capacity disk memory. This is called *paging*.

Some of the arrays and buffers used during constrained optimization either vary in size, are not required as frequently as other arrays, or are not required throughout the Simplex iteration. Let a be the amount of memory in bytes required to store frequently accessed arrays of nonvarying size. Specify the MEMREP option in the PROC NETFLOW statement to get the value for a and a report of memory usage. If the size of the main working memory BYTES=b multiplied by COREFACTOR=c is greater than a, PROC NETFLOW keeps the frequently accessed arrays of nonvarying size resident in core throughout the optimization. If the other arrays cannot fit into core, they are paged in and out of the remaining part of the main working memory.

If b multiplied by c is less than a, PROC NETFLOW uses a different memory scheme. The working memory is used to store only the arrays needed in the part of the algorithm being executed. If necessary, these arrays are read from disk into the main working area. Paging, if required, is done for all these arrays, and sometimes information is written back to disk at the end of that part of the algorithm. This memory scheme is not as fast as the other memory schemes. However, problems can be solved with memory that is too small to store every array.

PROC NETFLOW is capable of solving very large problems in a modest amount of available memory. However, as more time is spent doing input/output operations, the speed of PROC NETFLOW decreases. It is important to choose the value of the COREFACTOR= option carefully. If COREFACTOR is too small, the memory scheme that needs to be used might not be as efficient as another that could have been used had a larger COREFACTOR been specified. If COREFACTOR is too large, too much of the main working memory is occupied by the frequently accessed, nonvarying sized arrays, leaving too little for the other arrays. The amount of input/output operations for these other arrays can be so high that another memory scheme might have been used more beneficially.

The valid values of COREFACTOR=c are between 0.0 and 0.95, inclusive. The default value for c is 0.75 when there are over 200 side constraints, and 0.9 when there is only one side constraint. When the problem has between 2 and 200 constraints, the value of c lies between the two points (1, 0.9) and (201, 0.75).

DEFCAPACITY=*c*
DC=*c*

requests that the default arc capacity and the default nonarc variable value upper bound be c. If this option is not specified, then DEFCAPACITY= INFINITY.

DEFCONTYPE=*c*
DEFTYPE=*c*
DCT=*c*

specifies the default constraint type. This default constraint type is either *less than or equal to* or is the type indicated by DEFCONTYPE=c. Valid values for this option are

LE, le, $<=$	for *less than or equal to*
EQ, eq, $=$	for *equal to*
GE, ge, $>=$	for *greater than or equal to*

The values do not need to be enclosed in quotes.

DEFCOST=*c*

requests that the default arc cost and the default nonarc variable objective function coefficient be c. If this option is not specified, then DEFCOST=0.0.

DEFMINFLOW=*m*
DMF=*m*

requests that the default lower flow bound through arcs and the default lower value bound of nonarc variables be m. If a value is not specified, then DEFMINFLOW=0.0.

DEMAND=*d*

specifies the demand at the SINK node specified by the SINK= option. The DEMAND= option should be used only if the SINK= option is given in the PROC NETFLOW statement and neither the SHORTPATH option nor the MAXFLOW option is specified. If you are solving a minimum cost network problem and the SINK= option is used to identify the sink node, and the DEMAND= option is not specified, then the demand at the sink node is made equal to the network's total supply.

DWIA=*i*

controls the initial amount of memory to be allocated to store the **LU** factors of the working basis matrix. DWIA stands for D^W *initial allocation* and i is the number of nonzeros and matrix row operations in the **LU** factors that can be stored in this memory. Due to fill-in in the **U** factor and the growth in the number of row operations, it is often necessary to move information about elements of a particular row or column to another location in the memory allocated for the **LU** factors. This process leaves some memory temporarily unoccupied. Therefore, DWIA=i must be greater than the memory required to store only the **LU** factors.

Occasionally, it is necessary to compress the **U** factor so that it again occupies contiguous memory. Specifying too large a value for DWIA means that more memory is required by PROC NETFLOW. This might cause more expensive memory mechanisms to be used than if a smaller but adequate value had been specified for DWIA=. Specifying too small a value for the DWIA= option can make time-consuming compressions more numerous. The default value for the DWIA= option is eight times the number of side constraints.

GROUPED=*c*

PROC NETFLOW can take a much shorter time to read data if the data have been grouped prior to the PROC NETFLOW call. This enables PROC NETFLOW to conclude that, for instance, a new NAME list variable value seen in an ARCDATA= data set grouped by the values of the NAME list variable before PROC NETFLOW was called is new. PROC NETFLOW does not need to check that the NAME has been read in a previous observation. See the "How to Make the Data Read of PROC NETFLOW More Efficient" section on page 296.

- GROUPED=ARCDATA indicates that the ARCDATA= data set has been grouped by values of the NAME list variable. If _NAME_ is the name of the NAME list variable, you could use PROC SORT DATA=ARCDATA;BY _NAME_; prior to calling PROC NETFLOW. Technically, you do not have to sort the data, only ensure that all similar values of the NAME list variable are grouped together. If you specify the ARCS_ONLY_ARCDATA option, PROC NETFLOW automatically works as if GROUPED=ARCDATA is also specified.
- GROUPED=CONDATA indicates that the CONDATA= data set has been grouped.

  If the CONDATA= data set has a dense format, GROUPED=CONDATA indicates that the CONDATA= data set has been grouped by values of the ROW list variable. If _ROW_ is the name of the ROW list variable, you could use

PROC SORT DATA=CONDATA;BY _ROW_; prior to calling PROC NETFLOW. Technically, you do not have to sort the data, only ensure that all similar values of the ROW list variable are grouped together. If you specify the CON_SINGLE_OBS option, or if there is no ROW list variable, PROC NETFLOW automatically works as if GROUPED=CONDATA has been specified.

If CONDATA has the sparse format, GROUPED=CONDATA indicates that CONDATA has been grouped by values of the COLUMN list variable. If _COL_ is the name of the COLUMN list variable, you could use PROC SORT DATA=CONDATA;BY _COL_; prior to calling PROC NETFLOW. Technically, you do not have to sort the data, only ensure that all similar values of the COLUMN list variable are grouped together.

- GROUPED=BOTH indicates that both GROUPED=ARCDATA and GROUPED=CONDATA are TRUE.
- GROUPED=NONE indicates that the data sets have not been grouped, that is, neither GROUPED=ARCDATA nor GROUPED=CONDATA is TRUE. This is the default, but it is much better if GROUPED=ARCDATA, or GROUPED=CONDATA, or GROUPED=BOTH.

A data set like

```
. . .  _XXXXX_  . . . .
         bbb
         bbb
         aaa
         ccc
         ccc
```

is a candidate for the GROUPED= option. Similar values are grouped together. When PROC NETFLOW is reading the ith observation, either the value of the _XXXXX_ variable is the same as the $(i-1)$th (that is, the previous observation's) _XXXXX_ value, or it is a new _XXXXX_ value not seen in any previous observation. This also means that if the ith _XXXXX_ value is different from the $(i-1)$th _XXXXX_ value, the value of the $(i-1)$th _XXXXX_ variable will not be seen in any observations i, $i+1$, ...

INFINITY=*i*
INF=*i*

is the largest number used by PROC NETFLOW in computations. A number too small can adversely affect the solution process. You should avoid specifying an enormous value for the INFINITY= option because numerical roundoff errors can result. If a value is not specified, then INFINITY=999999. The INFINITY= option cannot be assigned a value less than 9999.

INVD_2D

controls the way in that the inverse of the working basis matrix is stored. How this matrix is stored affects computations as well as how the working basis or its inverse is updated. The working basis matrix is defined in the "Details" section on page 267.

If INVD_2D is specified, the working basis matrix inverse is stored as a matrix. Typically, this memory scheme is best when there are few side constraints or when the working basis is dense.

If INVD_2D is not specified, lower (**L**) and upper (**U**) factors of the working basis matrix are used. **U** is an upper triangular matrix and **L** is a lower triangular matrix corresponding to a sequence of elementary matrix row operations. The sparsity-exploiting variant of the Bartels-Golub decomposition is used to update the **LU** factors. This scheme works well when the side constraint coefficient matrix is sparse or when many side constraints are nonbinding.

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 the MAXARRAYBYTES= option ensures that arrays that need lots of memory do not consume too much memory at the expense of other arrays.

There is one array that 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 are told what this memory amount is on the SAS log if you specify the MEMREP option in the PROC NETFLOW statement.

MAXFLOW
MF

specifies that PROC NETFLOW solve a maximum flow problem. In this case, the PROC NETFLOW procedure finds the maximum flow from the node specified by the SOURCE= option to the node specified by the SINK= option. PROC NETFLOW automatically assigns an INFINITY= option supply to the SOURCE= option node and the SINK= option is assigned the INFINITY= option demand. In this way, the MAXFLOW option sets up a maximum flow problem as an equivalent minimum cost problem.

You can use the MAXFLOW option when solving any flow problem (not necessarily a maximum flow problem) when the network has one supply node (with infinite supply) and one demand node (with infinite demand). The MAXFLOW option can be used in conjunction with all other options (except SHORTPATH, SUPPLY=, and DEMAND=) and capabilities of PROC NETFLOW.

MAXIMIZE
MAX

specifies that PROC NETFLOW find the maximum cost flow through the network. If both the MAXIMIZE and the SHORTPATH options are specified, the solution obtained is the longest path between the SOURCE= and SINK= nodes. Similarly, MAXIMIZE and MAXFLOW together cause PROC NETFLOW to find the minimum flow between these two nodes; this is zero if there are no nonzero lower flow bounds.

MEMREP

indicates that information on the memory usage and paging schemes (if necessary) is reported by PROC NETFLOW on the SAS log. As optimization proceeds, you are informed of any changes in the memory requirements and schemes used by PROC NETFLOW.

NAMECTRL=*i*

is used to interpret arc and nonarc variable names in the CONDATA= data set.

In the ARCDATA= data set, an arc is identified by its tail and head node. In the CONDATA= data set, arcs are identified by names. You can give a name to an arc by having a NAME list specification that indicates a SAS variable in the ARCDATA= data set that has names of arcs as values.

PROC NETFLOW requires arcs that have information about them in the CONDATA= data set to have names, but arcs that do not have information about them in the CONDATA= data set can also have names. Unlike a nonarc variable whose name uniquely identifies it, an arc can have several different names. An arc has a default name in the form *tail_head*, that is, the name of the arc's tail node followed by an underscore and the name of the arc's head node.

In the CONDATA= data set, if the dense data format is used, (described in the "CONDATA= Data Set" section on page 267) a name of an arc or a nonarc variable is the *name* of a SAS variable listed in the VAR list specification. If the sparse data format of the CONDATA= data set is used, a name of an arc or a nonarc variable is a *value* of the SAS variable listed in the COLUMN list specification.

The NAMECTRL= option is used when a name of an arc or nonarc variable in the CONDATA= data set (either a VAR list SAS variable name or value of the COLUMN list SAS variable) is in the form *tail_head* and there exists an arc with these end nodes. If *tail_head* has not already been tagged as belonging to an arc or nonarc variable in the ARCDATA= data set, PROC NETFLOW needs to know whether *tail_head* is the name of the arc or the name of a nonarc variable.

If you specify NAMECTRL=1, a name that is not defined in the ARCDATA= data set is assumed to be the name of a nonarc variable. NAMECTRL=2 treats *tail_head* as the name of the arc with these endnodes, provided no other name is used to associate data in the CONDATA= data set with this arc. If the arc does have other names that appear in the CONDATA= data set, *tail_head* is assumed to be the name of a nonarc variable. If you specify NAMECTRL=3, *tail_head* is assumed to be a name of the arc with these end nodes, whether the arc has other names or not. The default value of NAMECTRL is 3. Note that if you use the dense side constraint input format, the default arc name *tail_head* is not recognized (regardless of the NAMECTRL value) unless the head node and tail node names contain no lowercase letters.

If the dense format is used for the CONDATA= data set, the SAS System converts SAS variable names in a SAS program to uppercase. The VAR list variable names are uppercased. Because of this, PROC NETFLOW automatically uppercases names of arcs and nonarc variables (the values of the NAME list variable) in the ARCDATA= data set. The names of arcs and nonarc variables (the values of the NAME list variable) appear uppercased in the ARCOUT= data set and the CONOUT= data set, and in the PRINT statement output.

Also, if the dense format is used for the CONDATA= data set, be careful with default arc names (names in the form tailnode_headnode). Node names (values in the TAILNODE and HEADNODE list variables) in the ARCDATA= data set are not uppercased by PROC NETFLOW. Consider the following code:

```
data arcdata;
   input _from_ $ _to_ $ _name $ ;
   datalines;
from to1 .
from to2 arc2
TAIL TO3 .
;
data densecon;
   input from_to1 from_to2 arc2 tail_to3;
   datalines;
2 3 5
;
proc netflow
   arcdata=arcdata condata=densecon;
run;
```

The SAS System does not uppercase character string values. PROC NETFLOW never uppercases node names, so the arcs in observations 1, 2, and 3 in the preceeding ARCDATA= data set have the default names "from_to1", "from_to2", and "TAIL_TO3", respectively. When the dense format of the CONDATA= data set is used, PROC NETFLOW does uppercase values of the NAME list variable, so the name of the arc in the second observation of the ARCDATA= data set is "ARC2". Thus, the second arc has two names; it's default "from_to2" and the other that was specified "ARC2".

As the SAS System does uppercases program code, you must think of the input statement

```
input from_to1 from_to2 arc2 tail_to3;
```

as really being

```
INPUT FROM_TO1 FROM_TO2 ARC2 TAIL_TO3;
```

The SAS variables named "FROM_TO1" and "FROM_TO2" are *not* associated with any of the arcs in the preceeding ARCDATA= data set. The values "FROM_TO1" and "FROM_TO2" are different from all of the arc names "from_to1", "from_to2", "TAIL_TO3", and "ARC2". "FROM_TO1" and "FROM_TO2" could end up being the names of two nonarc variables. It is sometimes useful to specify PRINT NONARCS; before commencing optimization to ensure that the model is correct (has the right set of nonarc variables).

The SAS variable named "ARC2" is the name of the second arc in the ARCDATA= data set, even though the name specified in the ARCDATA= data set looks like "arc2". The SAS variable named "TAIL_TO3" is the default name of the third arc in the ARCDATA= data set.

RHSOBS=*charstr*
specifies the keyword that identifies a right-hand-side observation when using the sparse format for data in the CONDATA= data set. The keyword is expected as a value of the SAS variable in the CONDATA= data set named in the COLUMN list specification. The default value of the RHSOBS= option is _RHS_ or _rhs_. If *charstr* is not a valid SAS variable name, enclose it in single quotes.

NARCS=*n*
specifies the approximate number of arcs. See the "How to Make the Data Read of PROC NETFLOW More Efficient" section on page 296.

NCOEFS=*n*
specifies approximate number of constraint coefficients. See the "How to Make the Data Read of PROC NETFLOW More Efficient" section on page 296.

NCONS=*n*
specifies approximate number of constraints. See the "How to Make the Data Read of PROC NETFLOW More Efficient" section on page 296.

NNAS=*n*
specifies approximate number of nonarc variables. See the "How to Make the Data Read of PROC NETFLOW More Efficient" section on page 296.

NNODES=*n*
specifies approximate number of nodes. See the "How to Make the Data Read of PROC NETFLOW More Efficient" section on page 296.

NON_REPLIC=*c*
prevents PROC NETFLOW from doing unnecessary checks of data previously read.

- NON_REPLIC=COEFS indicates that each constraint coefficient is specified *once* in the CONDATA= data set.
- NON_REPLIC=NONE indicates that constraint coefficients can be specified more than once in the CONDATA= data set. NON_REPLIC=NONE is the default.

See the "How to Make the Data Read of PROC NETFLOW More Efficient" section on page 296.

SAME_NONARC_DATA
SND
If all nonarc variable data are given in the ARCDATA= data set, or if 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 nonconstraint nonarc variable data in the CONDATA= data set is to be ignored. Only side constraint data in the CONDATA= data set are read.

If you use an unconstrained warm start and SAME_NONARC_DATA is not specified, any nonarc variable objective function coefficient, upper bound, or lower bound can be changed. Any nonarc variable data in the CONDATA= data set overrides (without warning messages) corresponding data in the ARCDATA= data set. You can possibly introduce new nonarc variables to the problem, that is, 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 the CONDATA= data set are to be deliberately ignored. Consider

```
proc netflow options arcdata=arc0 nodedata=node0
          condata=con0
             /* this data set has nonarc variable        */
             /* objective function coefficient data      */
          future1 arcout=arc1 nodeout=node1;
run;
data arc2;
  reset arc1;   /* this data set has nonarc variable obs */
  if _cost_<50.0 then _cost_=_cost_*1.25;
                /* some objective coefficients of nonarc */
                /* variable might be changed             */
proc netflow options
          warm arcdata=arc2 nodedata=node1
          condata=con0 same_nonarc_data
             /* This data set has old nonarc variable    */
             /* obj, fn. coefficients. same_nonarc_data  */
             /* indicates that the "new" coefs in the    */
             /* arcdata=arc2 are to be used.             */
run;
```

SCALE=*s*

indicates that the side constraints are to be scaled. Scaling is useful when some coefficients of a constraint or nonarc variable are either much larger or much smaller than other coefficients. Scaling might make all coefficients have values that have a smaller range, and this can make computations more stable numerically. Try the SCALE= option if PROC NETFLOW is unable to solve a problem because of numerical instability. Specify

- SCALE=ROW, SCALE=CON, or SCALE=CONSTRAINT if the largest absolute value of coefficients in each constraint is about 1.0
- SCALE=COL, SCALE=COLUMN, or SCALE=NONARC if nonarc variable columns are scaled so that the absolute value of the largest constraint coefficient of a nonarc variable is near to 1
- SCALE=BOTH if the largest absolute value of coefficients in each constraint, and the absolute value of the largest constraint coefficient of a nonarc variable is near to 1. This is the default
- SCALE=NONE if no scaling should be done

SHORTPATH
SP

specifies that PROC NETFLOW solve a shortest path problem. The NETFLOW procedure finds the shortest path between the nodes specified in the SOURCE= option and the SINK= option. The costs of arcs are their *lengths* . PROC NETFLOW automatically assigns a supply of one flow unit to the SOURCE= node, and the SINK= node is assigned to have a one flow unit demand. In this way, the SHORTPATH option sets up a shortest path problem as an equivalent minimum cost problem.

If a network has one supply node (with supply of one unit) and one demand node (with demand of one unit), you could specify the SHORTPATH option, with the SOURCE= and SINK= nodes, even if the problem is not a shortest path problem. You then should not provide any supply or demand data in the NODEDATA= data set or the ARCDATA= data set.

SINK=*sinkname*
SINKNODE=*sinkname*

identifies the demand node. The SINK= option is useful when you specify the MAXFLOW option or the SHORTPATH option and need to specify toward which node the shortest path or maximum flow is directed. The SINK= option also can be used when a minimum cost problem has only one demand node. Rather than having this information in the ARCDATA= data set or the NODEDATA= data set, use the SINK= option with an accompanying DEMAND= specification for this node. The SINK= option must be the name of a head node of at least one arc; thus, it must have a character value. If the the value of the SINK= option is not a valid SAS character variable name, it must be enclosed in single quotes and can contain embedded blanks.

SOURCE=*sourcename*
SOURCENODE=*sourcename*

identifies a supply node. The SOURCE= option is useful when you specify the MAXFLOW or the SHORTPATH option and need to specify from which node the shortest path or maximum flow originates. The SOURCE= option also can be used when a minimum cost problem has only one supply node. Rather than having this information in the the ARCDATA= data set or the NODEDATA= data set, use the SOURCE= option with an accompanying SUPPLY= amount of supply at this node. The SOURCE= option must be the name of a tail node of at least one arc; thus, it must have a character value. If the value of the SOURCE= option is not a valid SAS character variable name, it must be enclosed in single quotes and can contain embedded blanks.

SPARSECONDATA
SCDATA

indicates that the CONDATA= data set has data in the sparse data format. Otherwise, it is assumed that the data are in the dense format.

Note: If the SPARSECONDATA option is not specified, and you are running SAS software Version 6 or you have specified options validvarname=v6;, all NAME list variable values in the ARCDATA= data set are uppercased. See the "Case Sensitivity" section on page 279.

SPARSEP2
SP2

indicates that the new column of the working basis matrix that replaces another column be held in a linked list. If the SPARSEP2 option is not specified, a one-dimensional array is used to store this column's information, that can contain elements that are 0.0 and use more memory than the linked list. The linked list mechanism requires more work if the column has numerous nonzero elements in many iterations. Otherwise, it is superior. Sometimes, specifying SPARSEP2 is beneficial when the side constrained coefficient matrix is very sparse or when some paging is necessary.

SUPPLY=*s*

specifies the supply at the source node specified by the SOURCE= option. The SUPPLY= option should be used only if the SOURCE= option is given in the PROC NETFLOW statement and neither the SHORTPATH option nor the MAXFLOW option is specified. If you are solving a minimum cost network problem and the SOURCE= option is used to identify the source node and the SUPPLY= option is not specified, then by default the supply at the source node is made equal to the network's total demand.

THRUNET

tells PROC NETFLOW to force through the network any excess supply (the amount by which total supply exceeds total demand) or any excess demand (the amount by which total demand exceeds total supply) as is required. If a network problem has unequal total supply and total demand and the THRUNET option is not specified, PROC NETFLOW drains away the excess supply or excess demand in an optimal manner.

TYPEOBS=*charstr*

specifies the keyword that identifies a type observation when using the sparse format for data in the CONDATA= data set. The keyword is expected as a value of the SAS variable in the CONDATA= data set named in the COLUMN list specification. The default value of the TYPEOBS= option is _TYPE_ or _type_. If *charstr* is not a valid SAS variable name, enclose it in single quotes.

WARM

indicates that the NODEDATA= data set or the DUALIN= data set and the the ARCDATA= data set contain extra information of a warm start to be used by PROC NETFLOW. See the "Warm Starts" section on page 292.

CAPACITY Statement

CAPACITY *variable* **;**
CAPAC *variable* **;**

UPPERBD *variable* **;**

The CAPACITY statement identifies the SAS variable in the ARCDATA= data set that contains the maximum feasible flow or capacity of the network arcs. If an observation contains nonarc variable information, the CAPACITY list variable is the upper value bound for the nonarc variable named in the NAME list variable in that observation. The CAPACITY list variable must have numeric values. It is not necessary to have a CAPACITY statement if the name of the SAS variable is _CAPAC_, _UPPER_, _UPPERBD, or _HI_.

COEF Statement

COEF *variables* **;**

The COEF list is used with the sparse input format of the CONDATA= data set. The COEF list can contain more than one SAS variable, each of which must have numeric values. If the COEF statement is not specified, the CONDATA= data set is searched and SAS variables with names beginning with _COE are used. The number of SAS variables in the COEF list must be no greater than the number of SAS variables in the ROW list.

The values of the COEF list variables in an observation can be interpreted differently than these variables values in other observations. The values can be coefficients in the side constraints, costs and objective function coefficients, bound data, constraint type data, or rhs data. If the COLUMN list variable has a value that is a name of an arc or nonarc variable, the ith COEF list variable is associated with the constraint or special row name named in the ith ROW list variable. Otherwise, the COEF list variables indicate type values, rhs values, or missing values.

COLUMN Statement

COLUMN *variable* **;**

The COLUMN list is used with the sparse input format of side constraints. This list consists of one SAS variable in the CONDATA= data set that has as values the names of arc variables, nonarc variables, or missing values. Some, if not all of these values, also can be values of the NAME list variables of the ARCDATA= data set. The COLUMN list variable can have other special values (Refer to the TYPEOBS= and RHSOBS= options). If the COLUMN list is not specified after the PROC NET-

FLOW statement, the CONDATA= data set is searched and a SAS variable named _COLUMN_ is used. The COLUMN list variable must have character values.

CONOPT Statement

CONOPT;

The CONOPT statement has no options. It is equivalent to specifying RESET SCRATCH;. The CONOPT statement should be used before stage 2 optimization commences. It indicates that the optimization performed next should consider the side constraints.

Usually, the optimal unconstrained network solution is used as a starting solution for constrained optimization. Finding the unconstrained optimum usually reduces the amount of stage 2 optimization. Furthermore, the unconstrained optimum is almost always “closer” to the constrained optimum than the initial basic solution determined before any optimization is performed. However, as the optimum is approached during stage 1 optimization, the flow change candidates become scarcer and a solution good enough to start stage 2 optimization may already be at hand. You should then specify the CONOPT statement.

COST Statement

COST *variable* **;**
OBJFN *variable* **;**

The COST statement identifies the SAS variable in the ARCDATA= data set that contains the per unit flow cost through an arc. If an observation contains nonarc variable information, the value of the COST list variable is the objective function coefficient of the nonarc variable named in the NAME list variable in that observation. The COST list variable must have numeric values. It is not necessary to specify a COST statement if the name of the SAS variable is _COST_ or _LENGTH_

DEMAND Statement

DEMAND *variable* **;**

The DEMAND statement identifies the SAS variable in the ARCDATA= data set that contains the demand at the node named in the corresponding HEADNODE list variable. The DEMAND list variable must have numeric values. It is not necessary to have a DEMAND statement if the name of this SAS variable is _DEMAND_.

HEADNODE Statement

HEADNODE *variable* **;**
HEAD *variable* **;**

TONODE *variable* **;**

TO *variable* **;**

The HEADNODE statement specifies the SAS variable that must be present in the ARCDATA= data set that contains the names of nodes toward which arcs are directed. It is not necessary to have a HEADNODE statement if the name of the SAS variable is _HEAD_ or _TO_. The HEADNODE variable must have character values.

ID Statement

ID *variables* **;**

The ID statement specifies SAS variables containing values for pre- and post-optimal processing and analysis. These variables are not processed by PROC NETFLOW but are read by the procedure and written in the ARCOUT= and CONOUT= data sets and the output of PRINT statements. For example, imagine a network used to model a distribution system. The SAS variables listed on the ID statement can contain information on type of vehicle, transportation mode, condition of road, time to complete journey, name of driver, or other ancillary information useful for report writing or describing facets of the operation that do not have bearing on the optimization. The ID variables can be character, numeric, or both.

If no ID list is specified, the procedure forms an ID list of all SAS variables not included in any other implicit or explicit list specification. If the ID list is specified, any SAS variables in the ARCDATA= data set not in any list are dropped and do not appear in the ARCOUT= or CONOUT= data sets, or in the PRINT statement output.

LO Statement

LO *variable* **;**
LOWERBD *variable* **;**

MINFLOW *variable* **;**

The LO statement identifies the SAS variable in the ARCDATA= data set that contains the minimum feasible flow or lower flow bound for arcs in the network. If an observation contains nonarc variable information, the LO list variable has the value of the lower bound for the nonarc variable named in the NAME list variable. The LO list variables must have numeric values. It is not necessary to have a LO statement if the name of this SAS variable is _LOWER_, _LO_, _LOWERBD, or _MINFLOW

NAME Statement

NAME *variable* **;**
ARCNAME *variable* **;**

VARNAME *variable* **;**

Each arc and nonarc variable that has data in the CONDATA= data set must have a unique name. This name is a value of the NAME list variable. The NAME list variable must have character values (see the NAMECTRL= option in the PROC NETFLOW statement for more information). It is not necessary to have a NAME statement if the name of this SAS variable is _NAME_.

NODE Statement

NODE *variable* **;**

The NODE list variable, which must be present in the NODEDATA= data set, has names of nodes as values. These values must also be values of the TAILNODE list variable, the HEADNODE list variable, or both. If this list is not explicitly specified, the NODEDATA= data set is searched for a SAS variable with the name _NODE_. The NODE list variable must have character values.

PIVOT Statement

PIVOT;

The PIVOT statement has no options. It indicates that one Simplex iteration is to be performed. The PIVOT statement forces a Simplex iteration to be performed in spite of the continued presence of any reasons or solution conditions that caused optimization to be halted. For example, if the number of iterations performed exceeds the value of the MAXIT1= or MAXIT2= option and you issue a PIVOT statement, the iteration is performed even though the MAXIT1= or MAXIT2= value has not yet been changed using a RESET statement.

PRINT Statement

PRINT *options / qualifiers* **;**

The options available with the PRINT statement of PROC NETFLOW are summarized by purpose in Table 4.19.

Table 4.19. Functional Summary, PRINT statement: PROC NETFLOW

Description	Statement	Option
PRINT statement options		
display everything	PRINT	PROBLEM
display arc information	PRINT	ARCS
display nonarc variable information	PRINT	NONARCS
display variables information	PRINT	VARIABLES
display constraint information	PRINT	CONSTRAINTS
display information for some arcs	PRINT	SOME_ARCS
display information for some nonarc variables	PRINT	SOME_NONARCS
display information for some variables	PRINT	SOME_VARIABLES
display information for some constraints	PRINT	SOME_CONS
display information for some constraints associated with some arcs	PRINT	CON_ARCS
display information for some constraints associated with some nonarc variables	PRINT	CON_NONARCS
display information for some constraints associated with some variables	PRINT	CON_VARIABLES
PRINT statement qualifiers		
produce a short report	PRINT	/ SHORT
produce a long report	PRINT	/ LONG
only arcs (nonarc variables) with zero flow (value)	PRINT	/ ZERO
only arcs (nonarc variables) with nonzero flow (value)	PRINT	/ NONZERO
only for basics	PRINT	/ BASIC
only nonbasics	PRINT	/ NONBASIC

The PRINT statement enables you to examine parts or all of the problem. You can limit the amount of information displayed when a PRINT statement is processed by specifying PRINT statement options. The name of the PRINT option indicates what part of the problem is to be examined. If no options are specified, or PRINT PROBLEM is specified, information about the entire problem is produced.

The amount of displayed information can be limited further by following any PRINT statement options with a slash character (/) and one or more of the qualifiers SHORT or LONG, ZERO or NONZERO, BASIC or NONBASIC.

Some of the PRINT statement options require you to specify a list of some type of entity, thereby enabling you to indicate what entities are of interest. The entities of interest are the ones you want to display. These entities might be tail node names, head node names, nonarc variable names, or constraint names. The entity list is made up of one or more of the following constructs. Each construct can add none, one, or more entities to the set of entities to be displayed.

- _ALL_
  display all entities in the required list.
- entity
  display the named entity that is interesting.
- entity1 - entity2, (one hyphen)
  entity1 – entity2, (two hyphens)
  entity1 - CHARACTER - entity2, or
  entity1 - CHAR - entity2
  both entity1 and entity2 have names made up of the same character string prefix followed by a numeric suffix. The suffixes of both entity1 and entity2 have the same number of numerals but can have different values. A specification of entity1 - entity2 indicates that all entities with the same prefix and suffixes with values on or between the suffixes of entity1 and entity2 are to be put in the set of entities to be displayed. The numeric suffix of both entity1 and entity2 can be followed by a character string. For example, _OBS07_ - _OBS13_ is a valid construct of the forms entity1 - entity2.
- part_of_entity_name:
  display all entities in the required list that have names beginning with the character string preceding the colon.

The following options can appear in the PRINT statement.

- ARCS
  indicates that you want to have displayed information about all arcs.
- SOME_ARCS (*taillist*, *headlist*)
  is similar to the statement PRINT ARCS except that, instead of displaying information about all arcs, only arcs directed from nodes in a specified set of tail nodes to nodes in a specified set of head nodes are included. The nodes or node constructs belonging to the *taillist* list are separated by blanks. The nodes or node constructs belonging to the *headlist* list are also separated by blanks. The lists are separated by a comma.
- NONARCS
  indicates that information is to be displayed about all nonarc variables.

- SOME_NONARCS (*nonarclist*)
  is similar to the PRINT NONARCS statement except that, instead of displaying information about all nonarc variables, only those belonging to a specified set of nonarc variables have information displayed. The nonarc variables or nonarc variable constructs belonging to the *nonarclist* list are separated by blanks.
- CONSTRAINTS
  indicates that you want to have displayed information about all constraint coefficients.
- PRINT SOME_CONS (*conlist*)
  Information for coefficients in a specified set of constraints are displayed. The constraints or constraint constructs belonging to the *conlist* list are separated by blanks.
- CON_ARCS (*conlist* , *taillist* , *headlist*)
  is similar to the PRINT SOME_CONS (*conlist*) statement except that, instead of displaying information about all coefficients in specified constraints, information about only those coefficients that are associated with arcs directed from a set of specified tail nodes toward a set of specified head nodes is displayed. The constraints or constraint constructs belonging to the *conlist* list are separated by blanks; so too are the nodes or node constructs belonging to the *taillist* list and the nodes and node constructs belonging to the *headlist* list. The lists are separated by a comma.
- CON_NONARCS (*conlist* , *nonarclist*)
  is similar to the PRINT SOME_CONS (*conlist*) statement except that, instead of displaying information about all coefficients in specified constraints, information about only those coefficients that are associated with nonarc variables in a specified set is displayed. The constraints or constraint constructs belonging to the *conlist* list are separated by blanks. The nonarc variables or nonarc variable constructs belonging to the *nonarclist* list are separated by blanks. The lists are separated by a comma.
- PROBLEM
  is equivalent to the statement PRINT ARCS NONARCS CONSTRAINTS;

Following a slash (/), the qualifiers SHORT or LONG, ZERO or NONZERO, BASIC or NONBASIC can appear in any PRINT statement. These qualifiers are described below.

- BASIC
  only rows that are associated with arcs or nonarc variables that are basic are displayed. The _STATUS_ column values are KEY_ARC BASIC or NONKEY ARC BASIC for arcs, and NONKEY_BASIC for nonarc variables.
- LONG
  all table columns are displayed (the default when no qualifier is used).

- NONBASIC
  only rows that are associated with arcs or nonarc variables that are nonbasic are displayed. The _STATUS_ column values are LOWERBD NONBASIC or UPPERBD NONBASIC.
- NONZERO
  only rows that have nonzero _FLOW_ column values (nonzero arc flows, nonzero nonarc variable values) are displayed.
- SHORT
  the table columns are _N_, _FROM_, _TO_, _COST_, _CAPAC_, _LO_, _NAME_, and _FLOW_, or the names of the SAS variables specified in the corresponding variable lists (TAILNODE HEADNODE COST CAPACITY LO NAME lists). _COEF_ or the name of the SAS variable in the COEF list specification will head a column when the SHORT qualifier is used in PRINT CONSTRAINTS, SOME_CONS, CON_ARCS, or CON_NONARCS.
- ZERO
  Only rows that have zero _FLOW_ column values (zero arc flows, zero nonarc variable values) are displayed.

The default qualifiers are BASIC, NONBASIC, ZERO, NONZERO, and LONG.

Displaying Information On All Constraints

In the oil refinery problem, if you had entered

```
print constraints;
```

after the RUN statement, the output in Figure 4.9 would have been produced.

```
                              Oil Industry Example

                  Setting Up Condata = Cond1 For Proc Netflow

                               NETFLOW PROCEDURE

     _N_  _CON_     _type_       _rhs_  _name_    _from_       _to_

       1  _OBS1_    GE             -15  m_e_ref1  middle east  refinery 1
       2  _OBS1_    GE             -15  thruput1  refinery 1   r1
       3  _OBS2_    GE             -15  m_e_ref2  middle east  refinery 2
       4  _OBS2_    GE             -15  thruput2  refinery 2   r2
       5  _OBS3_    EQ               0  thruput1  refinery 1   r1
       6  _OBS3_    EQ               0  r1_gas    r1           ref1 gas
       7  _OBS4_    EQ               0  thruput2  refinery 2   r2
       8  _OBS4_    EQ               0  r2_gas    r2           ref2 gas

     _N_    _cost_   _capac_      _lo_  _SUPPLY_  _DEMAND_    _FLOW_    _COEF_

       1        63        95        20       100         .        80        -2
       2       200       175        50         .         .       145         1
       3        81        80        10       100         .        20        -2
       4       220       100        35         .         .        35         1
       5       200       175        50         .         .       145        -3
       6         0       140         0         .         .    108.75         4
       7       220       100        35         .         .        35        -3
       8         0       100         0         .         .     26.25         4

                    _N_   _FCOST_     _RCOST_    _STATUS_

                      1      5040           .    KEY_ARC    BASIC
                      2     29000           .    KEY_ARC    BASIC
                      3      1620           .    NONKEY ARC BASIC
                      4      7700          29    LOWERBD NONBASIC
                      5     29000           .    KEY_ARC    BASIC
                      6         0           .    KEY_ARC    BASIC
                      7      7700          29    LOWERBD NONBASIC
                      8         0           .    KEY_ARC    BASIC
```

Figure 4.9. print constraints

Displaying Information About Selected Arcs

In the oil refinery problem, if you had entered

```
print some_arcs(refin:,_all_)/short;
```

after the RUN statement, the output in Figure 4.10 would have been produced.

```
                              NETFLOW PROCEDURE

     _N_  _from_       _to_              _cost_   _capac_      _lo_  _name_

       1  refinery 1   r1                   200       175        50  thruput1
       2  refinery 2   r2                   220       100        35  thruput2

                                  _N_    _FLOW_

                                    1       145
                                    2        35
```

Figure 4.10. print some_arcs

Displaying Information About Selected Constraints

In the oil refinery problem, if you had entered

```
print some_cons(_obs3_-_obs4_)/nonzero short;
```

after the RUN statement, the output in Figure 4.11 would have been produced.

```
                              NETFLOW PROCEDURE

     _N_  _CON_     _type_       _rhs_  _name_    _from_       _to_

       1  _OBS3_    EQ               0  thruput1  refinery 1   r1
       2  _OBS3_    EQ               0  r1_gas    r1           ref1 gas
       3  _OBS4_    EQ               0  thruput2  refinery 2   r2
       4  _OBS4_    EQ               0  r2_gas    r2           ref2 gas

           _N_    _cost_     _capac_       _lo_      _FLOW_      _COEF_

             1       200         175         50         145          -3
             2         0         140          0      108.75           4
             3       220         100         35          35          -3
             4         0         100          0       26.25           4
```

Figure 4.11. print some_cons

If you had entered

```
print con_arcs(_all_,r1 r2,_all_)/short;
```

after the RUN statement, the output in Figure 4.12 would have been produced. Constraint information about arcs directed from selected tail nodes are displayed.

```
                              NETFLOW PROCEDURE

     _N_  _CON_     _type_       _rhs_  _name_    _from_       _to_

       1  _OBS3_    EQ               0  r1_gas    r1           ref1 gas
       2  _OBS4_    EQ               0  r2_gas    r2           ref2 gas

           _N_    _cost_     _capac_        _lo_      _FLOW_      _COEF_

             1         0         140           0      108.75           4
             2         0         100           0       26.25           4
```

Figure 4.12. print con_arcs

Cautions

This subsection has two parts; the first part is applicable if you are running Version 7 or later of the SAS System, and the second part is applicable if you are running Version 6. You can get Version 7 to "act" like Version 6 by specifying

```
options validvarname=v6;
```

For Version 7, PROC NETFLOW strictly respects case sensitivity. The PRINT statements of PROC NETFLOW that require lists of entities will work properly *only* if the entities have the same case as in the input data sets. Entities that contain blanks must be enclosed in single or double quotes. For example

```
print some_arcs (_all_,"Ref1 Gas");
```

In this example, a head node of an arc in the model is "Ref1 Gas" (without the quotes). If you omit the quotes, PROC NETFLOW issues a message on the SAS log:

```
WARNING: The node Ref1 in the list of head nodes in the PRINT
         SOME_ARCS or PRINT CON_ARCS is not a node in the
         problem. This statement will be ignored.
```

If you had specified

```
print some_arcs (_all_,"ref1 Gas");
```

(note the small r), you would have been warned

```
WARNING: The node ref1 Gas in the list of head nodes in the PRINT
         SOME_ARCS or PRINT CON_ARCS is not a node in the
         problem. This statement will be ignored.
```

If you are running Version 6 or Version 7 and you have specified

```
options validvarname=v6;
```

when information is parsed to procedures, the SAS System converts the text that makes up statements into uppercase. The PRINT statements of PROC NETFLOW that require lists of entities will work properly *only* if the entities are uppercase in the input data sets. If you do not want this uppercasing to occur, you must enclose the entity in single or double quotes.

```
print some_arcs('lowercase tail','lowercase head');
print some_cons('factory07'-'factory12');
print some_cons('_factory07_'-'_factory12_');
print some_nonarcs("CO2 content":);
```

Entities that contain blanks must be enclosed in single or double quotes.

QUIT Statement

QUIT;

The QUIT statement indicates that PROC NETFLOW is to be terminated immediately. The solution is not saved in the current output data sets. The QUIT statement has no options.

RESET Statement

RESET *options* **;**
SET *options* **;**

The RESET statement is used to change options after PROC NETFLOW has started execution. Any of the following options can appear in the PROC NETFLOW statement.

Another name for the RESET statement is SET. You can use SET when you are resetting options and RESET when you are setting options for the first time.

The following options fall roughly into five categories:

- output data set specifications
- options that indicate conditions under which optimization is to be halted temporarily, giving you an opportunity to use PROC NETFLOW interactively
- options that control aspects of the operation of the Network Primal Simplex optimization
- options that control the pricing strategies of the Network Simplex optimizer
- miscellaneous options

If you want to examine the setting of any options, use the SHOW statement. If you are interested in looking at only those options that fall into a particular category, the SHOW statement has options that enable you to do this.

The execution of PROC NETFLOW has three stages. In stage zero the problem data are read from the NODEDATA= ARCDATA= and CONDATA= data sets. If a warm start is not available, an initial basic feasible solution is found. Some options of the PROC NETFLOW statement control what occurs in stage zero. By the time the first RESET statement is processed, stage zero has already been completed.

In the first stage, an optimal solution to the network flow problem neglecting any side constraints is found. The primal and dual solutions for this relaxed problem can be saved in the ARCOUT= data set and the NODEOUT= data set, respectively.

In the second stage, the side constraints are examined and some initializations occur. Some preliminary work is also needed to commence optimization that considers the constraints. An optimal solution to the network flow problem with side constraints is found. The primal and dual solutions for this side-constrained problem are saved in the CONOUT= data set and the DUALOUT= data set, respectively.

Many options in the RESET statement have the same name except that they have as a suffix the numeral 1 or 2. Such options have much the same purpose, but option1 controls what occurs during the first stage when optimizing the network neglecting any side constraints and option2 controls what occurs in the second stage when PROC NETFLOW is performing constrained optimization.

Some options can be turned off by the option prefixed by the word *NO* . For example, FEASIBLEPAUSE1 may have been specified in a RESET statement and in a later RESET statement, you can specify NOFEASIBLEPAUSE1. In a later RESET statement, you can respecify FEASIBLEPAUSE1 and, in this way, toggle this option.

The options available with the PROC NETFLOW statement are summarized by purpose in Table 4.20.

Table 4.20. Functional Summary, RESET statement: PROC NETFLOW

Description	Statement	Option
Output Data Set Options		
unconstrained solution data set	RESET	ARCOUT=
unconstrained solution data set	RESET	NODEOUT=
constrained solution data set	RESET	CONOUT=
constrained solution data set	RESET	DUALOUT=
Stop Optimization Options		
pause after stage 1. Don't start stage 2	RESET	ENDPAUSE1
pause when feasible - stage 1	RESET	FEASIBLEPAUSE1
pause when feasible - stage 2	RESET	FEASIBLEPAUSE2
maximum number of iterations - stage 1	RESET	MAXIT1=
maximum number of iterations - stage 2	RESET	MAXIT2=
negates ENDPAUSE1	RESET	NOENDPAUSE1
negates FEASIBLEPAUSE1	RESET	NOFEASIBLEPAUSE1
negates FEASIBLEPAUSE2	RESET	NOFEASIBLEPAUSE2
pause every PAUSE1 iterations - stage 1	RESET	PAUSE1=
pause every PAUSE2 iterations - stage 2	RESET	PAUSE2=
Simplex Options		
Use Big M instead of twophase - stage 1	RESET	BIGM1
Use Big M instead of twophase - stage 2	RESET	BIGM2
anti-cycling option	RESET	CYCLEMULT1
interchange first eligible nonkey with leaving key variable	RESET	INTFIRST
invert working basis matrix (2-dim) every INVFREQ= iterations	RESET	INVFREQ=
maximum number of L row operations allowed before refactorization done instead of factor column update	RESET	MAXL=
maximum number fo LU factor column updates, otherwise refactorize	RESET	MAXLUUPDATES=
anticycling option	RESET	MINBLOCK1
first eligible leaving variable used, otherwise best is used	RESET	LRATIO1
first eligible leaving variable used, otherwise best is used	RESET	LRATIO2
otherwise best is used		
negates INTFIRST	RESET	NOINTFIRST
negates LRATIO1	RESET	NOLRATIO1
negates LRATIO2	RESET	NOLRATIO2
negates PERTURB1	RESET	NOPERTURB1
anti-cycling option	RESET	PERTURB1

Description	Statement	Option
re-factorize working basis matrix every REFACTFREQ= iterations	RESET	REFACTFREQ=
use twophase instead of Big M - stage 1	RESET	TWOPHASE1
use twophase instead of Big M - stage 2	RESET	TWOPHASE2
pivot element selection parameter	RESET	U=
zero tolerance - stage 1	RESET	ZERO1=
zero tolerance - stage 2	RESET	ZERO2=
zero tolerance - real number comparisons	RESET	ZEROTOL=
Pricing Options		
calculate dual values every DUALFREQ= iterations after major iterations	RESET	DUALFREQ=
pricing strategy - stage 1	RESET	PRICETYPE1=
pricing strategy - stage 2	RESET	PRICETYPE2=
used when P1SCAN=PARTIAL	RESET	P1NPARTIAL=
how nonbasis variables (PRICETYPE1=NOQ) or queue candidates (PRICETYPE1=Q) are scanned	RESET	P1SCAN=
used when P2SCAN=PARTIAL	RESET	P2NPARTIAL=
how nonbasis variables (PRICETYPE2=NOQ) or queue candidates (PRICETYPE2=Q) are scanned	RESET	P2SCAN=
initial queue size - stage 1	RESET	QSIZE1=
initial queue size - stage 2	RESET	QSIZE2=
used when Q1FILLSCAN=PARTIAL	RESET	Q1FILLNPARTIAL=
how candidates are scanned when filling queue - stage 1	RESET	Q1FILLSCAN=
used when Q2FILLSCAN=PARTIAL	RESET	Q2FILLNPARTIAL=
how candidates are scanned when filling queue - stage 2	RESET	Q2FILLSCAN=
queue size reduction factor	RESET	REDUCEQSIZE1=
queue size reduction factor	RESET	REDUCEQSIZE2=
when the queue are refreshed - stage 1	RESET	REFRESHQ1=
when the queue are refreshed - stage 2	RESET	REFRESHQ2=
Miscellaneous Options		
output complete basis information to ARCOUT and NODEOUT data sets	RESET	FUTURE1
output complete basis information to CONOUT and DUALOUT data sets	RESET	FUTURE2
turn off infeasibility or optimality flags as more optimization is to be done	RESET	MOREOPT
negates FUTURE1	RESET	NOFUTURE1
negates FUTURE2	RESET	NOFUTURE2
negates SCRATCH	RESET	NOSCRATCH

Description	Statement	Option
negates ZTOL1	RESET	NOZTOL1
negates ZTOL2	RESET	NOZTOL2
do not do stage 1 optimization. Do stage 2 optimization	RESET	SCRATCH
display this number of similar SAS log messages, suppress the rest	RESET	VERBOSE=
use zero tolerance- stage 1	RESET	ZTOL1
use zero tolerance- stage 2	RESET	ZTOL2
Interior Point algorithm Options		
allowed amount of dual infeasibility	RESET	TOLDINF=
allowed amount of primal infeasibility	RESET	TOLPINF=
cut-off tolerance for Cholesky factorization	RESET	CHOLTINYTOL=
density threshold for Cholesky processing	RESET	DENSETHR=
maximum number of Interior Point algorithm iterations	RESET	MAXITERB=
Primal-Dual (Duality) gap tolerance	RESET	PDGAPTOL=
step-length multiplier	RESET	PDSTEPMULT=
preprocessing type	RESET	PRSLTYPE=

Output Data Set Specifications

In a RESET statement, you can specify an ARCOUT= data set, a NODEOUT= data set, a CONOUT= data set, or a DUALOUT= data set. You are advised to specify these output data sets early because if you make a syntax error when using PROC NETFLOW interactively or, for some other reason, PROC NETFLOW encounters or does something unexpected, these data sets will contain information about the solution that was reached. If you had specified the FUTURE1 or FUTURE2 option in a RESET statement, PROC NETFLOW may be able to resume optimization in a subsequent run.

You can turn off these current output data set specifications by specifying ARCOUT=NULL, NODEOUT=NULL, CONOUT=NULL, or DUALOUT=NULL.

If PROC NETFLOW is outputting observations to an output data set and you want this to stop, press the keys used to stop SAS procedures. PROC NETFLOW waiys, if necessary, and then executes the next statement.

ARCOUT=*SAS-data-set*
AOUT=*SAS-data-set*

names the output data set that receives all information concerning arc and nonarc variables, including flows and and other information concerning the current solution and the supply and demand information. The current solution is the latest solution found by the optimizer when the optimization neglecting side constraints is halted or the unconstrained optimum is reached.

You can specify an ARCOUT= data set in any RESET statement before the unconstrained optimum is found (even at commencement). Once the unconstrained optimum has been reached, use the SAVE statement to produce observations in an ARCOUT= data set. Once optimization that considers constraints starts, you are will not unable to obtain an ARCOUT= data set. Instead, use a CONOUT= data set to get the current solution. See the “ARCOUT= and CONOUT= Data Sets” section on page 276 for more information.

CONOUT=*SAS-data-set*
COUT=*SAS-data-set*

names the output data set that contains the primal solution obtained after optimization considering side constraints reaches the optimal solution. You can specify a CONOUT= data set in any RESET statement before the constrained optimum is found (even at commencement or while optimizing neglecting constraints). Once the constrained optimum has been reached, or during stage 2 optimization, use the SAVE statement to produce observations in a CONOUT= data set. See the “ARCOUT= and CONOUT= Data Sets” section on page 276 for more information.

DUALOUT=*SAS-data-set*
DOUT=*SAS-data-set*

names the output data set that contains the dual solution obtained after doing optimization that considering side constraints reaches the optimal solution. You can specify a DUALOUT= data set in any RESET statement before the constrained optimum is found (even at commencement or while optimizing neglecting constraints). Once the constrained optimum has been reached, or during stage 2 optimization, use the SAVE statement to produce observations in a DUALOUT= data set. See the “NODEOUT= and DUALOUT= Data Sets” section on page 277 for more information.

NODEOUT=*SAS-data-set*
NOUT=*SAS-data-set*

names the output data set that receives all information about nodes (supply/demand and nodal dual variable values) and other information concerning the unconstrained optimal solution.

You can specify a NODEOUT= data set in any RESET statement before the unconstrained optimum is found (even at commencement). Once the unconstrained optimum has been reached, or during stage 1 optimization, use the SAVE statement to produce observations in a NODEOUT= data set. Once optimization that considers constraints starts, you will not be able to obtain a NODEOUT= data set. Instead use a DUALOUT= data set to get the current solution. See the “NODEOUT= and DUALOUT= Data Sets” section on page 277 for more information.

Options to Halt Optimization

The following options indicate conditions when optimization is to be halted. You then have a chance to use PROC NETFLOW interactively. If the NETFLOW procedure is optimizing and you want optimization to halt immediately, press the CTRL-BREAK key combination used to stop SAS procedures. Doing this is equivalent to PROC NETFLOW finding that some prespecified condition of the current solution under which optimization should stop has occurred.

If optimization does halt, you may need to change the conditions for when optimization should stop again. For example, if the number of iterations exceeded MAXIT2, use the RESET statement to specify a larger value for the MAXIT2= option before the next RUN statement. Otherwise, PROC NETFLOW will immediately find that the number of iterations still exceeds MAXIT2= and halt without doing any additional optimization.

ENDPAUSE1

indicates that PROC NETFLOW will pause after the unconstrained optimal solution has been obtained and information about this solution has been output to the current ARCOUT= data set, NODEOUT= data set, or both. The procedure then executes the next statement, or waits if no subsequent statement has been specified.

FEASIBLEPAUSE1
FP1

indicates that unconstrained optimization should stop once a feasible solution is reached. PROC NETFLOW checks for feasibility every 10 iterations. A solution is feasible if there are no artificial arcs having nonzero flow assigned to be conveyed through them. The presence of artificial arcs with nonzero flows means that the current solution does not satisfy all the nodal flow conservation constraints implicit in network problems.

MAXIT1=*m*

specifies the maximum number of Primal Simplex iterations PROC NETFLOW is to perform in stage 1. The default value for the MAXIT1= option is 1000. If MAXIT1= iterations are performed and you want to continue unconstrained optimization, reset MAXIT1= to a number larger than the number of iterations already performed and issue another RUN statement.

NOENDPAUSE1
NOEP1

negates the ENDPAUSE1 option.

NOFEASIBLEPAUSE1
NOFP1

negates the FEASIBLEPAUSE1 option.

PAUSE1=*p*

indicates that PROC NETFLOW will halt unconstrained optimization and pause when the remainder of the number of stage 1 iterations divided by he value of the PAUSE1= option is zero. If present, the next statement is executed; if not, the procedure waits for the next statement to be specified. The default value for PAUSE1= is 999999.

FEASIBLEPAUSE2
FP2
NOFEASIBLEPAUSE2
NOFP2
PAUSE2=*p*
MAXIT2=*m*

are the stage 2 constrained optimization counterparts of the options described previously and having as a suffix the numeral 1.

Options Controlling the Network Simplex Optimization

BIGM1
NOTWOPHASE1
TWOPHASE1
NOBIGM1

BIGM1 indicates that the "big-M" approach to optimization is used. Artificial variables are treated like real arcs, slacks, surpluses and nonarc variables. Artificials have very expensive costs. BIGM1 is the default.

TWOPHASE1 indicates that the two-phase approach is used instead of the big-M approach. At first, artificial variables are the only variables to have nonzero objective function coefficients. An artificial's objective function coefficient is temporarily set to 1 and PROC NETFLOW minimizes. When all artificial variables have zero value, PROC NETFLOW has found a feasible solution, and phase 2 commences. Arcs and nonarc variables have their real costs and objective function coefficients.

Before all artificial variables are driven to have zero value, you can toggle between the big-M and the two-phase approaches by specifying BIGM1 or TWOPHASE1 in a RESET statement. The option NOTWOPHASE1 is synonymous with BIGM1, and NOBIGM1 is synonymous with TWOPHASE1.

CYCLEMULT1=*c*
MINBLOCK1=*m*
NOPERTURB1
PERTURB1

In an effort to reduce the number of iterations performed when the problem is highly degenerate, PROC NETFLOW has in stage 1 optimization adopted an algorithm outlined in Ryan and Osborne 1988.

If the number of consecutive degenerate pivots (those with no progress toward the optimum) performed equals the value of the CYCLEMULT1= option times the number of nodes, the arcs that were "blocking" (can leave the basis) are added to 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. In other words, perference is given to arcs that "block" many iterations. After several iterations, the list is cleared.

If the number of blocking arcs is less than the value of the MINBLOCK1= option, a list is not kept. Otherwise, if PERTURB1 is specified, the arc flows are perturbed by a random quantity, so that arcs on the list that block subsequent iterations are chosen to leave the basis randomly. Although perturbation often pays off, it is computationally expensive. Periodically, PROC NETFLOW has to clear out the lists and un-perturb the solution. You can specify NOPERTURB1 to prevent perturbation occuring.

Defaults are CYCLEMULT1=0.15, MINBLOCK1=2, and NOPERTURB1.

LRATIO1

specifies the type of ratio test to use in determining which arc leaves the basis in stage 1. In some iterations, more than one arc is eligible to leave the basis. Of those arcs that can leave the basis, the leaving arc is the first encountered by the algorithm if the LRATIO1 option is specified. Specifying the LRATIO1 option can decrease the chance of cycling but can increase solution times. The alternative to the LRATIO1 option is the NOLRATIO1 option, which is the default.

LRATIO2

specifies the type of ratio test to use in determining what leaves the basis in stage 2. In some iterations, more than one arc, constraint slack, surplus, or nonarc variable is eligible to leave the basis. If the LRATIO2 option is specified, the leaving arc, constraint slack, surplus, or nonarc variable is the one that is eligible to leave the basis first encountered by the algorithm. Specifying the LRATIO2 option can decrease the chance of cycling but can increase solution times. The alternative to the LRATIO2 option is the NOLRATIO2 option, which is the default.

NOLRATIO1

specifies the type of ratio test to use in determining which arc leaves the basis in stage 1. If the NOLRATIO1 option is specified, of those arcs that can leave the basis, the leaving arc has the minimum (maximum) cost if the leaving arc is to be nonbasic with flow capacity equal to its capacity (lower flow bound). If more than one possible leaving arc has the minimum (maximum) cost, the first such arc encountered is chosen. Specifying the NOLRATIO1 option can decrease solution times, but can increase the chance of cycling. The alternative to the NOLRATIO1 option is the LRATIO1 option. The NOLRATIO1 option is the default.

NOLRATIO2

specifies the type of ratio test to use in determining which arc leaves the basis in stage 2. If the NOLRATIO2 option is specified, the leaving arc, constraint slack, surplus, or nonarc variable is the one eligible to leave the basis with the minimum (maximum) cost or objective function coefficient if the leaving arc, constraint slack or nonarc variable is to be nonbasic with flow or value equal to its capacity or upper value bound (lower flow or value bound), respectively. If several possible leaving arcs, constraint slacks, surpluses, or nonarc variables have the minimum (maximum) cost or objective function coefficient, then the first encountered is chosen. Specifying the NOLRATIO2 option can decrease solution times, but can increase the chance of cycling. The alternative to the NOLRATIO2 option is the LRATIO2 option. The NOLRATIO2 option is the default.

Options Applicable to Constrained Optimization

The INVFREQ= option is relevant only if INVD_2D is specified in the PROC NETFLOW statement; that is, the inverse of the working basis matrix is being stored and processed as a two-dimensional array. The REFACTFREQ=, U= , MAXLUUPDATES=, and MAXL= options are relevant if the INVD_2D option is not specified in the PROC NETFLOW statement; that is, if the working basis matrix is **LU** factored.

BIGM2
NOTWOPHASE2
TWOPHASE2
NOBIGM2

are the stage 2 constrained optimization counterparts of the options BIGM1, NOTWOPHASE1, TWOPHASE1, and NOBIGM1.

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

INVFREQ=*r*

recalculates the working basis matrix inverse whenever n iterations have been performed where n is the value of the INVFREQ= option. Although a relatively expensive task, it is prudent to do as roundoff errors accumulate, especially affecting the elements of this matrix inverse. The default is INVFREQ=50. The INVFREQ= option should be used only if the INVD_2D option is specified in the PROC NETFLOW statement.

INTFIRST

In some iterations, it is found that what must leave the basis is an arc that is part of the spanning tree representation of the network part of the basis, (called a *a key* arc). It is necessary to interchange another basic arc not part of the tree (called a *nonkey arc*) with the tree arc that leaves to permit the basis update to be performed efficiently. Specifying the INTFIRST option indicates that of the nonkey arcs eligible to be swapped with the leaving key arc, the one chosen to do so is the first encountered by the algorithm. If the INTFIRST option is not specified, all such arcs are examined and the one with the best cost is chosen.

The terms *key* and *nonkey* are used because the algorithm used by PROC NETFLOW for network optimization considering side constraints, (GUB-based, Primal Partitioning, or Factorization) is a variant of an algorithm originally developed to solve linear programming problems with generalized upper bounding constraints. The terms *key* and *nonkey* were coined then. The STATUS SAS variable in the ARCOUT= and CONOUT= data sets and the STATUS column in tables produced when PRINT statements are processed indicate whether basic arcs are key or nonkey. Basic nonarc variables are always nonkey.

MAXL=*m*

If the working basis matrix is **LU** factored, **U** is an upper triangular matrix and **L** is a lower triangular matrix corresponding to a sequence of elementary matrix row operations required to change the working basis matrix into **U** . **L** and **U** enable substitution techniques to be used to solve the linear systems of the Simplex algorithm. Among other things, the **LU** processing strives to keep the number of **L** elementary matrix row operation matrices small. A buildup in the number of these could indicate that fill-in is becoming excessive and the computations involving **L** and **U** will be hampered. Refactorization should be performed to restore **U** sparsity and reduce **L** information. When the number of **L** matrix row operations exceeds the value of the MAXL= option, a refactorization is done rather than one or more updates. The default value for MAXL= is 10 times the number of side constraints. The MAXL= option should not be used if INVD_2D is specified in the PROC NETFLOW statement.

MAXLUUPDATES=*m*
MLUU=*m*

In some iterations, PROC NETFLOW must either perform a series of single column updates or a complete refactorization of the working basis matrix. More than one column of the working basis matrix must change before the next Simplex iteration can begin. The single column updates can often be done faster than a complete refactorization, especially if few updates are necessary, the working basis matrix is sparse, or a refactorization has been performed recently. If the number of columns that must change is less than the value specified in the MAXLUUPDATES= option, the updates are attempted; otherwise, a refactorization is done. Refactorization also occurs if the sum of the number of columns that must be changed and the number of **LU** updates done since the last refactorization exceeds the value of the REFACTFREQ= option. The MAXLUUPDATES= option should not be used if the INVD_2D option is specified in the PROC NETFLOW statement.

In some iterations, a series of single column updates are not able to complete the changes required for a working basis matrix because, ideally, all columns should change at once. If the update cannot be completed, PROC NETFLOW performs a refactorization. The default value is 5.

NOINTFIRST

indicates that of the arcs eligible to be swapped with the leaving arc, the one chosen to do so has the best cost. See the INTFIRST option.

REFACTFREQ=*r*
RFF=*r*

specifies the maximum number of **L** and **U** updates between refactorization of the working basis matrix to reinitialize **LU** factors. In most iterations, one or several Bartels-Golub updates can be performed. An update is performed more quickly than a complete refactorization. However, after a series of updates, the sparsity of the **U** factor is degraded. A refactorization is necessary to regain sparsity and to make subsequent computations and updates more efficient. The default value is 50. The REFACTFREQ= option should not be used if INVD_2D is specified in the PROC NETFLOW statement.

U=*u*

controls the choice of pivot during **LU** decomposition or Bartels-Golub update. When searching for a pivot, any element less than the value of the U= option times the largest element in its matrix row is excluded, or matrix rows are interchanged to improve numerical stability. The U= option should have values on or between ZERO2 and 1.0. Decreasing the value of the U= option biases the algorithm toward maintaining sparsity at the expense of numerical stability and vice-versa. Reid 1975 suggests that the value of 0.01 is acceptable and this is the default for the U= option. This option should not be used if INVD_2D is specified in the PROC NETFLOW statement.

Pricing Strategy Options

There are three main types of pricing strategies

- PRICETYPEx=NOQ
- PRICETYPEx=BLAND
- PRICETYPEx=Q

The one that usually performs better than the others is PRICETYPEx=Q, so this is the default.

Because the pricing strategy takes a lot of computational time, you should experiment with the following options to find the optimum specification. These options influence how the pricing step of the Simplex iteration is performed. See the "Pricing Strategies" section on page 280 for further information).

PRICETYPEx=BLAND or PTYPEx=BLAND

PRICETYPEx=NOQ or PTYPEx=NOQ

- PxSCAN=BEST
- PxSCAN=FIRST
- PxSCAN=PARTIAL and PxNPARTIAL=*p*

PRICETYPEx=Q or PTYPEx=Q
QSIZEx=*q* or Qx=*q*
REFRESHQx=*r*
REDUCEQSIZEx=*r*
REDUCEQx=*r*

- PxSCAN=BEST
- PxSCAN=FIRST
- PxSCAN=PARTIAL and PxNPARTIAL=*p*
- QxFILLSCAN=BEST
- QxFILLSCAN=FIRST
- QxFILLSCAN=PARTIAL and QxFILLNPARTIAL=*q*

For stage 2 optimization, you can specify P2SCAN=ANY. P2SCAN=ANY is used in conjunction with the DUALFREQ= option.

Miscellaneous Options

FUTURE1

signals that PROC NETFLOW must output extra observations to the NODEOUT= and ARCOUT= data sets. These observations contain information about the solution found by doing optimization neglecting any side constraints. These two data sets then can be used as the NODEDATA= and ARCDATA= data sets, respectively, in subsequent PROC NETFLOW runs with the WARM option specified. See the "Warm Starts" section on page 292.

FUTURE2

signals that PROC NETFLOW must output extra observations to the DUALOUT= and CONOUT= data sets. These observations contain information about the solution found by optimization that considers side constraints. These two data sets then can be used as the NODEDATA= data set (also called the DUALIN= data set) and the ARCDATA= data sets, respectively, in subsequent PROC NETFLOW runs with the WARM option specified. See the "Warm Starts" section on page 292.

MOREOPT

The MOREOPT option turns off all optimality and infeasibility flags that may have been raised. Unless this is done, PROC NETFLOW will not do any optimization when a RUN statement is specified.

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 can try resetting options (particularly zero tolerances) in the hope that the infeasibility was raised incorrectly.

Consider the following example:

```
proc netflow
   nodedata=noded         /* supply and demand data   */
   arcdata=arcd1          /* the arc descriptions     */
   condata=cond1          /* the side constraints     */
   conout=solution;       /* output 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 */
```

If PROC NETFLOW finds an optimal solution, you might want to do additional optimization to confirm that an optimum has really been reached. Specify the MOREOPT option in a RESET statement. Reset options, but, in this case, tighten zero tolerances.

NOFUTURE1

negates the FUTURE1 option.

NOFUTURE2

negates the FUTURE2 option.

NOSCRATCH

negates the SCRATCH option.

NOZTOL1

indicates that the majority of tests for roundoff error should not be done. Specifying the NOZTOL1 option and obtaining the same optimal solution as when the the NOZTOL1 option is not specified in the PROC NETFLOW statement (or the ZTOL1 option is specified), verifies that the zero tolerances were not too high. Roundoff error checks that are critical to the successful functioning of PROC NETFLOW and any related readjustments are always done.

NOZTOL2

indicates that the majority of tests for roundoff error are not to be done during an optimization that considers side constraints. The reasons for specifying the NOZTOL2 option are the same as those for specifying the NOZTOL1 option for stage 1 optimization (see the NOZTOL1 option).

SCRATCH

specifies that you do not want PROC NETFLOW to enter or continue stage 1 of the algorithm. Rather than specify RESET SCRATCH, you can use the CONOPT statement.

VERBOSE=*v*

limits the number of similar messages that are displayed on the SAS log.

For example, when reading the ARCDATA= data set, PROC NETFLOW might have cause to issue the following message many times:

```
ERROR: The HEAD list variable value in obs i in the ARCDATA is
       missing, - the TAIL list variable value of this obs
       is nonmissing. This is an incomplete arc specification.
```

If there are lots of observations that have this fault, messages that are similar are issued for only the first VERBOSE= such observations. After the ARCDATA= data set has been read, PROC NETFLOW will issue the message:

```
NOTE: More messages similar to the ones immediately above
      could have been issued but were suppressed as
      VERBOSE= v.
```

If observations in the ARCDATA= data set have this error, PROC NETFLOW stops and you have to fix the data. Imagine that this error is only a warning and PROC NETFLOW proceeded to other operations such as reading the CONDATA= data set. If PROC NETFLOW finds there are numerous errors when reading that data set, the number of messages issued to the SAS log are also limited by the VERBOSE= option.

If you have a problem with a large number of side constraints and for some reason you stop stage 2 optimization early, PROC NETFLOW indicates that constraints are disobeyed by the current solution. Specifying VERBOSE=*v* allows at most *v* disobeyed constraints to be written to the log. If there are more, these are not displayed.

When PROC NETFLOW finishes and messages have been suppressed, the message

```
NOTE: To see all messages, specify VERBOSE=v.
```

is issued. The value of *v* is the smallest value that should be specified for the VERBOSE= option so that *all* messages are displayed if PROC NETFLOW is run again with the same data and everything else except that the VERBOSE= option is unchanged. No messages are suppressed.

The default value for the VERBOSE= option is 12.

ZERO1=*z*
Z1=*z*

specifies the zero tolerance level in stage 1. If the NOZTOL1 option is not specified, values within z of zero are set to 0.0, where z is the value of the ZERO1= option. Flows close to the lower flow bound or capacity of arcs are reassigned those exact values. Two values are deemed to be close if one is within z of the other. The default value for the ZERO1= option is 0.000001. Any value specified for the ZERO1= option that is < 0.0 or > 0.0001 is invalid.

ZERO2=*z*
Z2=*z*

specifies the zero tolerance level in stage 2. If the NOZTOL2 option is not specified, values within z of zero are set to 0.0, where z is the value of the ZERO2= option. Flows close to the lower flow bound or capacity of arcs are reassigned those exact values. If there are nonarc variables, values close to the lower or upper value bound of nonarc variables are reassigned those exact values. Two values are deemed to be close if one is within z of the other. The default value for the ZERO2= option is 0.000001. Any value specified for the ZERO2= option that is < 0.0 or > 0.0001 is invalid.

ZEROTOL=*z*

specifies the zero tolerance used when PROC NETFLOW must compare any real number with another real number, or zero. For example, if x and y are real numbers, then for x to be considered greater than y, x must be at least $y + ZEROTOL$ The ZEROTOL= option is used throughout any PROC NETFLOW run.

ZEROTOL=z controls the way PROC NETFLOW performs all double precision comparisons; that is whether a double precision number is equal to, not equal to, greater than (or equal to), or less than (or equal to) zero or some other double precision number. A double precision number is deemed to be the same as another such value if the absolute differences between them is less than or equal to the value of the ZEROTOL= option.

The default value for the ZEROTOL= option is 1.0E-14. You can specify the ZEROTOL= option in the NETFLOW or RESET statement. Valid values for the ZEROTOL= option must be > 0.0 and < 0.0001. Do not specify a value too close to zero as this defeats the purpose of the ZEROTOL= option. Neither should the value be too large, as comparisons might be incorrectly performed.

The ZEROTOL= option is different from the ZERO1= and ZERO2= options in that ZERO1= and ZERO2= options work when determining whether optimality has been reached whether an entry in the updated column in the ratio test of the simplex method is zero, whether a flow is the same as the arc's capacity or lower bound or whether the value of a nonarc variable is at a bound. The ZEROTOL= option is used in all other general double precision number comparisons.

ZTOL1

indicates that all tests for roundoff error are performed during stage 1 optimization. Any alterations are carried out. The opposite of the ZTOL1 option is the NOZTOL1 option.

ZTOL2

indicates that all tests for roundoff error are performed during stage 2 optimization. Any alterations are carried out. The opposite of the ZTOL2 option is the NOZTOL2 option.

Interior Point Algorithm Options

INTPOINT

indicates that the Interior Point algorithm is to be used. The INTPOINT option must be specified if you want the Interior Point algorithm to be used for solving network problems, otherwise the Simplex algorithm is used instead. For Linear Programming problems (problems with no network component), PROC NETFLOW must use the Interior Point algorithm, so you need not specify the INTPOINT option.

TOLDINF=*t*
RTOLDINF=*t*

specifies the allowed amount of dual infeasibility. In the "Interior Point Algorithmic Details" section on page 316, the vector $infeas_c$ is defined. If all elements of this vector are $\leq$ *t*, the solution is deemed feasible. $infeas_c$ is replaced by a zero vector, which makes computations faster. This option is the dual equivalent to the TOLPINF= option. Valid values for *t* are between $1.0E-12$ and $1.0E-1$. The default is $1.0E-7$.

TOLPINF=*t*
RTOLPINF=*t*

specifies the allowed amount of primal infeasibility. This option is the dual equivalent to the TOLDINF= option. In the "Interior Point Algorithmic Details" section on page 316, the vector $infeas_b$ is defined. If all elements of this vector are $\leq$ *t*, the solution is deemed feasible. $infeas_b$ is replaced by a zero vector, which makes computations faster. Increasing the value of the TOLPINF= option too much can lead to instability, but a modest increase can give the algorithm added flexibility and decrease the iteration count. Valid values for *t* are between $1.0E-12$ and $1.0E-1$. The default is $1.0E-7$.

CHOLTINYTOL=*c*
RCHOLTINYTOL=*c*

specifies the cut-off tolerance for Cholesky factorization of the $A\Theta A^{-1}$. If a diagonal value drops below *c*, the row is essentially treated as dependent and is ignored in the factorization. Valid values for *c* are between $1.0E-30$ and $1.0E-6$. The defualt value is $1.0E-8$.

DENSETHR=*d*
RDENSETHR=*d*

specifies the density threshold for Cholesky processing. When the symbolic factorization encounters a column of L that has DENSETHR= proportion of nonzeros and the remaining part of L is at least 12 x 12, the remainder of L is treated as dense. In practice, the lower right part of the Cholesky triangular factor L is quite dense and it can be computationally more efficient to treat it as 100% dense. The default value for *d* is 0.7. A specification of *d*$\leq$0.0 causes all dense processing; *d*$\geq$1.0 causes all sparse processing.

MAXITERB=*m*
IMAXITERB=*m*

specifies the maximum number of iterations of the IPBM algorithm that can be performed. The default value for *m* is 100. One of the most remarkable aspects of the Interior Point algorithm is that for most problems, it usually needs to do less than 50 iterations, *no matter the size of the problem*.

PDGAPTOL=*p*
RPDGAPTOL=*p*

specifies the Primal-Dual gap or Duality gap tolerance. Duality gap is defined in the "Interior Point Algorithmic Details" section on page 316. If the relative gap $(dualitygap/(c^T x))$ between the primal and dual objectives is smaller the the value of the PDGAPTOL= option and both the primal and dual problems are feasible, then PROC NETFLOW stops optimization with a solution that is deemed optimal. Valid values for *p* are between $1.0E-12$ and $1.0E-1$. the default is $1.0E-7$.

PDSTEPMULT=*p*
RPDSTEPMULT=*p*

specifies the step-length multiplier. The maximum feasible step-length chosen by the Primal-Dual with Predictor-Corrector algorithm is multiplied by the value of the PDSTEPMULT= option. This number must be less than 1 to avoid moving beyond the barrier. An actual step-length greater than 1 indicates numerical difficulties. Valid values for *p* are between 0.01 and 0.999999. The default value is 0.99995.

In the "Interior Point Algorithmic Details" section on page 316, the solution of the next iteration is obtained by moving along a direction from the current iterations solution.
$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \alpha(\Delta x^k, \Delta y^k, \Delta s^k)$

Let $\alpha = Min_i\{\alpha : x_i^k + \alpha\Delta x = 0 | s_i^k + \alpha\Delta s = 0\}$. If $\alpha \leq 1$, then $\alpha =$*p*α. α is a value as large as possible but ≤ 1.0 and not so large that a x_i^{k+1} or s_i^{k+1} of some variable i is not "too close" to zero.

PRSLTYPE=*p*
IPRSLTYPE=*p*

Preprocessing the Linear Programming problem often succeeds in allowing some variables and constraints to be temporarily eliminated from the LP that must be solved. This reduces the solution time and possibly also the chance that the optimizer will run into numerical differculties. The task of preprocessing is inexpensive to do.

You control how much preprocessing to do by specifying the PRSLTYPE= *p*, where *p* can be -1, 0, 1, 2, or 3.

-1	do not perform preprocessing. For most problems, specifying PRSLTYPE= -1 is *not* recommended.

0	Given upper and lower bounds on each variable, the greatest and least contribution to the row activity of each variable is computed. If these are within the limits set by the upper and lower bounds on the row activity, then the row is redundant and can be discarded. Try to tighten the bounds on any of the variables it can. For example, if all coefficients in a constraint are positive and all variables have zero lower bounds, then the row's smallest contribution is zero. If the rhs value of this constraint is zero, then if the constraint type is $=$ or $\leq$, all the variables in that constraint can be fixed to zero. These variables and the constraint can be removed. If the constraint type is $\geq$, the constraint is redundant. If the rhs is negative and the constraint is $\leq$, the problem is infeasible. If just one variable in a row is not fixed, use the row to impose an implicit upper or lower bound on the variable and then eliminate the row. The preprocessor also tries to tighten the bounds on constraint right hand sides. PRSLTYPE=0 is the default.
1	When there are exactly two unfixed variables with coefficients in an equality constraint, solve for one in terms of the other. The problem will have one less variable. The new matrix will have at least two fewer coefficients and one less constraint. In other constraints where both variables appear, two coefs are combined into one. type=0 reductions are also done.
2	It may be possible to determine that an equality constraint is not constraining a variable. That is, if all variables are nonnegative, then x - SUM yi = 0 does not constrain x, since it must be nonnegative if all the yi's are nonnegative. In this case, eliminate x by subtracting this equation from all others containing x. This is useful when the only other entry for x is in the objective function. Perform this reduction if there is at most one other nonobjective coefficient. type=0 reductions are also done.
3	All possible reductions are performed.

Preprocessing is iterative. As variables are fixed and eliminated, and constraints are found to be redundant and they too are eliminated, and as variable bounds s and constraint right hand sides are tightened, the LP to be optimized is modified to reflect these changes. Another iteration of preprocessing of the modified LP may reveal more variables and constraints that can be eliminated.

RHS Statement

RHS *variable* **;**

The RHS variable list is used when the dense format of the CONDATA= data set is used. The values of the SAS variable specified in the RHS list are constraint right-hand-side values. If the RHS list is not specified, the CONDATA= data set is searched and a SAS variable with the name _RHS_ is used. If there is no RHS list and no

SAS variable named _RHS_, all constraints are assumed to have zero right-hand-side values. The RHS list variable must have numeric values.

ROW Statement

ROW *variables* **;**

The ROW list is used when either the sparse or dense format of side constraints is being used. SAS variables in the ROW list have values that are constraint or special row names. The SAS variables in the ROW list must have character values.

If the dense data format is used, there must be only one SAS variable in this list. In this case, if a ROW list is not specified, the CONDATA= data set is searched and the SAS variable with the name _ROW_ or _CON_ is used.

If the sparse data format is used and the ROW statement is not specified, the CONDATA= data set is searched and SAS variables with names beginning with _ROW or _CON are used. The number of SAS variables in the ROW list must not be less than the number of SAS variables in the COEF list. The ith ROW list variable is paired with the ith COEF list variable. If the number of ROW list variables is greater than the number of COEF list variables, the last ROW list variables have no COEF partner. These ROW list variables that have no corresponding COEF list variable are used in observations that have a TYPE list variable value. All ROW list variable values are tagged as having the type indicated. If there is no TYPE list variable, all ROW list variable values are constraint names.

RUN Statement

RUN;

The RUN statement causes optimization to be started or resumed. The RUN statement has no options. If PROC NETFLOW is called and is not terminated because of an error or a QUIT statement, and you have not used a RUN statement, a RUN statement is assumed implicitly as the last statement of PROC NETFLOW. Therefore, PROC NETFLOW always performs optimization and saves the obtained (optimal) solution in the current output data sets.

SAVE Statement

SAVE *options* **;**

The options available with the SAVE statement of PROC NETFLOW are summarized by purpose in Table 4.21.

Table 4.21. Functional Summary, SAVE statement: PROC NETFLOW

Description	Statement	Option
Output Data Set Options: SAVE		
unconstrained solution data set	SAVE	ARCOUT=
unconstrained solution data set	SAVE	NODEOUT=
constrained solution data set	SAVE	CONOUT=
constrained solution data set	SAVE	DUALOUT=

The SAVE statement can be used to specify output data sets and create observations in these data sets. Use the SAVE statement if no optimization is to be performed before these output data sets are created.

The SAVE statement must be used to save solutions in data sets if there is no more optimization to do. If more optimization is to be performed, after which you want to save the solution, then do one of the following:

- Submit a RUN statement followed by a SAVE statement.
- Use the PROC NETFLOW or RESET statement to specify current output data sets. After optimization, output data sets are created and observations are automatically sent to the current output data sets.

Consider the following example:

```
proc netflow options; lists;
   reset maxit1=10 maxit2=25
         arcout=arcout0   nodeout=nodeout0
         conout=conout0   dualout=dualout0;
   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 nodeout0 are created.              */
   /* Arcout0 is not created as arcout=arcout2 over- */
   /* rides the arcout=arcout0 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.                 */
/* Conout0 is created.                             */
/* No dualout is created as the last NETFLOW or    */
/* reset statements 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 preceeding example are all distinct, but this need not be the case. The only exception to this is that the ARCOUT= data set and the NODEOUT= data set (or the CONOUT= data set and the DUALOUT= data set) that are being created at the same time must be distinct. Use the SHOW DATA SETS; statement to examine what data sets are current and that have already been created and when.

The following options can appear in the SAVE statement:

ARCOUT= *SAS-data-set***;** , or
AOUT= *SAS-data-set***;**
NODEOUT= *SAS-data-set***;** , or
NOUT= *SAS-data-set***;**
CONOUT= *SAS-data-set***;** , or
COUT= *SAS-data-set***;**
DUALOUT= *SAS-data-set***;** , or
DOUT= *SAS-data-set***;**

SHOW Statement

SHOW *options / qualifiers* **;**

The options available with the SHOW statement of PROC NETFLOW are summarized by purpose in Table 4.22.

Table 4.22. Functional Summary, SHOW statement: PROC NETFLOW

Description	Statement	Option
SHOW statement options		
show problem, optimization status	SHOW	STATUS
show network model parameters	SHOW	NETSTMT
show data sets that have, will be created	SHOW	DATA SETS
show options that pause optimization	SHOW	PAUSE

Description	Statement	Option
show Simplex algorithm options	SHOW	SIMPLEX
show pricing strategy options	SHOW	PRICING
show miscellaneous options	SHOW	MISC
SHOW statement qualifiers		
only if relevant	SHOW	RELEVANT
only stage 1 options when doing stage 1. only stage 2 options when doing stage 2	SHOW	STAGE

The SHOW statement enables you to examine the status of the problem and values of the RESET statement options. All output of the SHOW statement appears on the SAS log. The amount of information displayed when a SHOW statement is processed can be limited if some of the options of the SHOW statement are specified. These options indicate whether the problem status or a specific category of the RESET options is of interest. If no options are specified, the problem status and information on all RESET statement options in every category is displayed. The amount of displayed information can be limited further by following any SHOW statement options with a slash (/) and one or both qualifiers, RELEVANT and STAGE.

The following options can appear in the SHOW statement.

STATUS

produces one of the following optimization status reports, whichever is applicable. The warning messages are issued only if the network or entire problem is infeasible.

```
NOTE: Optimization Status.
      Optimization has not started yet.

NOTE: Optimization Status.
      Optimizing network (ignoring any side constraints).
      Number of iterations=17
      Of these, 3 were degenerate
WARNING: This optimization has detected that the network is
         infeasible.

NOTE: Optimization Status.
      Found network optimum (ignoring side constraints)
      Number of iterations=23
      Of these, 8 were degenerate

NOTE: Optimization Status.
      Optimizing side constrained network.
      Number of iterations=27
      Of these, 9 were degenerate
WARNING: This optimization has detected that the problem is
         infeasible.
```

```
NOTE: Optimization Status.
      Found side constrained network optimum
      Number of iterations=6
      Of these, 0 were degenerate
```

DATA SETS

produces a report on output data sets.

```
NOTE: Current output SAS data sets
      No output data sets have been specified

NOTE: Current output SAS data sets
      ARCOUT=libname.memname
      NODEOUT=libname.memname
      CONOUT=libname.memname
      DUALOUT=libname.memname

NOTE: Other SAS data sets specified in previous ARCOUT=, NODEOUT=,
      CONOUT=, or DUALOUT=.
      libname.memname
              .
              .
              .

NOTE: Current output SAS data sets (SHOW DATA SETS)
      libname.memname
              .
              .
              .
NOTE: SAS data sets specified as ARCOUT= NODEOUT= CONOUT= or
      DUALOUT= data sets in previous PROC NETFLOW, SET, RESET
      and SAVE statements.
      The number following the data set specification was the
      iteration number when observations were placed into the
      data set.
      libname.memname      iteration_number
              .                    .
              .                    .
              .                    .
```

PAUSE

produces a report on the current settings of options used to make optimization pause.

```
NOTE: Options and parameters that stop optimization for reasons
      other than infeasibility or optimality (SHOW PAUSE)
      FEASIBLEPAUSE1=FALSE
      ENDPAUSE1=FALSE
      PAUSE1=999999
      MAXIT1=1000
      FEASIBLEPAUSE2=FALSE
      PAUSE2=999999
      MAXIT2=999999
```

SIMPLEX

produces the following:

```
NOTE: Options and parameters that control the Primal Simplex
      Network algorithm (excluding those that affect the
      pricing strategies) (SHOW SIMPLEX)
      LRATIO1=FALSE
      BIGM1=NOTWOPHASE1=TRUE, TWOPHASE1=NOBIGM1=FALSE
      CYCLEMULT1=0.15
      PERTURB1=FALSE
      MINBLOCK1=2
      INTFIRST=TRUE
      LRATIO2=FALSE
      BIGM2=NOTWOPHASE2=TRUE, TWOPHASE2=NOBIGM2=FALSE
      REFACTFREQ=50
      U=0.1
      MAXLUUPDATES=6
      MAXL=40
```

PRICING

produces the following:

```
NOTE: Options and parameters that control the Primal Simplex
      Network algorithm pricing strategies (SHOW PRICING)
      PRICETYPE1=Q
      P1SCAN=FIRST
      P1NPARTIAL=10
      Q1FILLSCAN=FIRST
      QSIZE1=24
      REFRESHQ1=0.75
      REDUCEQSIZE1=1
      Q1FILLNPARTIAL=10
      PRICETYPE2=Q
      P2SCAN=FIRST
      P2NPARTIAL=10
      DUALFREQ=4
      Q2FILLSCAN=FIRST
      QSIZE2=24
      REFRESHQ2=0.75
      REDUCEQSIZE2=1
      Q2FILLNPARTIAL=10
```

MISC

produces the following:

```
NOTE: Miscellaneous options and parameters (SHOW MISC)
      VERBOSE=12
      ZTOL1=TRUE
      ZERO1=1E-6
      FUTURE1=FALSE
      ZTOL2=TRUE
      ZERO2=1E-6
      FUTURE2=FALSE
```

Following a slash (/), the qualifiers below can appear in any SHOW statement.

RELEVANT

indicates that you want information only on relevant options of the RESET statement. The following will *not* be displayed if / RELEVANT is specified:

- information on noncurrent data sets
- the options that control the reasons why stage 1 optimization should be halted and the options that control the Simplex algorithm during stage 1 optimization, if the unconstrained optimum has been reached or constrained optimization has been performed
- if P1SCAN=BEST or P1SCAN=FIRST, the P1NPARTIAL option is irrelevant
- if PRICETYPE1=BLAND or PRICETYPE1=NOQ, the options QSIZE1=, Q1FILLSCAN=, REFRESHQ1=, and REDUCEQSIZE1= are irrelevant
- if Q1FILLSCAN=BEST or Q1FILLSCAN=FIRST, the Q1FILLNPARTIAL= option is irrelevant
- the options that control the reasons stage 2 optimization should be halted, the options that control the Simplex algorithm during stage 2 optimization, if the constrained optimum has been reached
- If P2SCAN=BEST or P2SCAN=FIRST, the P2NPARTIAL= option is irrelevant
- if PRICETYPE2=BLAND or PRICETYPE2=NOQ, the options QSIZE2=, Q2FILLSCAN=, REFRESHQ2=, and REDUCEQSIZE2= are irrelevant.
- If Q2FILLSCAN=BEST or Q2FILLSCAN=FIRST, the Q2FILLNPARTIAL= option is irrelevant.

STAGE

indicates that you want to examine only the options that affect the optimization that is performed if a RUN statement is executed next. Before any optimization has been done, only stage 2 options are displayed if the problem has side constraints and the SCRATCH option is used, or if the CONOPT statement is specified. Otherwise, stage 1 options are displayed. If still optimizing neglecting constraints, only stage 1 options will be displayed. If the unconstrained optimum has been reached and optimization that considers constraints has not been performed, stage 1 options are displayed. If the problem has constraints, stage 2 options are displayed. If any optimization that considers constraints has been performed, only stage 2 options are displayed.

SUPDEM Statement

SUPDEM *variable* **;**

The SAS variable in this list, that must be present in the NODEDATA= data set, contains supply and demand information for the nodes in the NODE list. A positive SUPDEM list variable value s ($s > 0$) denotes that the node named in the NODE list variable can supply s units of flow. A negative SUPDEM list variable value $-d$ ($d > 0$) means that this node demands d units of flow. If a SAS variable is not explicitly specified, a SAS variable with the name _SUPDEM_ or _SD_ in NODEDATA=data set is used as the SUPDEM variable. If a node is a transshipment node (neither a supply nor a demand node), an observation associated with this node need not be present in the NODEDATA= data set. If present, the SUPDEM list variable value must be zero or a missing value.

SUPPLY Statement

SUPPLY *variable* **;**

The SUPPLY statement identifies the SAS variable in the ARCDATA= data set that contains the supply at the node named in that observation's TAILNODE list variable. If a tail node does not supply flow, use zero or a missing value for the observation's SUPPLY list variable value. If a tail node has supply capability, a missing value indicates that the supply quantity is given in another observation. It is not necessary to have a SUPPLY statement if the name of this SAS variable is _SUPPLY_.

TAILNODE Statement

TAILNODE *variable* **;**
TAIL *variable* **;**

FROMNODE *variable* **;**

FROM *variable* **;**

The TAILNODE statement specifies the SAS variable that must be present in the ARCDATA= data set that has as values the names of tail nodes of arcs. The TAILNODE variable must have character values. It is not necessary to have a TAILNODE statement if the name of the SAS variable is _TAIL_ or _FROM_. If the TAILNODE list variable value is missing, it is assumed that the observation of ARCDATA= data set contains information concerning a nonarc variable.

TYPE Statement

TYPE *variable* **;**
CONTYPE *variable* **;**

The TYPE list, which is optional, names the variable that has as values keywords that indicate either the constraint type for each constraint or the type of special rows in the CONDATA= data set. The values of the TYPE list variable also indicate, in each observation of the CONDATA= data set, how values of the VAR or COEF list variables are to be interpreted and how the type of each constraint or special row name is determined. If the TYPE list is not specified, the CONDATA= data set is searched and a SAS variable with the name _TYPE_ is used. Valid keywords for the TYPE variable are given below. If there is no TYPE statement and no other method is used to furnish type information (see the DEFCONTYPE= option), all constraints are assumed to be of the type “less than or equal to” and no special rows are used. The TYPE list variable must have character values and can be used when the data in the CONDATA= data set is in either the sparse or dense format. If the TYPE list variable value has a * as its first character, the observation is ignored because it is a comment observation.

TYPE List Variable Values

The following are valid TYPE list variable values. The letters in boldface denote the characters that PROC NETFLOW uses to determine what type the value suggests. You need to have at least these characters. Below, the minimal TYPE list variable values have additional characters to aid you in remembering these values.

The valid TYPE list variable values are

<	less than or equal to ($\leq$)
=	equal to (=)
>	greater than or equal to ($\geq$)
CAPAC	capacity
COST	cost
EQ	equal to
FREE	free row (used only for Linear Programs solved by Interior Point)
GE	greater than or equal to
LE	less than or equal to
LOWERBD	lower flow or value bound
LOW*blank*	lower flow or value bound
MAXIMIZE	maximize (opposite of cost)
MINIMIZE	minimize (same as cost)
OBJECTIVE	objective function (same as cost)
RHS	rhs of constraint

TYPE	type of constraint
UPPCOST	reserved for future use
UNREST	unrestricted variable (used only for Linear Programs solved by Interior Point)
UPPER	upper value bound or capacity; second letter must not be N

The valid TYPE list variable values in function order are

- **LE** less than or equal to ($\leq$)
- **EQ** Equal to (=)
- **GE** Greater than or equal to ($\geq$)
- **CO**ST
  MINIMIZE
  MAXIMIZE
  OBJECTIVE
  cost or objective function coefficient
- **CA**PAC
  UPPER capacity or upper value bound
- **LOW**ERBD
  LOW*blank* Lower flow or value bound
- **R**HS rhs of constraint
- **T**YPE type of constraint

A TYPE list variable value that has the first character $*$ causes the observation is be treated as a comment. If the first character is a negative sign, then $\leq$, zero =, a positive number $\geq$.

VAR Statement

VAR *variables* **;**

The VAR variable list is used when the dense data format is used. The names of these SAS variables are also names of the arc and nonarc variables that have data in the CONDATA= data set. If no explicit VAR list is specified, all numeric variables not on other lists are put onto the VAR list. The VAR list variables must have numeric values. The values of the VAR list variables in some observations can be interpreted differently than in other observations. The values can be coefficients in the side constraints, costs and objective function coefficients, or bound data. How these numeric values are interpreted depends on the value of each observation's TYPE or ROW list variable value. If there are no TYPE list variables, the VAR list variable values are all assumed to be side constraint coefficients.

Details

Input Data Sets

PROC NETFLOW is designed so that there are as few rules as possible that you must obey when inputting a problem's data. Raw data are acceptable. This should cut the amount of processing required to groom the data before it is input to PROC NETFLOW. Data formats are so flexible that, due to space restrictions, all possible forms for a problem's data are not shown here. Try any reasonable form for your problem's data; it should be acceptable. PROC NETFLOW will outline its objections.

There are several ways to supply the same piece of data. You do not have to restrict yourself to using any particular one. If you use several ways, PROC NETFLOW checks that the data are consistent each time the data are encountered. After all input data sets have been read, data are merged so that the problem is described completely. The order of the observations is not important in any of the input data sets.

ARCDATA= Data Set

See the "Getting Started" section on page 196 and the "Introductory Example" section on page 197 for a description of this input data set.

Note: information for an arc or nonarc variable can be specified in more than one observation. For example, consider an arc directed from node A toward node B that has a cost of 50, capacity of 100, and lower flow bound of 10 flow units. Some possible observations in the ARCDATA= data set may be

```
_TAIL_  _HEAD_  _COST_  _CAPAC_  _LO_
  A       B      50        .       .
  A       B       .       100      .
  A       B       .        .      10
  A       B      50       100      .
  A       B       .       100     10
  A       B      50        .      10
  A       B      50       100     10
```

Similarly, for a nonarc variable with upperbd=100, lowerbd=10, and objective function coefficient=50, the _TAIL_ and _HEAD_ values are missing.

CONDATA= Data Set

Regardless of whether the data in the CONDATA= data set is in the sparse or dense format, you will receive a warning if PROC NETFLOW finds a constraint row that has no coefficients. You will also be warned if any nonarc variable has no constraint coefficients.

Dense Input Format

If the dense format is used, most SAS variables in the CONDATA= data set belong to the VAR list and have names of arc and nonarc variables. These names can be values of the NAME list SAS variables in the ARCDATA= data set, or names of nonarc variables, or names in the form *tail_head* , or any combination of these three forms. Names in the form *tail_head* are default arc names, and if you use them, you must specify node names in the ARCDATA= data set (values of the TAILNODE and HEADNODE list SAS variables) using no lowercase letters.

There can be three other variables in the CONDATA= data set, belonging, respectively, to the ROW, TYPE, and RHS lists. The CONDATA= data set of the oil industry example in the “Introductory Example” section on page 197 uses the dense data format.

Consider the SAS code that creates a dense format CONDATA= data set that has data for three constraints. This data set was used in the “Introductory Example” section on page 197.

```
data cond1;
   input m_e_ref1 m_e_ref2 thruput1 r1_gas thruput2 r2_gas
      _type_ $ _rhs_;
   datalines;
-2  .  1 .  . . >= -15
 . -2  . .  1 . GE -15
 .  . -3 4  . . EQ   0
 .  .  . . -3 4  =   0
;
```

You can use nonconstraint type values to furnish data on costs, capacities, lower flow bounds (and, if there are nonarc variables, objective function coefficients and upper and lower bounds). You need not have such (or as much) data in the ARCDATA= data set. The first three observations in the following data set are examples of observations that provide cost, capacity and lower bound data.

```
data cond1b;
   input m_e_ref1 m_e_ref2 thruput1 r1_gas thruput2 r2_gas
         _type_ $ _rhs_;
   datalines;
63 81 200   . 220   . cost   .
95 80 175 140 100 100 capac  .
20 10  50   .  35   . lo     .
-2  .   1   .   .   . >=   -15
 . -2   .   .   1   . GE   -15
 .  .  -3   4   .   . EQ     0
 .  .   .   .  -3   4  =     0
;
```

If a ROW list variable is used, the data for a constraint can be spread over more than 1 observation. To illustrate, the data for the first constraint, (which is called con1), and the cost and capacity data (in special rows called costrow and caprow, respectively) will be spread over more than one observation in the following data set.

```
data cond1b;
   input _row_ $
         m_e_ref1 m_e_ref2 thruput1 r1_gas thruput2 r2_gas
         _type_ $ _rhs_;
   datalines;
costrow 63  .   .   .   .  . .      .
```

```
costrow  . 81 200   .   .   . cost   .
.        .  .   .   . 220   . cost   .
caprow   .  .   .   .   .   . capac  .
caprow  95  . 175   . 100 100 .      .
caprow   . 80 175 140   .   . .      .
lorow   20 10  50   .  35   . lo     .
con1    -2  .   1   .   .   . .      .
con1     .  .   .   .   .   . >=   -15
con2     . -2   .   .   1   . GE   -15
con3     .  .  -3   4   .   . EQ     0
con4     .  .   .   .  -3   4  =     0
;
```

Using both ROW and TYPE lists, you can use special row names. Examples of these are "costrow" and "caprow" in the last data set. It should be restated that in any of the input data sets of PROC NETFLOW, the order of the observation does not matter. However, the CONDATA= data set can be read more quickly if PROC NETFLOW knows what type of constraint or special row a ROW list variable value is. For example, when the first observation is read, PROC NETFLOW does not know whether costrow is a constraint or special row and how to interpret the value 63 for the arc with the name m_e_ref1. When PROC NETFLOW reads the second observation, it learns that costrow has cost type and that the values 81 and 200 are costs. When the entire CONDATA= data set has been read, PROC NETFLOW knows the type of all special rows and constraints. Data that PROC NETFLOW had to set aside (such as the first observation 63 value and the costrow ROW list variable value, which at the time had unknown type, but is then known to be a cost special row) is reprocessed. During this second pass, if a ROW list variable value has unassigned constraint or special row type, it is treated as a constraint with DEFCONTYPE= (or DEFCONTYPE= default) type. Associated VAR list valiable values as coefficients of that constraint.

Sparse Input Format

The side constraints usually become sparse as the problem size increases. When the sparse data format of the CONDATA= data set is used, only nonzero constraint coefficients must be specified. Remember to specify the SPARSECONDATA option in the PROC NETFLOW statement. With the sparse method of specifying constraint information, the names of arc and nonarc variables do not have to be valid SAS variable names.

A sparse format CONDATA= data set for the oil industry example in the "Introductory Example" section on page 197 is displayed below.

```
title 'Setting Up Condata = Cond2 for PROC NETFLOW';
data cond2;
   input _column_ $ _row1 $ _coef1 _row2 $ _coef2 ;
   datalines;
m_e_ref1  con1  -2     .   .
m_e_ref2  con2  -2     .   .
thruput1  con1   1  con3  -3
r1_gas       .   .  con3   4
thruput2  con2   1  con4  -3
r2_gas       .   .  con4   4
```

```
_type_    con1   1  con2   1
_type_    con3   0  con4   0
_rhs_     con1 -15  con2 -15
;
```

Recall that the COLUMN list variable values "_type_" and "_rhs_" are the default values of the TYPEOBS= and RHSOBS= options. Also, the default rhs value of constraints (con3 and con4) is zero. The third to last observation has the value "_type_" for the COLUMN list variable. The _ROW1 variable value is con1, and the _COEF1_ variable has the value 1. This indicates that the constraint con1 is *greater than* or equal to type (because the value 1 is *greater than* zero). Similarly, the data in the second to last observation's _ROW2 and _COEF2 variables indicate that con2 is an *equality* constraint (0 *equals* zero).

An alternative, using a TYPE list variable is

```
title 'Setting Up Condata = Cond3 for PROC NETFLOW';
data cond3;
   input _column_ $ _row1 $ _coef1 _row2 $ _coef2 _type_ $ ;
   datalines;
m_e_ref1  con1  -2     .   . >=
m_e_ref2  con2  -2     .   . .
thruput1  con1   1  con3  -3 .
r1_gas       .   .  con3   4 .
thruput2  con2   1  con4  -3 .
r2_gas       .   .  con4   4 .
.         con3   .  con4   . eq
.         con1 -15  con2 -15 ge
;
```

If the COLUMN list variable is missing in a particular observation (the last 2 observations in the data set cond3, for instance), the constraints named in the ROW list variables all have the constraint type indicated by the value in the TYPE list variable. It is for this type of observation that you are allowed more ROW list variables than COEF list variables. If corresponding COEF list variables are not missing (for example, the last observation in the data set cond3), these values are the rhs values of those constraints. Therefore, you can specify both constraint type and rhs in the same observation.

As in the previous CONDATA= data set, if the COLUMN list variable is an arc or nonarc variable, the COEF list variable values are coefficient values for that arc or nonarc variable in the constraints indicated in the corresponding ROW list variables. If in this same observation, the TYPE list variable contains a constraint type, all constraints named in the ROW list variables in that observation have this constraint type (for example, the first observation in the data set cond3). Therefore, you can specify both constraint type and coefficient information in the same observation.

Also note that DEFCONTYPE=EQ could have been specified, saving you from having to include in the data that CON3 and CON4 are of this type.

In the oil industry example, arc costs, capacities, and lower flow bounds are presented in the ARCDATA= data set. Alternatively, you could have used the following input data sets:

```
title3 'Setting Up Arcdata = Arcd2 for PROC NETFLOW';
data arcd2;
   input  _from_&$11. _to_&$15. ;
   datalines;
middle east  refinery 1
middle east  refinery 2
u.s.a.       refinery 1
u.s.a.       refinery 2
refinery 1   r1
refinery 2   r2
r1           ref1 gas
r1           ref1 diesel
r2           ref2 gas
r2           ref2 diesel
ref1 gas     servstn1 gas
ref1 gas     servstn2 gas
ref1 diesel  servstn1 diesel
ref1 diesel  servstn2 diesel
ref2 gas     servstn1 gas
ref2 gas     servstn2 gas
ref2 diesel  servstn1 diesel
ref2 diesel  servstn2 diesel
;

title 'Setting Up Condata = Cond4 for PROC NETFLOW';
data cond4;
   input _column_&$27. _row1 $ _coef1 _row2 $ _coef2 _type_ $ ;
   datalines;
.                              con1 -15   con2 -15    ge
.                           costrow   .      .   .  cost
.                                 .   . caprow   . capac
middle east_refinery 1         con1  -2      .   .     .
middle east_refinery 2         con2  -2      .   .     .
refinery 1_r1                  con1   1   con3  -3     .
r1_ref1 gas                       .   .   con3   4     =
refinery 2_r2                  con2   1   con4  -3     .
r2_ref2 gas                       .   .   con4   4    eq
middle east_refinery 1      costrow  63 caprow  95     .
middle east_refinery 2      costrow  81 caprow  80     .
u.s.a._refinery 1           costrow  55      .   .     .
u.s.a._refinery 2           costrow  49      .   .     .
refinery 1_r1               costrow 200 caprow 175     .
refinery 2_r2               costrow 220 caprow 100     .
r1_ref1 gas                       .   . caprow 140     .
r1_ref1 diesel                    .   . caprow  75     .
r2_ref2 gas                       .   . caprow 100     .
r2_ref2 diesel                    .   . caprow  75     .
ref1 gas_servstn1 gas       costrow  15 caprow  70     .
ref1 gas_servstn2 gas       costrow  22 caprow  60     .
ref1 diesel_servstn1 diesel costrow  18      .   .     .
```

```
ref1 diesel_servstn2 diesel  costrow  17       .   .     .
ref2 gas_servstn1 gas        costrow  17 caprow  35     .
ref2 gas_servstn2 gas        costrow  31       .   .     .
ref2 diesel_servstn1 diesel  costrow  36       .   .     .
ref2 diesel_servstn2 diesel  costrow  23       .   .     .
middle east_refinery 1             .  20       .   .    lo
middle east_refinery 2             .  10       .   .    lo
refinery 1_r1                      .  50       .   .    lo
refinery 2_r2                      .  35       .   .    lo
ref2 gas_servstn1 gas              .   5       .   .    lo
;
```

The first observation in the cond4 data set defines con1 and con2 as *greater than or equal to* ($\geq$) constraints that both (by coincidence) have rhs values of -15. The second observation defines the special row costrow as a cost row. When costrow is a ROW list variable value, the associated COEF list variable value is interpreted as a cost or objective function coefficient. PROC NETFLOW has to do less work if constraint names and special rows are defined in observations near the top of a data set, but this is not a strict requirement. The fourth to ninth observations contain constraint coefficient data. Observations 7 and 9 have TYPE list variable values that indicate that constraints con3 and con4 are equality constraints. The last five observations contain lower flow bound data. Observations that have an arc or nonarc variable name in the COLUMN list variable, a nonconstraint type TYPE list variable value, and a value in (one of) the COEF list variables are valid.

The following data set is equivalent to the cond4 data set.

```
title 'Setting Up Condata = Cond5 for PROC NETFLOW';
data cond5;
   input _column_&$27. _row1 $ _coef1 _row2 $ _coef2 _type_ $ ;
   datalines;
middle east_refinery 1          con1  -2 costrow  63     .
middle east_refinery 2          con2  -2   lorow  10     .
refinery 1_r1                      .   .    con3  -3     =
r1_ref1 gas                   caprow 140    con3   4     .
refinery 2_r2                   con2   1    con4  -3     .
r2_ref2 gas                        .   .    con4   4    eq
.                               CON1 -15    CON2 -15    GE
ref2 diesel_servstn1 diesel        .  36 costrow   .  cost
.                                  .   .  caprow   . capac
.                              lorow   .       .   .    lo
middle east_refinery 1        caprow  95   lorow  20     .
middle east_refinery 2        caprow  80 costrow  81     .
u.s.a._refinery 1                  .   .       .  55  cost
u.s.a._refinery 2            costrow  49       .   .     .
refinery 1_r1                   con1   1  caprow 175     .
refinery 1_r1                  lorow  50 costrow 200     .
refinery 2_r2                costrow 220  caprow 100     .
refinery 2_r2                      .  35       .   .    lo
r1_ref1 diesel               caprow2  75       .   . capac
r2_ref2 gas                        .   .  caprow 100     .
r2_ref2 diesel               caprow2  75       .   .     .
```

```
ref1 gas_servstn1 gas        costrow  15  caprow  70     .
ref1 gas_servstn2 gas        caprow2  60 costrow  22     .
ref1 diesel_servstn1 diesel        .   . costrow  18     .
ref1 diesel_servstn2 diesel  costrow  17       .   .     .
ref2 gas_servstn1 gas        costrow  17   lorow   5     .
ref2 gas_servstn1 gas              .   . caprow2  35     .
ref2 gas_servstn2 gas              .  31       .   .  cost
ref2 diesel_servstn2 diesel        .   . costrow  23     .
;
```

If you have data for a linear programming program that has an embedded network, the steps required to change that data into a form that is acceptable by PROC NETFLOW are

1. Identify the nodal flow conservation constraints. The coefficient matrix of these constraints (a submatrix of the LP's constraint coefficient matrix) has only two nonzero elements in each column, -1 and 1.
2. Assign a node to each nodal flow conservation constraint.
3. The rhs values of conservation constraints are the corresponding node's supplies and demands. Use this information to create a NODEDATA= data set.
4. Assign an arc to each column of the flow conservation constraint coefficient matrix. The arc is directed from the node associated with the row that has the 1 element in it and directed toward to the node associated with the row that has the -1 element in it. Set up an ARCDATA= data set that has two SAS variables. This data set could resemble ARCDATA=arcd2. These will eventually be the TAILNODE and HEADNODE list variables when PROC NETFLOW is used. Each observation consists of the tail and head node of each arc.
5. Remove from the data of the linear program all data concerning the nodal flow conservation constraints.
6. Put the remaining data into a CONDATA= data set. This data set will probably resemble CONDATA=cond4 or CONDATA=COND5.

The Sparse Format Summary

The following list illustrates possible CONDATA= data set observation sparse formats. a1, b1, b2, b3 and c1 have as a _COLUMN_ variable value either the name of an arc (possibly in the form *tail_head*) or the name of a nonarc variable.

- If there is no TYPE list variable in the CONDATA= data set, the problem must be constrained and there is no nonconstraint data in the CONDATA= data set.

```
       _COLUMN_       _ROWx_      _COEFx_       _ROWy_
                                             (no _COEFy_)
                                             (may not be
                                              in CONDATA)
   a1  variable      constraint   lhs coef    +------------+
   a2  _TYPE_ or     constraint    -1 0 1     |            |
        TYPEOBS=                              |            |
   a3  _RHS_  or     constraint   rhs value   | constraint |
        RHSOBS= or                            |     or     |
        missing                               |  missing   |
   a4  _TYPE_ or     constraint    missing    |            |
        TYPEOBS=                              |            |
   a5  _RHS_  or     constraint    missing    |            |
        RHSOBS= or                            +------------+
        missing
```

  Observations of the form a4 and a5 serve no useful purpose but are still allowed to make problem generation easier.

- If there are no ROW list variables in the data set, the problem has no constraints and the information is nonconstraint data. There must be a TYPE list variable and only one COEF list variable in this case. The COLUMN list variable has as values the names of arcs or nonarc variables and must not have missing values or special row names as values.

```
         _COLUMN_         _TYPE_          _COEFx_

     b1  variable         UPPERBD         capacity
     b2  variable         LOWERBD         lower flow
     b3  variable         COST            cost
```

- Using a TYPE list variable for constraint data implies the following:

```
       _COLUMN_    _TYPE_  _ROWx_    _COEFx_     _ROWy_
                                              (no _COEFy_)
                                              (may not be
                                               in CONDATA)
   c1  variable    missing +-----+  lhs coef  +------------+
   c2  _TYPE_ or   missing |  c  |   -1 0 1   |            |
        TYPEOBS=           |  o  |            |            |
   c3  _RHS_ or    missing |  n  |  rhs value | constraint |
       missing             |  s  |            |     or     |
       or RHSOBS=          |  t  |            |  missing   |
   c4  variable   con type |  r  |  lhs coef  |            |
   c5  _RHS_ or   con type |  a  |  rhs value |            |
       missing             |  i  |            |            |
       or RHSOBS=          |  n  |            |            |
   c6  missing      TYPE   |  t  |   -1 0 1   |            |
   c7  missing      RHS    +-----+  rhs value +------------+
```

  If the observation is in form c4 or c5, and the _COEFx_ values are missing, the constraint is assigned the type data specified in the _TYPE_ variable.

- Using a TYPE list variable for arc and nonarc variable data implies the following:

```
        _COLUMN_   _TYPE_      _ROWx_     _COEFx_    _ROWy_
                                                   (no _COEFy_)
                                                   (may not be
                                                    in CONDATA)
                 +---------+ +---------+            +---------+
     d1 variable | UPPERBD | | missing | capacity   | missing |
     d2 variable | LOWERBD | |   or    | lowerflow  |   or    |
     d3 variable | COST    | | special | cost       | special |
                 |         | |  row    |            |  row    |
                 |         | |  name   |            |  name   |
                 |         | +---------+            |         |
     d4 missing  |         | | special |            |         |
                 |         | |  row    |            |         |
                 +---------+ |  name   |            +---------+
     d5 variable   missing   |         | value that   missing
                             |         |is interpreted
                             |         |according to
                             +---------+   _ROWx_
```

The observation with form d1 to d5 can have ROW list variable values. Observation d4 must have ROW list variable values. The ROW value is put into the ROW name tree so that when dealing with observation d4 or d5, the COEF list variable value is interpreted according to the type of ROW list variable value. For example, the following three observations define the _ROWx_ variable values up_row, lo_row and co_row as being a upper value bound row, lower value bound row, and cost row, respectively.

```
_COLUMN_          _TYPE_         _ROWx_          _COEFx_

   .              UPPERBD        up_row             .
variable_a        LOWERBD        lo_row         lower flow
variable_b        COST           co_row         cost
```

PROC NETFLOW is now able to correctly interpret the following observation:

```
_COLUMN_ _TYPE_ _ROW1_ _COEF1_ _ROW2_ _COEF2_ _ROW3_ _COEF3_

 var_c     .    up_row  upval  lo_row  loval  co_row   cost
```

If the TYPE list variable value is a constraint type and the value of the COLUMN list variable equals the value of the TYPEOBS= option or the default value _TYPE_, the TYPE list variable value is ignored.

NODEDATA= Data Set

See the “Getting Started” section on page 196 and the “Introductory Example” section on page 197 for a description of this input data set.

Output Data Sets

The procedure determines the flow that should pass through each arc as well as the value assigned to each nonarc variable. The goal is that the minimum flow bounds, capacities, lower and upper value bounds, and side constraints are not violated. This goal is reached when total cost incurred by such a flow pattern and value assignment is feasible and optimal. The solution found must also conserve flow at each node.

The ARCOUT= data set contains a solution obtained when performing optimization that does not consider any constraints. The NODEOUT= data set contains nodal dual variable information for this type of solution. You can choose to have PROC NETFLOW create the ARCOUT= data set and the NODEOUT= data set and save the optimum of the network or the nodal dual variable values before any optimization that considers the side constraints is performed.

If there are side constraints, the CONOUT= data set can be produced and contains a solution obtained after performing optimization that considers constraints. The DUALOUT= data set contains dual variable information for nodes and side constraints from the solution obtained after optimization that considers the constraints. The CONOUT= data set and DUALOUT= data set can be used to save the constrained optimal solution.

ARCOUT= and CONOUT= Data Sets

The ARCOUT= and CONOUT= data sets contain the same variables. These variables and their possible values in an observation are

FROM a tail node of an arc. This is a missing value if an observation has information about a nonarc variable

_TO_ a head node of an arc. This is a missing value if an observation has information about a nonarc variable

COST the cost of an arc or the objective function coefficient of a nonarc variable

CAPAC the capacity of an arc or upper value bound of a nonarc variable

_LO_ the lower flow bound of an arc or lower value bound of a nonarc variable

NAME a name of an arc or nonarc variable

SUPPLY the supply of the tail node of the arc in the observation. This is a missing value if an observation has information about a nonarc variable

DEMAND the demand of the head node of the arc in the observation. This is a missing value if an observation has information about a nonarc variable

FLOW the flow through the arc or value of the nonarc variable

FCOST flow cost, the product of _COST_ and _FLOW_

RCOST the reduced cost of the arc or nonarc variable

ANUMB the number of the arc (positive) or nonarc variable (nonpositive). Used for warm starting PROC NETFLOW

TNUMB	the number of the tail node in the network basis spanning tree. Used for warm starting PROC NETFLOW
STATUS	the status of the arc or nonarc variable

The variables present in the ARCDATA= data set are present in an ARCOUT= data set or a CONOUT= data set. For example, if there is a variable called tail in the ARCDATA= data set and you specified the SAS variable list

```
from tail;
```

then tail is a variable in the ARCOUT= and CONOUT= data sets instead of _FROM_. Any ID list variables also appear in the ARCOUT= and CONOUT= data sets.

NODEOUT= and DUALOUT= Data Sets

There are two types of observations in the NODEOUT= and DUALOUT= data sets. One type of observation contains information about a node. These are called *type N* observations. There is one such observation of this type for each node. The _NODE_ variable has a name of a node, and the _CON_ variable values in these observations are missing values.

The other type of observation contains information about constraints. These are called the *type C* observations. There is one such observation for each constraint. The _CON_ variable has a name of a constraint, and the _NODE_ variable values in these observations are missing values.

Many of the variables in the NODEOUT= and DUALOUT= data sets contain information used to warm start PROC NETFLOW. The variables _NODE_, _SD_, _DUAL_, _VALUE_, _RHS_, _TYPE_, and _CON_ contain information that might be of interest to you.

The NODEOUT= and DUALOUT= data sets look similar, as the same variables are in both. These variables and their values in an observation of each type are

NODE	Type N: the node name. Type C: a missing value.
_SD_	Type N: the supply (positive) or demand (negative) of the node. Type C: a missing value.
DUAL	Type N: the dual variable value of the node in _NODE_. Type C: the dual variable value of the constraint named in _CON_.
NNUMB	Type N: the number of the node named in _NODE_. Type C: the number of the constraint named in _CON_.
PRED	Type N: the predecessor in the network basis spanning tree of the node named in _NODE_. Type C: the number of the node toward which the arc with number in _ARCID_ is directed, or the constraint number associated with the slack, surplus, or artificial variable basic in this row.

TRAV Type N: the traversal thread label of the node named in _NODE_.
Type C: a missing value.

SCESS Type N: the number of successors (including itself) in the network basis spanning tree of the node named in _NODE_.
Type C: a missing value.

ARCID Type N: if _ARCID_ is nonnegative, _ARCID_ is the number of the network basis spanning tree arc directed from the node with number _PRED_ to the node named in _NODE_. If _ARCID_ is negative, minus _ARCID_ is the number of the network basis spanning tree arc directed from the node named in _NODE_ to the node with number _PRED_.
Type C: if _ARCID_ is positive, _ARCID_ is the number of the arc basic in a constraint row. If nonpositive, minus _ARCID_ is the number of the nonarc variable basic in a constraint row.

FLOW Type N: the flow minus the lower flow bound of the arc _ARCID_.
Type C: the flow minus lower flow bound of the arc _ARCID_ or value lower bound of the nonarc variable value minus _ARCID_.

FBQ Type N: if _FBQ_ is positive, then _FBQ_ is the subscript in arc length arrays of the first arc directed toward the node named in _NODE_. PROC NETFLOW's arc length arrays are sorted so that data of arcs directed toward the same head node are together. If _FBQ_ is negative, no arcs are directed toward the node named in _NODE_. Arcs directed toward node i have subscripts in the arc length arrays between observations FBQ(i) abd (FBQ($i+1$))-1, inclusive.
Type C: missing value.

VALUE Type N: missing value.
Type C: the lhs value (the sum of the products of coefficient and flows or values) of the constraint named in _CON_.

RHS Type N: missing value.
Type C: the rhs value of the constraint named in _CON_.

TYPE Type N: missing value.
Type C: the type of the constraint named in _CON_.

CON Type N: missing value.
Type C: the name of the constraint.

If specified in variable lists, the variables in the input data sets are used instead of some of the previous variables. These variables are specified in the NODE, SUPDEM, RHS, TYPE, and ROW (if there is only one variable in the ROW list) lists and are used instead of _NODE_, _SD_, _RHS_, _TYPE_, and _CON_, respectively.

Case Sensitivity

Whenever the NETFLOW procedure has to compare character strings, whether they are node names, arc names, nonarc names, or constraint names, if the two strings have different lengths, or on a character by character basis the character is different *or has different cases* PROC NETFLOW judges the character strings to be different.

Not only is this rule enforced when one or both character strings are obtained as values of SAS variables in PROC NETFLOWs input data sets, it also should be obeyed if one or both character strings were originally SAS variable names, or were obtained as the values of options or statements parsed to PROC NETFLOW. For example, if the network has only one node that has supply capability, or if you are solving a MAXFLOW or SHORTPATH problem, you can indicate that node using the SOURCE option. If you specify

```
proc netflow source=NotableNode
```

then PROC NETFLOW looks for a value of the TAILNODE list variable that is “NotableNode” (without the quotes).

Version 6 of the SAS System converts text that makes up statements into uppercase. The name of the node searched for would be NOTABLENODE, even if this was your SAS code:

```
proc netflow source=NotableNode
```

If you want PROC NETFLOW to behave as it did in Version 6, specify

```
options validvarname=v6;
```

If the SPARSECONDATA option is not specified, and you are running SAS software Version 6, or have specified options validvarname=v6; using Version 7, all NAME list variable values in the ARCDATA= data set are uppercased. This is because the SAS System has uppercased all SAS variable names, particularly those in the VAR list of the CONDATA= data set.

Entities that contain blanks must be enclosed in single or double quotes.

See the “Cautions” section on page 238 for additional discussion of case sensitivity.

Loop Arcs

When using the Primal Simplex Network algorithm, loop arcs (arcs directed toward nodes from which they originate) are prohibited. Rather, introduce a dummy intermediate node in loop arcs. For example, replace arc (A,A) with (A,B) and (B,A). B is the name of a new node, and it must be distinct for each loop arc.

Multiple Arcs

Multiple arcs with the same tail and head nodes are prohibited. PROC NETFLOW checks to ensure there are no such arcs before proceeding with the optimization. Introduce a new dummy intermediate node in multiple arcs. This node must be distinct for each multiple arc. For example, if some network has three arcs directed from node A toward node B, then replace one of these three with arcs (A,C) and (C,B) and replace another one with (A,D) and (D,B). C and D are new nodes added to the network.

Pricing Strategies

The pricing strategy is the part of the Simplex iteration that selects the nonbasic arc, constraint slack, surplus, or nonarc variable that should have a flow or value change, and perhaps enter the basis so that the total cost incurred is improved.

The pricing mechanism takes a large amount of computational effort, so it is important to use the appropriate pricing strategy for the problem under study. As in other large scale mathematical programming software, network codes can spend more than half of their execution time performing Simplex iterations in the pricing step. Some compromise must be made between using a fast strategy and improving the quality of the flow or value change candidate selection, although more Simplex iterations may need to be executed.

The configuration of the problem to be optimized has a great effect on the choice of strategy. If a problem is to be run repeatedly, experimentation on that problem to determine which scheme is best may prove worthwhile. The best pricing strategy to use when there is a large amount of work to do (for example, when a cold start is used) may not be appropriate when there is little work required to reach the optimum (such as when a warm start is used). If paging is necessary, then a pricing strategy that reduces the number of Simplex iterations performed might have the advantage. The proportion of time spent doing the pricing step during stage 1 optimization is usually less than the same proportion when doing stage 2 optimization. Therefore, it is more important to choose a stage 2 pricing strategy that causes fewer, but not necessarily the fewest, iterations to be executed.

There are many similarities between the pricing strategies for optimizing an unconstrained problem (or when constraints are temporarily ignored) and the pricing mechanisms for optimizing considering constraints. To prevent repetition, options have a suffix or embedded x . Replace x with 1 for optimization without constraint consideration and 2 for optimization with constraint consideration.

There are three main types of pricing strategy:

- PRICETYPEx=NOQ
- PRICETYPEx=BLAND
- PRICETYPEx=Q

The pricing strategy that usually performs better than the others is PRICETYPEx=Q. For this reason, PRICETYPEx=Q is the default.

PRICETYPEx=NOQ

PRICETYPEx=NOQ is the least complex pricing strategy, but it is nevertheless quite efficient. In contrast to the specification of PRICETYPEx=Q, a candidate queue is not set up.

The PxSCAN= option controls the amount of additional candidate selection work done to find a better candidate after an eligible candidate has been found .

If PxSCAN=FIRST is specified, the search for candidates finishes when the first eligible candidate is found, with this exception: if a node has more than one eligible arc directed toward it, the best such arc is chosen.

If PxSCAN=BEST is specified, everything that is nonbasic is examined, and the best candidate of all is chosen.

If PxSCAN=PARTIAL is specified, once an eligible candidate is found, the scan continues for another PxNPARTIAL= cycles in the hope that during the additional scan, a better candidate is found. Examining all nonbasic arcs directed toward a single node is counted as only one cycle.

If PxSCAN=FIRST or PxSCAN=PARTIAL is specified, the scan for entering candidates starts where the last iteration's search left off. For example, if the last iteration's scan terminated after examining arcs that are directed toward the node with internal number i, next iterations scan starts by examining arcs directed toward the node with internal number $i + 1$. If i is the largest node number, next iterations scan begins by scanning arcs directed toward node 1 (during stage 1) or scanning constraint slack or surplus variables, if any, or nonarc variables, if any, (during stage 2). During stage 2, if the scan terminated after examining the slack or surplus of constraint i, next iterations scan starts by examining the slack or surplus of the constraint with the internal number greater than i that has such a logical variable. If the scan terminated after examining the nonarc variable i, the next iterations scan starts by examining the nonarc variable with internal number $i + 1$, (or arcs directed to the node with the smallest internal number if the nonarc variable with the greatest number has been examined). This is termed a *wraparound search* .

PRICETYPEx=Q

If PRICETYPEx=Q, a queue is set up. Candidates currently on the queue are tested at each iteration and either enter the basis or are removed from the queue. The size of the queue can be specified by using the option QSIZEx=. The default value for QSIZE1= is

```
QSIZE1=number of arcs/200
if (QSIZE1<24) QSIZE1=24
else if (QSIZE1>100) QSIZE1=100
```

The default value for QSIZE2= is

```
QSIZE2=(number of arcs+number of nonarc variables)/200
if (QSIZE2<24) QSIZE2=24
else if (QSIZE2>100) QSIZE2=100
```

The PxSCAN= option controls the amount of additional candidate selection work done to find a better candidate after an eligible candidate has been found *in the queue*.

If you specify PxSCAN=BEST, the best eligible candidate found is removed from the queue. It can sustain a flow or value change and possibly enter the basis.

If you specify PxSCAN=FIRST, the first eligible candidate found is removed from the queue, and possibly sustains a flow or value change and enters the basis.

If you specify PxSCAN=PARTIAL, PxNPARTIAL= can then be specified as well. After an eligible candidate has been found in the PxNPARTIAL= more queue members are examined and the best of the eligible candidates found is chosen.

When PxSCAN=FIRST or PxSCAN=PARTIAL, the scan of the queue is wraparound. When the member last added to the queue has been examined, the scan continues from the member that was first added to the queue.

When the queue is empty, or after QSIZEx= times REFRESHQx= iterations have been executed since the queue was last refreshed, new candidates are found and put onto the queue. Valid values for the REFRESHQx= options are greater than 0.0 and less than or equal to 1.0. The default for REFRESHQx is 0.75. If the scan cannot find enough candidates to fill the queue, the procedure reduces the value of QSIZEx=. If *qfound* is the number of candidates found, the new QSIZEx= value is $qfound + ((oldQSIZEx = -qfound) * REDUCEQSIZEx =)$. Valid values of the REDUCEQSIZEx= option are between 0.0 and 1.0, inclusive. The default for REDUCEQSIZEx= is 1.0.

The QxFILLSCAN= option controls the amount of additional candidate selection work performed to find better candidates to put into the queue after the queue has been filled.

If you specify QxFILLSCAN=FIRST, the nonbasic arcs, and during stage 2 optimization, nonbasic constraint slack and surplus variables, and nonbasic nonarc variables are scanned; the scan stops when the queue is filled. If a node has more than one eligible arc directed toward it, the best such arc is put onto the queue. QxFILLSCAN=FIRST is the default.

If QxFILLSCAN=BEST is specified, everything that is nonbasic is scanned and the best eligible candidates are used to fill the queue.

If QxFILLSCAN=PARTIAL is specified, after the queue is fill, the scan continues for another QxFILLNPARTIAL= cycles in the hope that during the additional scan, better candidates are found to replace other candidates previously put onto the queue. QxFILLNPARTIAL=10 is the default. If QxFILLSCAN=FIRST or QxFILLSCAN=PARTIAL , the scan starts where the previous iteration ended; that is, it is wrap-around.

In the following section, dual variables and reduced costs are explained. These help PROC NETFLOW determine whether an arc, constraint slack, surplus, or nonarc variable should have a flow or value change. P2SCAN=ANY and the DUALFREQ= option can be specified to control stage 2 pricing, and how often dual variables and reduced costs are calculated.

What usually happens when PRICETYPE2=Q is specified is that before the first iteration, the queue is filled with nonbasic variables that are eligible to enter the basis. At the start of each iteration, a candidate on the queue is examined and its reduced cost is calculated to ensure that it is still eligible to enter the basis. If it is ineligible to enter the basis, it is removed from the queue and another candidate on the queue is examined, until a candidate on the queue is found that can enter the basis. When this happens, a *minor* iteration occurs. If there are no candidates left on the queue, or several iterations have been performed since the queue was refreshed, new nonbasic variables that are eligible to enter the basis are found and are placed on the queue. When this occurs, the iteration is termed a *major* iteration. Dual variables are calculated or maintained every iteration.

During most optimizations, if a variable is put onto the queue during a major iteration, it usually remains eligible to enter the basis in later minor iterations. Specifying P2SCAN=ANY indicates that PROC NETFLOW should choose *any* candidate on the queue and use that as the entering variable. Reduced costs are not calculated. It is simply hoped that the chosen candidate is eligible. Sometimes, a candidate on the queue is chosen that has become ineligible and the optimization takes “a step backward” rather than “a step forward” toward the optimum. However, the disadvantages of incurring an occasional step backwards and the possible danger of never converging to the optimum, are offset by not having to calculate reduced costs and, more importantly, not having to maintain dual variable values. The calculation of dual variables is one of two large linear equation systems that must be solved each iteration in the Simplex iteration.

If P2SCAN=ANY is specified, dual variables are calculated after DUALFREQ= iterations have been performed since they were last calculated. These are used to calculate the reduced costs of all the candidates currently on the queue. Any candidate found to be ineligible to enter the basis is removed from the queue. DUALFREQ=4 is the default.

Once again, the practice of not maintaining correct dual variable values is dangerous because backward steps are allowed, so the optimization is not guaranteed to converge to the optimum. However, if PROC NETFLOW does not run forever, it can find the optimum much more quickly than when the P2SCAN= option is not ANY. Before concluding that any solution is optimal, PROC NETFLOW calculates true dual variable values and reduced costs and uses these to verify that the optimum is really at hand.

Whether P2SCAN=ANY is specified or not, dual variables are always calculated at the start of major iterations.

PRICETYPEx=BLAND

PRICETYPEx=BLAND is equivalent to specifying in the PROC NETFLOW or a RESET statement all three options PRICETYPEx=NOQ, PxSCAN=FIRST, and LRATIOx , and the scans are not wraparound. Bland 1977 proved that this pivot rule prevents the Simplex algorithm from cycling. However, because the pivots concentrate on the lower indexed arcs, constraint slack, surplus, and nonarc variables, optimization with PRICETYPEx=BLAND can make the optimization execute slowly.

Dual Variables, Reduced Costs, and Status

During optimization, dual variables and reduced costs are used to determine whether an arc, constraint slack, surplus, or nonarc variable should have a flow or value change. The ARCOUT= and CONOUT= data sets each have a variable called _RCOST_ that contains reduced cost values. In the CONOUT= data set, this variable also has the reduced costs of nonarc variables. For an arc, the reduced cost is the amount that would be added to the total cost if that arc were made to convey one more unit of flow. For a nonarc variable, the reduced cost is the amount that would be added to the total cost if the value currently assigned to that nonarc variable were increased by one.

During the optimization of a minimization problem, if an arc has a positive reduced cost, PROC NETFLOW takes steps to decrease the flow through it. If an arc has a negative reduced cost, PROC NETFLOW takes steps to increase the flow through it. At optimality, the reduced costs of arcs with flow at their respective lower bounds are nonnegative; otherwise, the optimizer would have tried to increase the flow, thereby decreasing the total cost. The _STATUS_ of each such nonbasic arc is LOWERBD NONBASIC. The reduced costs of arcs with flow at capacity are nonpositive. The _STATUS_ of each such nonbasic arc is UPPERBD NONBASIC. Even though it would decrease total cost, the optimizer cannot increase the flows through such arcs because of the capacity bound. Similar arguments apply for nonarc variables.

The reduced cost is also the amount that would be subtracted from the total cost if that arc was made to convey one less unit of flow. Similarly, a reduced cost is the amount subtracted from the total cost if the value currently assigned to that nonarc variable is decreased by one.

The dual variables and reduced costs can be used to detect whether multiple optimal solutions exist. A zero reduced cost of a nonbasic arc indicates the existence of multiple optimal solutions. A zero reduced cost indicates, by definition, that the flow through such arcs can be changed with zero change to the total cost. (Basic arcs and basic nonarc variables technically have zero reduced costs. A missing value is used for these so that reduced costs of nonbasic arcs and nonbasic nonarc variables that are zero are highlighted.)

The range over which costs can vary before the present solution becomes nonoptimal can be determined through examination of the reduced costs. For any nonbasic arc with assigned flow equal to its lower bound, the amount by which the cost must be decreased before it becomes profitable for this arc to convey additional flow is the value of its reduced cost. The cost reduction necessary for a nonbasic arc currently assigned capacity flow to undergo a worthwhile flow decrease is the absolute value of its reduced cost. In both cases, this minimum cost reduction changes the reduced cost to zero. Any further reduction promotes a possible basis change.

The reduced cost of an arc (t, h) is $rc_{t,h} = c_{t,h} - \pi_t + \pi_h$ where π_i is the dual value for node i and $c_{t,h}$ is the cost of the arc with tail node t and head node h.

If the problem has side constraints and arc (t, h) has nonzero lhs coefficients, then the following term must be subtracted from $rc_{t,h}$.

$$\sum_i condual_i H_{i,(t,h)}$$

where $condual_i$ is the dual variable of constraint i, $H_{i,(t,h)}$ is the coefficient of arc (t, h) in constraint i.

If d_n is the objective function coefficient of nonarc variable n, the reduced cost is $rc_n = d_n - \sum_i condual_i Q_{i,n}$ where $Q_{i,n}$ is the coefficient of nonarc variable n in constraint i.

The Working Basis Matrix

Let $\mathbf{T}$ be the basis matrix of NPSC. The following partitioning is done:

$$\mathbf{T} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

where

- n is the number of nodes
- k is the number of side constraints
- $\mathbf{A}$ (n x n) is the network component of the basis. Most of the columns of this matrix are columns of the problem's node-arc incidence matrix. The arcs associated with columns of $\mathbf{A}$, called key basic variables or key arcs, form a spanning tree. The data structures of the spanning tree of this submatrix of the basis $\mathbf{T}$ enable the computations involving $\mathbf{T}$ and the manner in that $\mathbf{T}$ is updated to be very efficient, especially those dealing with $\mathbf{A}$ (or $\mathbf{A^{-1}}$).
- $\mathbf{C}$ (k x n) are the key arcs' side constraint coefficient columns.
- $\mathbf{B}$ (n x k) are the node-arc incidence matrix columns of the nontree arcs. The columns of **B** having nonzero elements are associated with basic nonspanning tree arcs.
- $\mathbf{D}$ (k x k) are the constraint coefficient columns of nonkey basic variables. Nonkey basic variables not only include nontree basic arcs but also basic slack, surplus, artificial, or nonarc variables.

It is more convenient to factor $\mathbf{T}$ by block triangular matrices $\mathbf{P}$ and $\mathbf{M}$, such that $\mathbf{P} = \mathbf{TM}$. The matrices $\mathbf{P}$ and $\mathbf{M}$ are used instead of $\mathbf{T}$ because they are less burdensome to work with. You can perform block substitution when solving the Simplex iteration linear systems of equations.

$$\mathbf{P} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{C} & \mathbf{D_w} \end{bmatrix}$$

$$\mathbf{M} = \begin{bmatrix} \mathbf{I} & -\mathbf{A^{-1}B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

where $\mathbf{D_w} = \mathbf{D} - \mathbf{CA^{-1}B}$ and is called the working basis matrix.

To perform block substitution, you need the tree data structure of the $\mathbf{A}$ matrix, also the $\mathbf{C}$, $\mathbf{B}$, and $\mathbf{D_w}$ matrices. Because the $\mathbf{C}$ matrix consists of columns of the constraint coefficient matrix, the maintenance of $\mathbf{C}$ from iteration to iteration simply entails changing information specifying which columns of the constraint coefficient matrix compose $\mathbf{C}$.

The $\mathbf{A^{-1}B}$ matrix is usually very sparse. Fortunately, the information in $\mathbf{A^{-1}B}$ can be initialized easily using the tree structures. In most iterations, only one column is replaced by a new one. The values of the elements of the new column may already be known from preceding steps of the Simplex iteration.

The working basis matrix is the submatrix that presents the most computational complexity. However, PROC NETFLOW usually can use classical Simplex pivot techniques. In many iterations, only one column of $\mathbf{D_w}$ changes. Sometimes it is not necessary to update $\mathbf{D_w}$ or its inverse at all.

If INVD_2D is specified in the PROC NETFLOW statement, only one row and one column may need to be changed in the $\mathbf{D_w^{-1}}$ before the next Simplex iteration can begin. The new contents of the changed column are already known. The new elements of the row that changes are influenced by the contents of a row of $\mathbf{A^{-1}B}$ that is very sparse.

If INVD_2D is not specified in the PROC NETFLOW statement, the Bartels-Golub update can be used to update the $\mathbf{LU}$ factors of $\mathbf{D_w}$. The choice must be made whether to perform a series of updates (how many depends on the number of nonzeros in a row of $\mathbf{A^{-1}B}$), or refactorization.

Flow and Value Bounds

The capacity and lower flow bound of an arc can be equal. Negative arc capacities and lower flow bounds are permitted. If both arc capacities and lower flow bounds are negative, the lower flow bound must be at least as negative as the capacity. An arc (A, B) that has a negative flow of $-f$ units can be interpreted as an arc that conveys f units of flow from node B to node A.

The upper and lower value bound of a nonarc variable can be equal. Negative upper and lower bounds are permitted. If both are negative, the lower bound must be at least as negative as the upper bound.

Tightening Bounds and Side Constraints

If any piece of data is furnished to PROC NETFLOW more than once, PROC NETFLOW checks for consistency so that no conflict exists concerning the data values. For example, if the cost of some arc is seen to be one value and as more data are read, the cost of the same arc is seen to be another value, PROC NETFLOW issues an error message on the SAS log and stops. There are two exceptions to this.

- The bounds of arcs and nonarc variables are made as tight as possible. If several different values are given for the lower flow bound of an arc, the greatest value is used. If several different values are given for the lower bound of a nonarc variable, the greatest value is used. If several different values are given for the capacity of an arc, the smallest value is used. If several different values are given for the upper bound of a nonarc variable, the smallest value is used.

- Several values can be given for inequality constraint right-hand-sides. For a particular constraint, the lowest rhs value is used for the rhs if the constraint is of *less than or equal to* type. For a particular constraint, the greatest rhs value is used for the rhs if the constraint is of *greater than or equal to* type.

Reasons for Infeasibility

Before optimization commences, PROC NETFLOW tests to ensure that the problem is not infeasible by ensuring that, with respect to supplies, demands, and arc flow bounds, flow conservation can be obeyed at each node.

- Let IN be the sum of lower flow bounds of arcs directed toward a node plus the node's supply. Let OUT be the sum of capacities of arcs directed from that node plus the node's demand. If IN exceeds OUT, not enough flow can leave the node.
- Let OUT be the sum of lower flow bounds of arcs directed from a node plus the node's demand. Let IN be the total capacity of arcs directed toward the node plus the node's supply. If OUT exceeds IN, not enough flow can arrive at the node.

Reasons why a network problem can be infeasible are similar to those previously mentioned but apply to a set of nodes rather than for an individual node.

Consider the network illustrated in Figure 4.13.

```
                NODE_1----------------->NODE_2
               /          capac=55           \
              /              lo=50            \
             /                                 \
            /                                   \
           /                                     \
     NODE_3                                       NODE_4
supply=100 \                                     / demand=120
            \                                   /
             \                                 /
              \           capac=62            /
               \             lo=60           /
                NODE_5----------------->NODE_6
```

Figure 4.13. An infeasible network.

The demand of NODE_4 is 120. That can never be satisfied because the maximal flow through arcs (NODE_1, NODE_2) and (NODE_5, NODE_6) is 117. More specifically, the implicit supply of NODE_2 and NODE_6 is only 117, which is insufficient to satisfy the demand of other nodes (real or implicit) in the network.

Furthermore, the lower flow bounds of arcs (NODE_1, NODE_2) and (NODE_5, NODE_6) are greater than the flow that can reach the tail nodes of these arcs, that, by coincidence, is the total supply of the network. The implicit demand of nodes NODE_1 and NODE_5 is 110, which is greater than the amount of flow that can reach these nodes.

When PROC NETFLOW detects that the problem is infeasible, it indicates why the solution, obtained after optimization stopped, is infeasible. It can report that the solution cannot obey flow conservation constraints and which nodes these conservation constraints are associated with. If applicable, the side constraints that the solution disobeys are also output.

If stage 1 optimization obtains a feasible solution to the network, stage 2 optimization can determine that the problem is infeasible and note that some flow conservation constraint is broken while all side constraints are obeyed. The infeasibility messages issued by PROC NETFLOW pertain to why the *current* solution is infeasible, not quite the same as the reasons why the *problem* is infeasible. However, the messages highlight *areas* in the problem where the infeasibility can be tracked down. If the problem is infeasible, make PROC NETFLOW do a stage 1 unconstrained optimization by removing the CONDATA= data set specification in the PROC NETFLOW statement. If a feasible network solution is found, then the side constraints are the source of the infeasibility in the problem.

Missing S Supply and Missing D Demand Values

In some models, you may want a node to be either a supply or demand node but you want the node to supply or demand the optimal number of flow units. To indicate that a node is such a supply node, use a missing S value in the SUPPLY list variable in the ARCDATA= data set or the SUPDEM list variable in the NODEDATA= data set. To indicate that a node is such a demand node, use a missing D value in the DEMAND list variable in the ARCDATA= data set or the SUPDEM list variable in the NODEDATA= data set.

Once a missing S or missing D value is found in the input data sets, the THRUNET option is automatically made active.

Suppose the Oil example in the “Introductory Example” section on page 197 is changed so that crude oil can be obtained from either the Middle East or U.S.A. in any amounts. You should specify that the node “middle east” is a supply node, but you do not want to stipulate that it supplies 100 units, as before. The node “u.s.a.” should also remain a supply node, but you do not want to stipulate that it supplies 80 units. You must specify that these nodes have missing S supply capabilities.

```
title  'Oil Industry Example';
title3 'Crude Oil can come from anywhere';
data miss_s;
   missing S;
   input   _node_&$15. _sd_;
   datalines;
middle east         S
u.s.a.              S
servstn1 gas      -95
servstn1 diesel   -30
```

```
servstn2 gas      -40
servstn2 diesel   -15
;
```

The following PROC NETFLOW run uses the same ARCDATA= and CONDATA= data sets used in the “Introductory Example” section on page 197.

```
proc netflow
   nodedata=miss_s       /* the supply (missing S) and */
                         /* demand data                */
   arcdata=arcd1         /* the arc descriptions       */
   condata=cond1         /* the side constraints       */
   conout=solution;      /* the solution data set      */
run;
print some_arcs('middle east' 'u.s.a.',_all_)/short;
proc print;sum _fcost_;run;
```

The following messages appear on the SAS log:

```
NOTE: Number of nodes= 14 .
NOTE: Number of supply nodes= 2 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 0 , total demand= 180 .
NOTE: Number of arcs= 18 .
NOTE: Number of iterations performed (neglecting any
      constraints)= 9 .
NOTE: Of these, 0 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= 50040 .
NOTE: Number of <= side constraints= 0 .
NOTE: Number of == side constraints= 2 .
NOTE: Number of >= side constraints= 2 .
NOTE: Number of arc and nonarc variable side
      constraint coefficients= 8 .
NOTE: Number of iterations, optimizing with
      constraints= 3 .
NOTE: Of these, 0 were degenerate.
NOTE: Optimum reached.
NOTE: Minimal total cost= 50075 .
```

The PRINT statement reports the arcs directed away from the supply nodes, shown in Figure 4.14. The amount of crude obtained from the Middle East and U.S.A. is 145 and 35 units.

Oil Industry Example

Crude Oil can come from anywhere

NETFLOW PROCEDURE

_N_	_from_	_to_	_cost_	_capac_	_lo_	_name_
1	middle east	refinery 1	63	95	20	m_e_ref1
2	u.s.a.	refinery 1	55	99999999	0	
3	middle east	refinery 2	81	80	10	m_e_ref2
4	u.s.a.	refinery 2	49	99999999	0	

_N_	_FLOW_
1	20
2	125
3	10
4	25

Figure 4.14. Print statement, Oil example, missing S supplies.

The CONOUT= data set is shown in Figure 4.15.

Obs	_from_	_to_	_cost_	_capac_	_lo_	_name_	_SUPPLY_
1	refinery 1	r1	200	175	50	thruput1	.
2	refinery 2	r2	220	100	35	thruput2	.
3	r1	ref1 diesel	0	75	0		.
4	r1	ref1 gas	0	140	0	r1_gas	.
5	r2	ref2 diesel	0	75	0		.
6	r2	ref2 gas	0	100	0	r2_gas	.
7	middle east	refinery 1	63	95	20	m_e_ref1	S
8	u.s.a.	refinery 1	55	99999999	0		S
9	middle east	refinery 2	81	80	10	m_e_ref2	S
10	u.s.a.	refinery 2	49	99999999	0		S
11	ref1 diesel	servstn1 diesel	18	99999999	0		.
12	ref2 diesel	servstn1 diesel	36	99999999	0		.
13	ref1 gas	servstn1 gas	15	70	0		.
14	ref2 gas	servstn1 gas	17	35	5		.
15	ref1 diesel	servstn2 diesel	17	99999999	0		.
16	ref2 diesel	servstn2 diesel	23	99999999	0		.
17	ref1 gas	servstn2 gas	22	60	0		.
18	ref2 gas	servstn2 gas	31	99999999	0		.

Obs	_DEMAND_	_FLOW_	_FCOST_	_RCOST_	_ANUMB_	_TNUMB_	_STATUS_
1	.	145.00	29000.00	.	7	2	KEY_ARC BASIC
2	.	35.00	7700.00	17	8	3	LOWERBD NONBASIC
3	.	36.25	0.00	.	10	5	KEY_ARC BASIC
4	.	108.75	0.00	.	9	5	KEY_ARC BASIC
5	.	8.75	0.00	.	12	6	KEY_ARC BASIC
6	.	26.25	0.00	.	11	6	KEY_ARC BASIC
7	.	20.00	1260.00	8	2	1	LOWERBD NONBASIC
8	.	125.00	6875.00	.	3	4	KEY_ARC BASIC
9	.	10.00	810.00	32	4	1	LOWERBD NONBASIC
10	.	25.00	1225.00	.	5	4	KEY_ARC BASIC
11	30	30.00	540.00	.	17	8	KEY_ARC BASIC
12	30	0.00	0.00	12	18	10	LOWERBD NONBASIC
13	95	68.75	1031.25	.	13	7	KEY_ARC BASIC
14	95	26.25	446.25	.	14	9	NONKEY ARC BASIC
15	15	6.25	106.25	.	19	8	KEY_ARC BASIC
16	15	8.75	201.25	.	20	10	KEY_ARC BASIC
17	40	40.00	880.00	.	15	7	KEY_ARC BASIC
18	40	0.00	0.00	7	16	9	LOWERBD NONBASIC
			========				
			50075.00				

Figure 4.15. Missing S SUPDEM values in NODEDATA

The optimal supplies of nodes “middle east” and “u.s.a.” are 145 and 35 units, respectively. For this example, the same optimal solution is obtained if these nodes had supplies less than these values (each supplies 1 unit, for example) and the THRUNET option was specified in the PROC NETFLOW statement. With the THRUNET option active, when total supply exceeds total demand, the specified nonmissing demand values are the lowest number of flow units that must be absorbed by the corresponding node. This is demonstrated in the following PROC NETFLOW run. The missing S is most useful when nodes are to supply optimal numbers of flow units and it turns out that for some nodes, the optimal supply is zero.

```
data miss_s_x;
   missing S;
   input   _node_&$15. _sd_;
   datalines;
middle east         1
u.s.a.              1
servstn1 gas      -95
servstn1 diesel   -30
servstn2 gas      -40
servstn2 diesel   -15
;
proc netflow
   thrunet
   nodedata=miss_s_x     /* No supply (missing S)      */
   arcdata=arcd1         /* the arc descriptions       */
   condata=cond1         /* the side constraints       */
   conout=solution;      /* the solution data set      */
run;
print some_arcs('middle east' 'u.s.a.',_all_)/short;
proc print;sum _fcost_;run;
```

The following messages appear on the SAS log. Note that the Total supply= 2, not zero as in the last run.

```
NOTE: Number of nodes= 14 .
NOTE: Number of supply nodes= 2 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 2 , total demand= 180 .
NOTE: Number of arcs= 18 .
NOTE: Number of iterations performed (neglecting any
      constraints)= 13 .
NOTE: Of these, 0 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= 50040 .
NOTE: Number of <= side constraints= 0 .
NOTE: Number of == side constraints= 2 .
NOTE: Number of >= side constraints= 2 .
NOTE: Number of arc and nonarc variable side
      constraint coefficients= 8 .
NOTE: Number of iterations, optimizing with
      constraints= 3 .
NOTE: Of these, 0 were degenerate.
```

```
NOTE: Optimum reached.
NOTE: Minimal total cost= 50075 .
```

The PRINT statement and the CONDATA= data set are very similar; the supplies of the supply nodes are 1, not missing S. Otherwise, the solutions are identical.

If total supply exceeds total demand, any missing S values are ignored. If total demand exceeds total supply, any missing D values are ignored.

Warm Starts

Using a warm start can increase the overall speed of PROC NETFLOW when it is used repetitively on problems with similar structure. It is most beneficial when a solution of a previous optimization is close to the optimum of the same network with some of its parameters, for example, arc costs, changed. Whether a problem is changed or not, a nonoptimal solution resulting from a previous optimization can be used to restart optimization, thereby saving PROC NETFLOW from having to repeat work to reach the warm start already available.

Time also is saved in the data structure initialization part of the NETFLOW procedure's execution. Information about the previous optimal solution, particularly concerning the size of the problem, a description of the basis spanning tree structure, and what is basic in constraint rows, is known. Information about which nonbasic arcs have capacity flow and which nonbasic nonarc variables are at their respective upper bounds also makes up part of the warm start. The procedure can place arc data into the internal arc length arrays in precisely defined locations, in order of ascending head node internal number. It is not necessary to have multiple passes through the data because literals such as node, nonarc variable, arc, constraint, and special row names are defined and meaning is attached to each. This saves a considerable amount of memory as well. None of the pre-optimization feasibility checks need be repeated.

Warm starts also are useful if you want to determine the effect of arcs being closed to carrying flow. The costs of these arcs are set high enough to ensure that the next optimal solution never has flow through them. Similarly, the effect of opening arcs can be determined by changing the cost of such arcs from an extreme to a reasonable value.

Specify the FUTURE1 or FUTURE2 option to ensure that additional data about a solution to be used as a warm start are output to output data sets. If the FUTURE1 option is specified, extra observations with information on what is to be the warm start are set up for the NODEOUT= and ARCOUT= data sets. The warm start solution in these data sets is a solution obtained after optimization neglecting side constraints. Any cost list variable value in the ARCOUT= data set (and, if there are side constraints, any constraint data in the CONDATA= data set) can be changed before the solution is used as a warm start in a subsequent PROC NETFLOW run. Any nonarc variable data in the CONDATA= data set can be changed at this time as well. New nonarc variables not present in the original problem when the warm start was generated can also be added to the CONDATA= data set before the problem is warm started.

If the FUTURE2 option is specified, extra variables containing information on what will be the warm start solution are set up for the DUALOUT= and CONOUT= data sets. The warm start solution in these data sets is obtained after optimization that considers side constraints has been performed. Part of the warm start is concerned with the constraint part of the basis. Only cost list variable values in the CONOUT= data set can be changed before the solution is used as a warm start in a subsequent PROC NETFLOW run.

If a Primal Simplex optimization is to use a warm start, the WARM option must be specified in the PROC NETFLOW statement. Otherwise, the Primal Simplex Network algorithm processes the data for a cold start and the extra information is not used.

The ARCDATA= data set is either the ARCOUT= data set from a previous run of PROC NETFLOW with the FUTURE1 option specified (if an unconstrained warm start is used) or the CONOUT= data set from a previous run of PROC NETFLOW with the FUTURE2 option specified (if the warm start was obtained after optimization that considers side constraints was used).

The NODEDATA= data set is the NODEOUT= data set from a previous run of PROC NETFLOW with FUTURE1 specified if an unconstrained warm start is being used. Otherwise, the DUALIN= is the DUALOUT= data sets from a previous run of PROC NETFLOW with FUTURE2 specified, if the warm start was obtained after optimization that considers side constraints was used.

You never need to alter the NODEOUT= data set or the DUALOUT= data set between the time they are generated and when they are used as a warm start. The results would be unpredictable if incorrect changes were made to these data sets, or if a NODEDATA= or a DUALIN= data set were used with an ARCDATA= data set of a different solution.

It is possible, and often useful, to specify WARM and either FUTURE1 or FUTURE2, or both, in the same PROC NETFLOW statement if a new warm start is to be generated from the present warm start.

The extent of the changes allowed to a Primal Simplex warm start between the time it is generated and when it is used depends on whether the warm start describes an unconstrained or constrained solution. The following list describes parts of a constrained or an unconstrained warm start that can be altered:

- COST list variable values
- the value of an arc's capacity, as long as the new capacity value is not less than the lower flow bound or the flow through the arc
- any nonarc variable information, in an unconstrained warm start
- for an unconstrained warm start, any side constraint data

The changes that can be made in constraint data for a constrained warm start are more restrictive than those for an unconstrained warm start. The lhs coefficients type, and rhs value of a constraint can be changed as long as that constraint's slack, surplus, or artificial variable is basic. The constraint name cannot be changed.

Example of a Warm Start

The following sample SAS session demonstrates how the warm start facilities are used to obtain optimal solutions to an unconstrained network where some arc cost changes occur or optimization is halted before the optimum is found.

```
               /* data already in data sets node0 and arc0 */
proc netflow
     nodedata=node0  /* if supply_demand information  */
                     /* is in this SAS data set       */
     arcdata=arc0;
               /* variable list specifications go here    */
               /* assume that they are not necessary here */
               /* if they are, they must be included in   */
               /* all the PROC NETFLOW calls that follow  */
   reset
      future1
      nodeout=node2  /* nodeout and arcout are necessary  */
                     /* when FUTURE1 is used              */
      arcout=arc1;
proc print
      data=arc1;     /* display the optimal solution      */
proc fsedit
      data=arc1;     /* change some arc costs             */
data arc2;
   reset arc1;
      oldflow=_flow_;
      oldfc=_fcost_;
               /* make duplicates of the flow and flowcost*/
               /* variables. If a id list was explicitly  */
               /* specified, add oldflow and oldfc to this*/
               /* list so that they appear in subsequently*/
               /* created arcout= data sets               */
```

The following PROC NETFLOW uses the warm start created previously, performs 250 stage 2 iterations and saves that solution, which (as future1 and arcout= data set and nodeout= data set are specified) can be used as a warm start in another PROC NETFLOW run.

```
proc netflow
     warm
     nodedata=node2
     arcdata=arc2;
   reset
     maxit1=250
     future1;
   run;
   save
     nodeout=savelib.node3
     arcout=savelib.arc3;
            /* optimization halted because 250 iterations */
```

```
            /* were performed to resume optimization,     */
            /* possibly in another session (the output    */
            /* data sets were saved in a SAS library      */
            /* called savelib)                            */
```

Using the latest warm start, PROC NETFLOW is re-envoked to find the optimal solution.

```
proc netflow
     warm
     nodedata=savelib.node3
     arcdata=savelib.arc3;
   reset
     future1
     nodeout=node4
     arcout=arc4;
run;
```

If this problem has constraints with data in a data set called CON0, then in each of the previous PROC NETFLOW statements, specify CONDATA=CON0. Between PROC NETFLOW runs, you can change constraint data. In each of the RESET statements, you could specify the CONOUT= data set to save the last (possibly optimal) solution reached by the optimizer if it reaches stage 2. You could specify FUTURE2 and the DUALOUT= data set to generate a constrained warm start.

```
proc netflow
     warm
     nodedata=node4
     arcdata=arc4
     condata=con0;
   reset
     maxit2=125 /* optional, here as a reason why         */
                /* optimum will not be obtained           */
     scratch    /* optional, but warm start might be good */
                /* enough to start stage 2 optimization   */
     future2
   run;
     /* optimization halted after 125 stage 2 iterations  */
   save dualout=dual1 conout=conout1;
```

Stage 2 optimization halted before optimum was reached. Now you can make cost and nonarc variable objective function coefficient changes. Then to restart optimization, use

```
proc netflow
     warm
     condata=con0
           /* NB. NETFLOW reads constraint data only     */
     dualin=dual1
     arcdata=con1;
   reset
```

```
      future2
      dualout=dual2
      conout=con2;
   run;
```

How to Make the Data Read of PROC NETFLOW More Efficient

This section contains information useful when you want to solve large constrained network problems. However, much of this information is also useful if you have a large linear programming problem. All of the options described in this section that are not directly applicable to networks (options such as ARCS_ONLY_ARCDATA ARC_SINGLE_OBS, NNODES, and NARCS) can be specified to improve the speed at which LP data is read.

Large Constrained Network Problems

Many of the models presented to PROC NETFLOW are enormous. They can be considered large by linear programming standards; problems with thousands of variables and constraints. When dealing with side constrained network programming problems, models can have not only a linear programming component of that magnitude, but also a larger, possibly *much* larger, network component.

The majority of network problem's decision variables are arcs. Like an LP decision variable, an arc has an objective function coefficient, upper and lower value bounds, and a name. Arcs can have coefficients in constraints. Therefore, an arc is quite similar to an LP variable and places the same memory demands on optimization software as an LP variable. But a typical network model has many more arcs and nonarc variables than the typical LP model has variables. And arcs have tail and head nodes. Storing and processing node names require huge amounts of memory. To make matters worse, node names occupy memory at times when alot of other data should reside in memory as well.

While memory requirements are lower for a model with embedded network component compared with the equivalent LP *once optimization starts*, the same is usually not true *during the data read*. Even though nodal flow conservation constraints in the LP should not be specified in the constrained network formulation, the memory requirements to read the latter are greater because each arc (unlike an LP variable) originates at one node, and is directed toward another.

Paging

PROC NETFLOW has facilities to read data when the available memory is insufficient to store all the data at once. PROC NETFLOW does this by allocating memory for different purposes, for example, to store an array or receive data read from an input SAS data set. After that memory has filled, the information is sent to disk and PROC NETFLOW can resume filling that memory with new information. Often, information must be retrieved from disk so that data previously read can be examined or checked for consistency. Sometimes, to prevent any data from being lost, or to retain any changes made to the information in memory, the contents of the memory must be sent to disk before other information can take its place. This process of swapping information to and from disk is called paging. Paging can be very time consuming, so it is crucial to minimize the amount of paging performed.

There are several steps you can take to make PROC NETFLOW read the data of network and linear programming models more efficiently, particularly when memory is scarse and the amount of paging must be reduced. PROC NETFLOW will then be able to tackle large problems in what can be considered reasonable amounts of time.

The Order of Observations

PROC NETFLOW is quite flexible in the ways data can be supplied to it. Data can be given by any reasonable means. PROC NETFLOW has convenient defaults that can save you work when generating the data. There can be several ways to supply the same piece of data, and some pieces of data can be given more than once. PROC NETFLOW reads everything, then merges it all together. However, this flexibility and convenience come at a price; PROC NETFLOW may not assume the data has a characteristic that, if possessed by the data, could save time and memory during the data read. There are several options that indicate the data has some exploitable characteristic.

For example, an arc cost can be specified once or several times in the ARCDATA= or, CONDATA= data set, or both. Every time it is given in ARCDATA, a check is made to ensure that the new value is the same as any corresponding value read in a previous observation of ARCDATA. Every time it is given in CONDATA, a check is made to ensure that the new value is the same as the value read in a previous observation of CONDATA, or previously in ARCDATA. It would save PROC NETFLOW time if it knew that arc cost data would be encountered only once while reading ARCDATA, so performing the time consuming check for consistency would not be necessary. Also, if you indicate that CONDATA contains data for constraints only, PROC NETFLOW will not expect any arc information, so memory will not be allocated to receive such data while reading CONDATA. This memory is used for other purposes and this might lead to a reduction in paging. If applicable, use the ARC_SINGLE_OBS or the CON_SINGLE_OBS option, or both, and the NON_REPLIC=COEFS specification to improve how ARCDATA and CONDATA are read.

PROC NETFLOW allows the observations in input data sets to be in any order. However, major time savings can result if you are prepared to order observations in particular ways. Time spent by the SORT procedure to sort the input data sets, particularly the CONDATA= data set, may be more than made up for when PROC NETFLOW reads them, because PROC NETFLOW has in memory information possibly used when the previous observation was read. PROC NETFLOW can assume a piece of data is either similar to that of the last observation read or is new. In the first case, valuable information such as an arc or a nonarc variable number or a constraint number is retained from the previous observation. In the last case, checking the data with what has been read previously is not necessary.

Even if you do not sort the CONDATA= data set, grouping observations that contain data for the same arc or nonarc variable or the same row pays off. PROC NETFLOW establishes whether an observation being read is similar to the observation just read.

Practically, lots of input data sets for PROC NETFLOW might have this characteristic, because it is natural for data for each constraint to be grouped together (dense format of CONDATA) or data for each column to be grouped together (sparse format of CONDATA). If data for each arc or nonarc is spread over more than one observation of the ARCDATA= data set, it is natural to group these observation together.

Use the GROUPED parameter to indicate whether observations of the ARCDATA= data set, CONDATA= data set, or both, are grouped in a way that can be exploited during data read.

Time is saved if the type data for each row appears near the top of the CONDATA= data set, especially if it has the sparse format. Otherwise, when reading an observation, if PROC NETFLOW does not know if a row is a constraint or special row, so the data is set aside. Once the data set has been completely read, PROC NETFLOW must reprocess the data it set aside. By then, it knows the type of each constraint or row or, if it's type was not provided, it is assumed to have a default type.

Better Memory Utilization

In order for PROC NETFLOW to make better utilization of available memory, you can now specify options that indicate the approximate size of the model. PROC NETFLOW then knows what to expect. For example, if you indicate that the problem has no nonarc variables, PROC NETFLOW will not allocate memory to store nonarc data. That memory is utilized better for other purposes. Memory is often allocated to receive or store data of some type. If you indicate that the model does not have much data of a particular type, the memory that would otherwise have been allocated to receive or store that data can be used to receive or store data of another type.

- NNODES= approximate number of nodes
- NARCS= approximate number of arcs
- NNAS= approximate number of nonarc variables or LP variables
- NCONS= approximate number of constraints
- NCOEFS= approximate number of constraint coefficients

These options will sometimes be referred to as Nxxxx= options.

You do not need to specify all these option for the model, but the more you do, the better. If you do not specify some or all of these options, PROC NETFLOW guesses the size of the problem by using what it already knows about the model. Sometimes PROC NETFLOW guesses the size of the model by looking at the number of observations in the ARCDATA= and CONDATA= data sets. However, PROC NETFLOW uses rough rules of thumb; that typical models are proportioned in certain ways (for example, if there are constraints, arcs and nonarcsusually have 5 constraint coefficients). If your model has an unusual shape or a disproportionate number of something, you are encouraged to use these options.

If you do use the options, if you do not know the exact values to specify, *overestimate* the values. For example, if you specify NARCS=10000 but the model has 10100 arcs, when dealing with the last 100 arcs, PROC NETFLOW might have to page out data for 10000 arcs each time one of the last arcs must be dealt with. Memory could have been allocated for all 10100 arcs without affecting (much) the rest of the data read, so NARCS=10000 could be more of a hindrance than a help.

The point of these Nxxxx= options is to indicate the model size when PROC NETFLOW does not know it. When PROC NETFLOW knows the "real" value, that value is used instead of Nxxxx= .

When PROC NETFLOW is given a constrained solution warm start, PROC NETFLOW knows from the warm start information all model size parameters, so Nxxxx= options are not used. When an unconstrained warm start is used and the SAME_NONARC_DATA is specified, PROC NETFLOW knows the number of nonarc variables, so that is used instead of the value of the NNAS= option.

ARCS_ONLY_ARCDATA indicates that data for only arcs are in the ARCDATA= data set. Memory would not be wasted to receive data for nonarc and LP variables.

Use the memory usage parameters:

- BYTES= size of PROC NETFLOW main working memory in number of bytes
- MAXARRAYBYTES= maximum number of bytes that an array can occupy
- MEMREP indicates that memory usage report is to be displayed on the SAS log

Specifying the BYTES= parameter is particularly important. Specify as large a number as possible, but not such a large number of bytes that will cause PROC NETFLOW (rather, the SAS System running underneath PROC NETFLOW) to run out of memory. Use the MAXARRAYBYTES= option if the model is really large or “disproportionate”. Try increasing or decreasing the MAXARRAYBYTES= option. Limiting the amount of memory for use by big arrays is good if they would take up too much memory to the detriment of smaller arrays, buffers, and other things that require memory. However, too small a value of the MAXARRAYBYTES= option might cause PROC NETFLOW to page a big array excessively. Never specify a value for the MAXARRAYBYTES= option that is smaller than the main node length array. PROC NETFLOW reports the size of this array on the SAS log if you specify the MEMREP option. The MAXARRAYBYTES= option influences paging not only in the data read, but also during optimization. It is often better if optimization is performed as fast as possible, even if the read is made slower as a consequence.

Use Defaults to Reduce the Amount of Data

Use as much as possible the parameters that specify default values. For example, if there are lots of arcs with the same cost value *c*, use DEFCOST=*c* for arcs that have that cost. Use missing values in the COST variable in ARCDATA instead of *c*. PROC NETFLOW ignores missing values, but must read, store, and process nonmissing values, even if they are equal to a default option or could have been equal to a default parameter had it been specified. Sometimes, using default parameters makes the need for some SAS variables in the ARCDATA= and CONDATA= data sets no longer necessary, or reduces the quantity of data that must be read. The default options are:

- DEFCOST= default cost of arcs, objective function of nonarc variables or LP variables

- DEFMINFLOW= default lower flow bound of arcs, lower bound of nonarc variables or LP variables
- DEFCAPACITY= default capacity of arcs, upper bound of nonarc variables or LP variables
- DEFCONTYPE=LE DEFCONTYPE=<=
  DEFCONTYPE=EQ DEFCONTYPE==
  DEFCONTYPE=GE DEFCONTYPE=>= default constraint type

The default options themselves have defaults. For example, you do not need to specify DEFCOST=0 in the PROC NETFLOW statement. You should still have missing values in the COST variable in ARCDATA for arcs that have zero costs.

If the network has only one supply node, one demand node of both use

- SOURCE= name of single node that has supply capability
- SUPPLY= the amount of supply of SOURCE
- SINK= name of single node that demands flow
- DEMAND= the amount of flow SINK demands

Do not specify that a constraint has zero right-hand-side values. That is the default. The only time it might be practical to specify a zero rhs is in observations of CONDATA read early so that PROC NETFLOW can infer that a row is a constraint. This could prevent coefficient data from being put aside because PROC NETFLOW did not know the row was a constraint.

Names of Things

To cut data read time and memory requirements, reduce the number of bytes in the longest node name, longest arc name, and longest constraint name to 8 bytes or less. The longer a name, the more bytes must be stored and compared with other names.

If an arc has no constraint coefficients, do not give it a name in the NAME list variable in the ARCDATA= data set. Names for such arcs serve no purpose.

PROC NETFLOW can have a default name for each arc. If an arc is directed from node *tailname* toward node *headname*, the default name for that arc is *tailname_headname*. If you do not want PROC NETFLOW to use these default arc names, specify NAMECTRL=1. Otherwise, PROC NETFLOW must use memory for storing node names and these node names must be searched often.

If you want to use the default *tailname_headname name*, that is, NAMECTRL=2 or NAMECTRL=3, do not use underscores in node names. If a CONDATA has a dense format and has a variable in the VAR list A_B_C_D, or if the value A_B_C_D is encountered as a value of the COLUMN list variable when reading CONDATA that has the sparse format, PROC NETFLOW first looks for a node named A. If it finds it, it looks for a node called B_C_D. It then looks for a node with the name A_B and possibly a node with name C_D. A search for a node named A_B_C and possibly a node named D is done. Underscores could have caused PROC NETFLOW to look unnecessarily for nonexistant nodes. Searching for node names can be expensive, and the amount of memory to store node names large. It might be better to assign the arc name A_B_C_D directly to an arc by having that value as a NAME list variable value for that arc in ARCDATA and specify NAMECTRL=1.

Other Ways to Speed-up Data Reads

Use warm starts as much as possible.

- WARM indicates that the input SAS data sets contain a warm start.

The data read of a warm start is much faster than a cold start data read. The model size is known before the read starts. The observations of the NODEDATA or DUALIN data sets have observations ordered by node name and constraint name. Information is stored directly into the data structures used by PROC NETFLOW. For a cold start, much of preprocessing must be performed before the information can be stored in the same way. And using a warm start can greatly reduce the time PROC NETFLOW spends doing optimization.

- SAME_NONARC_DATA is an option that excludes data from processing.

This option indicates that the warm start nonarc variable data in ARCDATA is read and any nonarc variable data in CONDATA is to be ignored. Use this option if it is applicable, or when CONDATA has no nonarc variable data, or such data is duplicated in ARCDATA. ARCDATA is always read before CONDATA.

Arcs and nonarc variables can have associated with them values or quantities that have no bearing with the optimization. This information is given in ARCDATA in the ID list variables. For example, in a distribution problem, information such as truck number and driver's name can be associated with each arc. This is useful when a solution is saved in an output SAS data set. However, PROC NETFLOW needs to reserve memory to process this information when data is being read. For large problems when memory is scarse, it might be better to remove ancilliary data from ARCDATA. After PROC NETFLOW runs, use SAS software to merge this information into the output data sets that contain the optimal solution.

Macro Variable _ORNETFL

The NETFLOW procedure always creates and initializes a SAS macro called _ORNETFL . After each PROC NETFLOW run, you can examine this macro by specifying %put _ORNETFL; and see whether PROC NETFLOW ran correctly or what error or difficulty it encountered.

The value of _ORNETFL consists of four parts:

- ERROR_STATUS, indicating the existence or absence of any errors
- OPT_STATUS, the stage of the optimization, or what solution has been found
- OBJECTIVE=$objective$ - the total cost or profit of the current solution. If PROC NETFLOW is solving a maximal flow problem, MAXFLOW=$maxflow$, the amount of the current solution's maximal flow,

will follow. If solving a minimal flow problem (MAXFLOW and MAXIMIZE specified at the same time), MINFLOW=$minflow$, the amount of the current solution's mimimal flow, will follow instead

- SOLUTION, describing the nature of the current solution

The value of _ORNETFL is in the form

ERROR_STATUS=$charstr$ OPT_STATUS=$charstr$ OBJECTIVE=$objective$ SOLUTION=$charstr$.

Nontrailing blank characters that are unnecessary are removed. Ideally, at the end a PROC NETFLOW run, _ORNETFL would have the value:

```
ERROR_STATUS=OK OPT_STATUS=OPTIMAL OBJECTIVE=x
SOLUTION=OPTIMAL
```

Table 4.23 has alternate values for the _ORNETFL value parts:

Table 4.23. PROC NETFLOW _ORNETFL macro values

Key word	Value	Meaning
ERROR_STATUS	OK	no errors
	MEMORY	memory request failed
	IO	input-output error
	DATA	error in the data
	BUG	error with PROC NETFLOW
	SEMANTIC	semantic error
	SYNTAX	syntax error
	UNKNOWN	unknown error
OPT_STATUS	START	no optimization has been done
	STAGE_1	performing stage 1 optimization
	UNCON_OPT	reached unconstrained optimum, but there are side constraints
	STAGE_2	performing stage 2 optimization
	OPTIMAL	reached the optimum
OBJECTIVE	$objective$	total cost or profit
MINFLOW	$minflow$	if MAXFLOW MAXIMIZE specified
MAXFLOW	$maxflow$	if MAXFLOW specified
SOLUTION	NONOPTIMAL	more optimization is required
	STAGE_2_REQUIRED	reached unconstrained optimum, stage 2 optimization is required.
	OPTIMAL	have determined the optimum
	INFEASIBLE	infeasible. No solution exists.

The Interior Point Algorithm

Introduction

The Simplex algorithm, developed shortly after World War II, was the main method used to solve Linear Programming problems. Over the last decade, the Interior Point algorithm has been developed to also solve Linear Programming problems. From the start it showed great theoretical promise, and considerable research in the area resulted in practical implementations that performed competitively with the Simplex algorithm. More recently, Interior Point algorithms have evolved to become superior to the Simplex algorithm, in general, especially when the problems are large.

The Interior Point algorithm has been implemented in PROC NETFLOW. This algorithm can be used to solve Linear Programs, as well as network problems. When PROC NETFLOW detects that the problem has no network component, it automatically envokes the Interior Point aglorithm to solve the problem. The data required by PROC NETFLOW for a Linear Program resembles the data for nonarc variables and constraints for constrained network problems.

If PROC NETFLOW does detect a network component to the problem, (the problem has arcs), you must specify the option INTPOINT in the PROC NETFLOW statement if you want to use the Interior Point algorithm. PROC NETFLOW first converts the constrained network model into an equivalent Linear Programming formulation, solves that, then converts the LP back to the network model. These models remain conceptionally easy since they are based on network diagrams that represent the problem pictorially. This procedure accepts the network specification in a format that is particularly suited to networks. This not only simplifies problem description but also aids in the interpretation of the solution. The conversion to and from the equivalent LP is done "behind the scenes".

There are many variations of Interior Point algorithms. PROC NETFLOW uses the Primal-Dual with Predictor-Corrector algorithm. This algorithm and related theory can be found in the texts by Roos, Terlaky, and Vial 1997, Wright 1996, and Ye 1996.

The remainder of this section is split into two parts. In the first part, how you use PROC NETFLOW's Interior Point algorithm to solve network problems is described. In the second part, using PROC NETFLOW to solve Linear Programming problems (it's Interior Point algorithm must be used) is described. Both parts are organized similarly:

- The way data is supplied to PROC NETFLOW is outlined in a "Getting Started" subsection.
- An "Introductory Example" is solved to demonstrate how the data is set up, how PROC NETFLOW is used to do the solution and the optimum saved.
- More sophisticated ways to use PROC NETFLOW interactively are detailed in a "Iteractively" subsection.

- A “Functional Summary” lists the statements and options that can be used to control PROC NETFLOW. Of particular interest are the options used to control the optimizer, and the way the solution is saved into output data sets or is displayed.

The part for Linear Programs has additional subsections:

- “Mathematical Description of LP”
- “Interior Point Algorithmic Details”, a brief theory of the algorithm containing information about the options that can be specified to control the Interior Point algorithm.
- “Syntax” subsection, which is a subset of the syntax when the Simplex algorithm is used. Gone are the statements and lists relevant only when the Simplex algorithm is used.

Network Models: Interior Point Algorithm

The data required by PROC NETFLOW for a network problem is *identical* whether the Simplex algorithm or the Interior Point algorithm is used as the optimizer. By default, the Simplex algorithm is used for problems with a network component. To use the Interior Point algorithm, all you need to do is specify the INTPOINT option in the PROC NETFLOW statement. You can optionally specify some options that control the Interior Point algorithm, of which there are only a few. The Interior Point algorithm is remarkedly robust when reasonable choices are made during the design and implementation, so it does not need to be tuned to the same extent as the Simplex algorithm.

When to Use INTPOINT: Network Models: Interior Point algorithm

PROC NETFLOW uses the Primal Simplex Network algorithm and the Primal Partitioning Algorithm to solve constrained network problems. These algorithms are fast, since they take advantage of algebraic properties of the network component of the problem.

If the network component of the model is large compared to the side constraint component, PROC NETFLOW’s optimizer can store what would otherwise be a large matrix as a spanning tree computer data structure. Computations involving the spanning tree data structure can be performed much faster than those using matrices. Only the nonnetwork part of the problem, hopefully quite small, needs to be manipulated by PROC NETFLOW as matrices.

In contrast, LP optimizers must contend with matrices that can be large for large problems. Arithmetic operations on matrices often accumulate rounding errors that cause difficulties for the algorithm. So in addition to the performance improvements, network optimization is generally more numerically stable than LP optimization.

The nodal flow conservation constraints do not need to be specified in the network model. They are implied by the network structure. However, flow conservation constraints do make up the data for the equivalent LP model. If you have an LP that is small after the flow conservation constraints are removed, that problem is a definite candidate for solution by PROC NETFLOW specialized Simplex method.

However, some constrained network problems are solved more quickly by the Interior Point algorithm than the network optimizer in PROC NETFLOW. Usually, they have a large number of side constraints or nonarc variables. These models are more like LPs than network problems. The network component of the problem is so small that PROC NETFLOW's Network Simplex method cannot recoup the effort to exploit that component rather than treat the whole problem as an LP. If this is the case, it is worthwhile to get PROC NETFLOW to convert a constrained network problem to the equivalent LP and use it's Interior Point algorithm. This conversion must be done before any optimization has been performed (specify the INTPOINT option in the PROC NETFLOW statement)

Even though some network problems are better solved by converting them to an LP, the input data and the output solution are more conveniently maintained as networks. You retain the advantages of casting problems as networks: ease of problem generation and expansion when more detail is required. The model and optimal solutions are easy to understand, as a network can be drawn.

Getting Started: Network Models: Interior Point algorithm

To solve network programming problems with side constraints using PROC NETFLOW, you save a representation of the network and the side constraints in three SAS data sets. These data sets are then passed to PROC NETFLOW for solution. There are various forms that a problem's data can take. You can use any one or a combination of several of these forms.

The NODEDATA= data set contains the names of the supply and demand nodes and the supply or demand associated with each. These are the elements in the column vector b in problem (NPSC).

The ARCDATA= data set contains information about the variables of the problem. Usually these are arcs, but there can be data related to nonarc variables in the ARCDATA= data set as well. If there are no arcs, this is a Linear Programming problem.

An arc is identified by the names of its tail node (where it originates) and head node (where it is directed). Each observation can be used to identify an arc in the network and, optionally, the cost per flow unit across the arc, the arc's capacity, lower flow bound, and name. These data are associated with the matrix F and the vectors c, l, and u in problem (LPSC).

Note: although F is a node-arc incidence matrix, it is specified in the ARCDATA= data set by arc definitions. Do not explicitly specify these flow conservation constraints as constraints of the problem.

In addition, the ARCDATA= data set can be used to specify information about nonarc variables, including objective function coefficients, lower and upper value bounds, and names. These data are the elements of the vectors d, m, and v in problem (NPSC). Data for an arc or nonarc variable can be given in more than one observation.

Supply and demand data also can be specified in the ARCDATA= data set. In such a case, the NODEDATA= data set may not be needed.

The CONDATA= data set describes the side constraints and their right-hand-sides. These data are elements of the matrices H and Q and the vector r. Constraint types are also specified in the CONDATA= data set. You can include in this data set upper bound values or capacities, lower flow or value bounds, and costs or objective function coefficients. It is possible to give all information about some or all nonarc variables in the CONDATA= data set.

An arc or nonarc variable is identified in this data set by it's name. If you specify an arc's name in the ARCDATA= data set, then this name is used to associate data in the CONDATA= data set with that arc. Each arc also has a default name that is the name of the tail and head node of the arc concatenated together and separated by an underscore character; *tail_head*, for example.

If you use the dense side constraint input format and want to use these default arc names, these arc names are names of SAS variables in the VAR list of the CONDATA= data set.

If you use the sparse side constraint input format (described later as well) and want to use these default arc names, these arc names are values of the COLUMN list SAS variable of the CONDATA= data set.

When using the Interior Point algorithm, the execution of PROC NETFLOW has two stages. In the preliminary (zeroth) stage, the data are read from the NODEDATA= data set, the ARCDATA= data set, and the CONDATA= data set. Error checking is performed. The model is converted into an equivalent Linear Program

In the next stage, the Linear Program is preprocessed. This is optional but highly recommended. Preprocessing analyses the model and tries to determine before optimization whether variables can be "fixed" to their optimal values. Knowing that, the model can be modified and these variables dropped out. It can be determined that some constraints are redundant. Sometimes, preprocessing succeeds in reducing the size of the problem, thereby making the subsequent optimization easier and faster.

The optimal solution to the Linear Program is then found. The Linear Program is converted back to the original constrained network problem, and the optimum for this is derived from the optimum of the equivalent Linear Program. If the problem was preprocessed, the model is now post-processed, where fixed variables are reintroduced. The solution can be saved in the CONOUT= data set. This data set is also named in the PROC NETFLOW, RESET, and SAVE statements.

The Interior Point algorithm cannot efficiently be warm started, so options such as FUTURE1 and FUTURE2 options are irrelevant.

Introductory Example: Network Models: Interior Point Algorithm

Consider the following transshipment problem for an oil company in the "Introductory Example" section on page 197. Recall that crude oil is shipped to refineries where it is processed into gasoline and diesel fuel. The gasoline and diesel fuel are then distributed to service stations. At each stage there are shipping, processing, and distribution costs. Also, there are lower flow bounds and capacities. In addition, there are side constraints to model crude mix stipulations, and model the limitations on the amount of Middle Eastern crude that can be processed by each refinery and the conversion proportions of crude to gasoline and diesel fuel. The network diagram is reproduced in Figure 4.16.

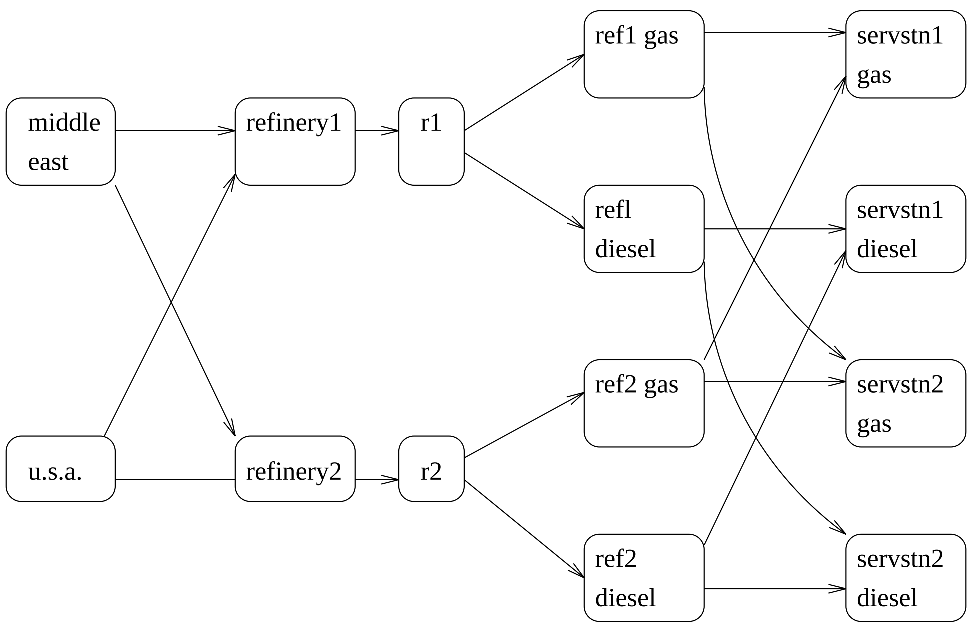


Figure 4.16. Oil Industry Example

To solve this problem with PROC NETFLOW, a representation of the model is saved in three SAS data sets, that are identical to the data sets supplied to PROC NETFLOW when the Simplex algorithm was used.

To find the minimum cost flow through the network that satisfies the supplies, demands, and side constraints, invoke PROC NETFLOW as follows:

```
proc netflow
   intpoint              /* <<<--- Interior Point used */
   nodedata=noded        /* the supply and demand data */
   arcdata=arcd1         /* the arc descriptions       */
   condata=cond1         /* the side constraints       */
   conout=solution;      /* the solution data set      */
run;
```

The following messages, that appear on the SAS log, summarize the model as read by PROC NETFLOW and note the progress toward a solution:

```
NOTE: Number of nodes= 14 .
NOTE: Number of supply nodes= 2 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 180 , total demand= 180 .
NOTE: Number of arcs= 18 .
NOTE: Number of variables= 18 .
NOTE: Number of <= constraints= 0 .
NOTE: Number of == constraints= 16 .
NOTE: Number of >= constraints= 2 .
NOTE: Number of constraint coefficients= 44 .
NOTE: After preprocessing, number of <= constraints= 0.
NOTE: After preprocessing, number of == constraints= 6.
NOTE: After preprocessing, number of >= constraints= 2.
NOTE: The preprocessor eliminated 10 constraints from
      the problem.
NOTE: The preprocessor eliminated 22 constraint
      coefficients from the problem.
NOTE: After preprocessing, number of variables= 8.
NOTE: The preprocessor eliminated 10 variables from the
      problem.
NOTE: 2 columns, 0 rows and 2 coefficients were added to
      the problem to handle unrestricted variables,
      variables that are split, and constraint slack or
      surplus variables.
NOTE: There are 18 nonzero elements in A * A transpose.
NOTE: Number of fill-ins=5.
NOTE: Of the 8 rows and columns, 2 are sparse.
NOTE: During factorization, 1 of the dense rows were found
      to be completely dense and were treated as such from
      then on. This should have saved time.
NOTE: There are 8 nonzero elements in the sparse rows of the
      factored A * A transpose. This includes fill-ins in the
      sparse rows.
NOTE: There are 3 operations of the form
      u[i,j]=u[i,j]-u[q,j]*u[q,i]/u[q,q]
      to factorize the sparse rows of A * A transpose.
NOTE: Constraint feasibility, bound feasibility, and dual
      feasibility attained by iteration 1 as a affine step
      (mu=0) of length 1 was done.
NOTE: Primal-Dual Predictor-Corrector Interior point algorithm
      performed 8 iterations.
NOTE: Objective = 50875.000253.
NOTE: The data set WORK.SOLUTION has 18 observations and
      14 variables.
```

The first set of messages provide statistics on the size of the equivalent Linear Programming problem. The number of variables may not equal the number of arcs if the problem has nonarc variables. This example has none. To convert a network to an equivalent LP problem, a flow conservation constraint must be created for each node (including an excess or bypass node, if required). This explains why the number of equality side constraints and the number of constraint coefficients change when the Interior Point algorithm is used.

If the preprocessor was successful in decreasing the problem size, some messages will report how well it did. In this example, the model size was cut in half!

The following set of messages describe aspects of the Interior Point algorithm. Of particular interest are those concerned with the Cholesky factorization of AA^T where A is the coefficient matrix of the final LP. It is crucial to preorder the rows and columns of this matrix to prevent *fill-in* and reduce the number of row operations to undertake the factorization. See the “Interior Point Algorithmic Details” section on page 316 for more explanation.

Unlike PROC LP, which displays the solution and other information as output, PROC NETFLOW saves the optimum in output SAS data sets you specify. For this example, the solution is saved in the SOLUTION data set. It can be displayed with PROC PRINT as

```
proc print data=solution;
   var _from_ _to_ _cost_ _capac_ _lo_ _name_
       _supply_ _demand_ _flow_ _fcost_ _rcost_;
   sum _fcost_;
   title3 'Constrained Optimum'; run;
```

Constrained Optimum

Obs	_from_	_to_	_cost_	_capac_	_lo_	_name_
1	refinery 1	r1	200	175	50	thruput1
2	refinery 2	r2	220	100	35	thruput2
3	r1	ref1 diesel	0	75	0	
4	r1	ref1 gas	0	140	0	r1_gas
5	r2	ref2 diesel	0	75	0	
6	r2	ref2 gas	0	100	0	r2_gas
7	middle east	refinery 1	63	95	20	m_e_ref1
8	u.s.a.	refinery 1	55	99999999	0	
9	middle east	refinery 2	81	80	10	m_e_ref2
10	u.s.a.	refinery 2	49	99999999	0	
11	ref1 diesel	servstn1 diesel	18	99999999	0	
12	ref2 diesel	servstn1 diesel	36	99999999	0	
13	ref1 gas	servstn1 gas	15	70	0	
14	ref2 gas	servstn1 gas	17	35	5	
15	ref1 diesel	servstn2 diesel	17	99999999	0	
16	ref2 diesel	servstn2 diesel	23	99999999	0	
17	ref1 gas	servstn2 gas	22	60	0	
18	ref2 gas	servstn2 gas	31	99999999	0	

Obs	_SUPPLY_	_DEMAND_	_FLOW_	_FCOST_	_RCOST_
1	.	.	145.00	29000.00	.
2	.	.	35.00	7700.00	29
3	.	.	36.25	0.00	.
4	.	.	108.75	0.00	.
5	.	.	8.75	0.00	.
6	.	.	26.25	0.00	.
7	100	.	80.00	5040.00	.
8	80	.	65.00	3575.00	.
9	100	.	20.00	1620.00	.
10	80	.	15.00	735.00	.
11	.	30	30.00	540.00	.
12	.	30	0.00	0.00	12
13	.	95	68.75	1031.25	.
14	.	95	26.25	446.25	.
15	.	15	6.25	106.25	.
16	.	15	8.75	201.25	.
17	.	40	40.00	880.00	.
18	.	40	0.00	0.00	7
				========	
				50875.00	

Figure 4.17. conout=solution

Notice that, in the solution data set (Figure 4.17), the optimal flow through each arc in the network is given in the variable named _FLOW_, and the cost of flow through each arc is given in the variable _FCOST_. As expected, the miminal total cost of the solution found by the Interior Point algorithm is equal to miminal total cost of the solution found by the Simplex algorithm. In this example, the solutions are the same (within several significant digits), but sometimes the solutions can be different.

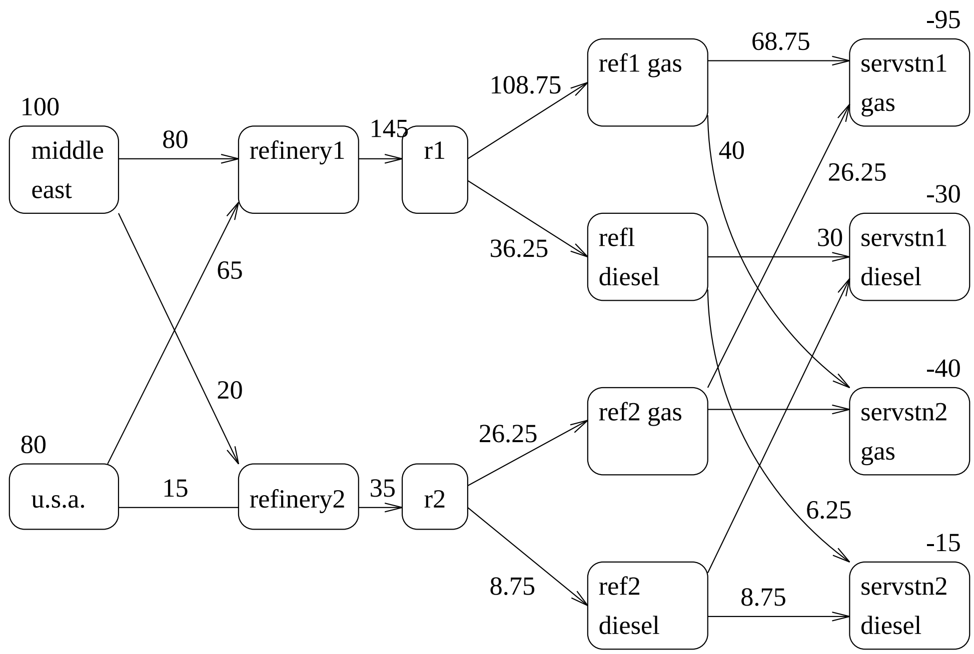


Figure 4.18. Oil Industry solution

Interactivity: Network Models: Interior Point algorithm

PROC NETFLOW can be used interactively. You begin by giving the PROC NETFLOW statement with INTPOINT specified, and you must specify the ARCDATA= data set. The CONDATA= data set must also be specified if the problem has side constraints. If necessary, specify the NODEDATA= data set.

The variable lists should be given next. If you have variables in the input data sets that have special names (for example, a variable in the ARCDATA= data set named _TAIL_ that has tail nodes of arcs as values), it may not be necessary to have many or any variable lists.

So far, this is the same when the Simplex algorithm is used, except the INTPOINT option is specified in the PROC NETFLOW statement. The PRINT, QUIT, SAVE, SHOW, RESET, and RUN statements follow and can be listed in any order. The QUIT statements can be used only once. The others can be used as many times as needed.

The CONOPT and PIVOT statements are not relevant to the Interior Point algorithm and should not be used.

Use the RESET or SAVE statement to change the name of the output data set. There is only on output data set, the CONOUT= data set. With the RESET statement, you can also indicate the reasons why optimization should stop, (for example, you can indicate the maximum number of iterations that can be performed). PROC NETFLOW then has a chance to either execute the next statement, or, if the next statement is one that PROC NETFLOW does not recognize (the next PROC or DATA step in the SAS session), do any allowed optimization and finish. If no new statement has been submitted, you are prompted for one. Some options of the RESET statement enable you to control aspects of the Interior Point algorithm. Specifying certain values for these options can reduce the time it takes to solve a problem. Note that any of the RESET options can be specified in the PROC NETFLOW statement.

The RUN statement starts optimization. Once the optimization has started, it runs until the optimum is reached. The RUN statement should be specified at most once.

The QUIT statement immediately stops PROC NETFLOW. The SAVE statement has options that allow you to name the output data set; information about the current solution is put in this output data set. Use the SHOW statement if you want to examine the values of options of other statements. Information about the amount of optimization that has been done and the STATUS of the current solution can also be displayed using the SHOW statement.

The PRINT statement makes PROC NETFLOW display parts of the problem. The way the PRINT statements are specified are identical whether the Interior Point algorithm or the Simplex algorithm is used, however there are minor differences in what is displayed for each arc, nonarc variable or constraint coefficient.

PRINT ARCS produces information on all arcs. PRINT SOME_ARCS limits this output to a subset of arcs. There are similar PRINT statements for nonarc variables and constraints:

```
PRINT NONARCS;
PRINT SOME_NONARCS;
PRINT CONSTRAINTS;
PRINT SOME_CONS;
```

PRINT CON_ARCS enables you to limit constraint information that is obtained to members of a set of arcs and that have nonzero constraint coefficients in a set of constraints. PRINT CON_NONARCS is the corresponding statement for nonarc variables.

For example, an interactive PROC NETFLOW run might look something like this:

```
proc netflow
       intpoint      /* use the Interior Point algorithm */
       arcdata=data set
       other options;
   variable list specifications;  /* if necessary        */
   reset options;
   print options;                 /* look at problem     */
run;                              /* do the optimization */
   print options;    /* look at the optimal solution     */
   save options;     /* keep optimal solution            */
```

If you are interested only in finding the optimal solution, have used SAS variables that have special names in the input data sets, and want to use default setting for everything, then the following statement is all you need.

proc netflow intpoint arcdata=*data set options* **;**

Functional Summary: Network Models: Interior Point algorithm

The following tables outline the options available for the NETFLOW procedure when the Interior Point algorithm is being used, classified by function.

Table 4.24. Input Data Set Options

Description	Statement	Option
arcs input data set	NETFLOW	ARCDATA=
nodes input data set	NETFLOW	NODEDATA=
constraint input data set	NETFLOW	CONDATA=

Table 4.25. Options for Networks

Description	Statement	Option
default arc cost	NETFLOW	DEFCOST=
default arc capacity	NETFLOW	DEFCAPACITY=
default arc lower flow bound	NETFLOW	DEFMINFLOW=
network's only supply node	NETFLOW	SOURCE=
SOURCE's supply capability	NETFLOW	SUPPLY=
network's only demand node	NETFLOW	SINK=
SINK's demand	NETFLOW	DEMAND=
excess supply or demand is conveyed through network	NETFLOW	THRUNET
find maximal flow between SOURCE and SINK	NETFLOW	MAXFLOW
cost of bypass arc when solving MAXFLOW problem	NETFLOW	BYPASSDIV=
find shortest path from SOURCE to SINK	NETFLOW	SHORTPATH

Table 4.26. Miscellaneous Options

Description	Statement	Option
infinity value	NETFLOW	INFINITY=
do constraint row and/or nonarc variable column coefficient scaling, or neither	NETFLOW	SCALE=
maximization instead of minimization	NETFLOW	MAXIMIZE

Table 4.27. Data Set Read Options

Description	Statement	Option
CONDATA has sparse data format	NETFLOW	SPARSECONDATA
default constraint type	NETFLOW	DEFCONTYPE=
special COLUMN variable value	NETFLOW	TYPEOBS=
special COLUMN variable value	NETFLOW	RHSOBS=
is used to interpret arc and nonarc variable names in the CONDATA	NETFLOW	NAMECTRL=
no new nonarc variables	NETFLOW	SAME_NONARC_DATA
no nonarc data in the ARCDATA	NETFLOW	ARCS_ONLY_ARCDATA
data for an arc found in only one obs of ARCDATA	NETFLOW	ARC_SINGLE_OBS
data for an constraint found in only one obs of CONDATA	NETFLOW	CON_SINGLE_OBS
data for a coefficient found once in CONDATA	NETFLOW	NON_REPLIC=
data is grouped, exploited during data read	NETFLOW	GROUPED=

Table 4.28. Problem Size (approx.) Options

Description	Statement	Option
number of nodes	NETFLOW	NNODES=
number of arcs	NETFLOW	NARCS=
number of nonarc variables	NETFLOW	NNAS=
number of coefficients	NETFLOW	NCOEFS=
number of constraints	NETFLOW	NCONS=

Table 4.29. Memory Control Options

Description	Statement	Option
issue memory usage messages to SASLOG	NETFLOW	MEMREP
number of bytes to use for main memory	NETFLOW	BYTES=
proportion of memory used by frequently accessed arrays	NETFLOW	COREFACTOR=
maximum bytes for a single array	NETFLOW	MAXARRAYBYTES=

Table 4.30. Output Data Set Options: RESET

Description	Statement	Option
constrained solution data set	RESET	CONOUT=

Table 4.31. PRINT Statement Options

Description	Statement	Option
display everything	PRINT	PROBLEM
display arc information	PRINT	ARCS
display nonarc variable information	PRINT	NONARCS
display variables information	PRINT	VARIABLES
display constraint information	PRINT	CONSTRAINTS
display information for some arcs	PRINT	SOME_ARCS
display information for some nonarc variables	PRINT	SOME_NONARCS
display information for some variables	PRINT	SOME_VARIABLES
display information for some constraints	PRINT	SOME_CONS
display information for some constraints associated with some arcs	PRINT	CON_ARCS
display information for some constraints associated with some nonarc variables	PRINT	CON_NONARCS
display information for some constraints associated with some variables	PRINT	CON_VARIABLES

Table 4.32. PRINT Statement Qualifiers

Description	Statement	Option
produce a short report	PRINT	/ SHORT
produce a long report	PRINT	/ LONG
only arcs (nonarc variables) with zero flow (value)	PRINT	/ ZERO
only arcs (nonarc variables) with nonzero flow (value)	PRINT	/ NONZERO

Table 4.33. SHOW Statement Options

Description	Statement	Option
show problem, optimization status	SHOW	STATUS
show network model parameters	SHOW	NETSTMT
show data sets that have, will be created	SHOW	DATA SETS

Table 4.34. Output Data Set Options: SAVE

Description	Statement	Option
constrained solution data set	SAVE	CONOUT=

Table 4.35. Interior Point algorithm Options

Description	Statement	Option
use Interior Point algorithm	NETFLOW	INTPOINT
allowed amount of dual infeasibility	RESET	TOLDINF=
allowed amount of primal infeasibility	RESET	TOLPINF=
cut-off tolerance for Cholesky factorization	RESET	CHOLTINYTOL=
density threshold for Cholesky processing	RESET	DENSETHR=
maximum number of Interior Point algorithm iterations	RESET	MAXITERB=
Primal-Dual (Duality) gap tolerance	RESET	PDGAPTOL=
step-length multiplier	RESET	PDSTEPMULT=
preprocessing type	RESET	PRSLTYPE=

Linear Programming Models: Interior Point Algorithm

By default, the Interior Point algorithm is used for problems without a network component, that is, a Linear Programming problem. You do not need to specify the INTPOINT option in the PROC NETFLOW statement (although you will do no harm if you do).

Data for a linear programming problem resembles the data for side constraints and nonarc variables supplied to PROC NETFLOW when solving a constrained network problem. It is also very similar to the data required by the LP procedure.

Mathematical Description of LP

If the network component of NPSC is removed, the result is the mathematical description of the Linear Programming problem. If an LP has g variables, and k constraints, then the formal statement of the problem solved by PROC NETFLOW is

$$
\begin{array}{ll}
 & min\{d^T z\} \\
\text{subject to} & Qz \geq, =, \leq r \\
 & m \leq z \leq v
\end{array}
$$

where

d is the g x 1 objective function coefficient of variables vector.

z is the g x 1 variable value vector.

Q is the k x g constraint coefficient matrix for variables, where $Q_{i,j}$ is the coefficient of variable j in the ith constraint.

r is the k x 1 side constraint right-hand-side vector.

m is the g x 1 variable value lower bound vector.

v is the g x 1 variable value upper bound vector.

Interior Point Algorithmic Details

After preprocessing, the Linear Program to be solved is

$$min\{c^T x\}$$

subject to $\quad Ax = b$

$$x \geq 0$$

This is the *primal* problem. The matrices of d z Q of NP have been renamed c x and A respectively, as these symbols are by convention used more, the problem to be solved is different from the original because of preprocessing, and there has been a change of primal variable to transform the LP into one whose variables have zero lower bounds. To simplify the algebra here, assume that variables have infinite bounds, and constraints are equalities. (Interior Point algorithms do efficiently handle finite bounds, and it is easy to introduce primal slack variables to change inequalities into equalities.) The problem has n variables. i is a variable number. k is an iteration number, and if used as a subscript or superscript denotes "of iteration k".

There exists an equivalent problem, the *dual* problem, stated as

$$max\{b^T y\}$$

subject to $\quad A^T y + s = c$

$$s \geq 0$$

where $\quad$ y are dual variables, and s are dual constraint slacks

What the Interior Point has to do is solve the system of equations to satisfy the Karush-Kuhn-Tucker (KKT) conditions for optimality:

$$Ax = b$$

$$A^T y + s = c$$

$$x^T s = 0$$

$$x \geq 0$$

$$s \geq 0$$

These are the conditions for feasibility, with the *complementarity* condition $x^T s = 0$ added. $c^T x = b^T y$ must occur at the optimum. Complementarity forces the optimal objectives of the primal and dual to be equal, $c^T x_{opt} = b^T y_{opt}$, as

$$0 = x_{opt}^T s_{opt} = s_{opt}^T x_{opt} = (c - A^T y_{opt})^T x_{opt} =$$
$$c^T x_{opt} - y_{opt}^T (A x_{opt}) = c^T x_{opt} - b^T y_{opt}$$
$$0 = c^T x_{opt} - b^T y_{opt}$$

Before the optimum is reached, a solution (x, y, s) may not satisfy the KKT conditions.

- Primal constraints can be broken, $infeas_b = b - Ax \neq 0$.
- Dual constraints can be broken, $infeas_c = c - A^T y - s \neq 0$.
- Complementarity is unsatisfied, $x^T s = c^T x - b^T y \neq 0$. This is called the *duality gap*.

Interior Point algorithm works by using Newtons method to find a direction to move $(\Delta x^k, \Delta y^k, \Delta s^k)$ from the current solution (x^k, y^k, s^k) toward a better solution.

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \alpha(\Delta x^k, \Delta y^k, \Delta s^k)$$

α is the *step length* and is assigned a value as large as possible but ≤ 1.0 and not so large that a x_i^{k+1} or s_i^{k+1} is "too close" to zero. The direction in which to move is found using

$$A\Delta x^k = -infeas_b$$
$$A^T \Delta y^k + \Delta s^k = -infeas_c$$
$$S^k \Delta x^k + X^k \Delta s^k = -X^k S^k e$$

where S = diag(s), X = diag(x), and e = vector with all elements = 1

To greatly improve performance, the third equation is changed to

$$S^k \Delta x^k + X^k \Delta s^k = -X^k S^k e + \sigma_k \mu_k e$$

where $\mu_k = 1/n X^k S^k e$, the average complementarity, and

$0 \leq \sigma_k \leq 1$

The effect now is to find a direction in which to move to reduce infeasibilities and to reduce the complementarity toward zero, but if any $x_i^k s_i^k$ is too close to zero, it is "nudged out" to μ, any $x_i^k s_i^k$ that is larger than μ is "nudged into" μ. A σ_k close to or equal to 0.0 biases a direction toward the optimum, and a value for σ_k close to or equal to 1.0 "centers" the direction toward a point where all pairwise products $x_i^k s_i^k = \mu$. Such points make up the *Central Path* in the interior. Although centering directions make little, if any, progress in reducing μ and moving the solution closer to the optimum, substantial progress toward the optimum can usually be made in the next iteration.

The Central Path is crucial to why the Interior Point algorithm is so efficient. This path "guides" the algorithm to the optimum through the interior of feasible space.

Without centering, the algorithm would find a series of solutions near each other close to the boundary of feasible space. Step lengths along the direction would be small and many more iterations would probably be required to reach the optimum.

That in a nutshell is the Primal-Dual Interior Point algorithm. Varieties of the algorithm differ in the way α and σ_k are chosen and the direction adjusted each iteration. A wealth of information can be found in the texts by Roos, Terlaky, and Vial 1997, Wright 1996, and Ye 1996.

The calculation of the direction is the most time-consuming step of the Interior Point algorithm. Assume the kth iteration is being performed, so the subscript and superscript k can be dropped from the algebra.

$$A\Delta x = -infeas_b$$

$$A^T\Delta y + \Delta s = -infeas_c$$

$$S\Delta x + X\Delta s = -XSe + \sigma\mu e$$

Rearranging the second equation

$$\Delta s = -infeas_c - A^T\Delta y$$

Rearranging the third equation

$$\Delta s = X^{-1}(-S\Delta x - XSe + \sigma\mu e)$$

$$\Delta s = -\Theta\Delta x - Se + X^{-1}\sigma\mu e$$

where $\Theta = SX^{-1}$

Equating these two expressions for Δs and rearranging

$$-\Theta\Delta x - Se + X^{-1}\sigma\mu e = -infeas_c - A^T\Delta y$$

$$-\Theta\Delta x = Se - X^{-1}\sigma\mu e - infeas_c - A^T\Delta y$$

$$\Delta x = \Theta^{-1}(-Se + X^{-1}\sigma\mu e + infeas_c + A^T\Delta y)$$

$$\Delta x = \rho + \Theta^{-1}A^T\Delta y$$

where $\rho = \Theta^{-1}(-Se + X^{-1}\sigma\mu e + infeas_c)$

Substituting into the first direction equation

$$A\Delta x = -infeas_b$$

$$A(\rho + \Theta^{-1}A^T\Delta y) = -infeas_b$$

$$A\Theta^{-1}A^T\Delta y = -infeas_b - A\rho$$

$$\Delta y = (A\Theta^{-1}A^T)^{-1}(-infeas_b - A\rho)$$

Θ, ρ, Δy, Δx and Δs are calculated in that order. The hardest term is the factorization of the $(A\Theta^{-1}A^T)$ matrix to determine Δy. Fortunately, although the *values* of $(A\Theta^{-1}A^T)$ is different each iteration, the *locations* of the nonzeroes in this matrix remain fixed; the nonzero locations is the same as those in the matrix (AA^T). This is due to $\Theta^{-1} = XS^{-1}$ being a diagonal matrix that has the effect of merely scaling the columns of (AA^T).

The fact that the nonzeroes in $A\Theta^{-1}A^T$ has a constant pattern is exploited by all Interior Point algorithms, and is a major reason for their excellent performance. Before iterations begin, AA^T is examined and it's rows and columns permutated so that during Cholesky Factorization, the number of *fillins* created is smaller. A list of arithmetic operations to perform the factorization is saved in concise computer data structures (working with memory locations rather than actual numerical values). This is called *symbolic factorization*. During iterations, when memory has been initialized with numerical values, the operations list is performed sequentially. Determining how the factorization should be performed again and again is unnecessary.

Getting Started: Linear Programming Models: Interior Point algorithm

To solve linear programming problem using PROC NETFLOW, you save a representation of the variables and the constraints in one or two SAS data sets. These data sets are then passed to PROC NETFLOW for solution. There are various forms that a problem's data can take. You can use any one or a combination of several of these forms.

The ARCDATA= data set contains information about the variables of the problem. Although this data set is called ARCDATA, it contains data for no arcs. Instead, all data in this data set are related to variables.

The ARCDATA= data set can be used to specify information about variables, including objective function coefficients, lower and upper value bounds, and names. These data are the elements of the vectors d, m, and v in problem (NP). Data for an variable can be given in more than one observation.

When the data for a constrained network problem is being provided, the ARCDATA= data set always contains information necessary for arcs, their tail and head nodes, and optionally the supply and demand information of these nodes. When the data for a linear programming problem is being provided, none of this information is present, as the model has no arcs. This is the way PROC NETFLOW decides which type of problem it is to solve.

PROC NETFLOW was originally designed to solve models with networks, so an ARCDATA= data set is always expected. If an ARCDATA= data set is not specified, by default the last data set created before PROC NETFLOW is envoked is assumed to be an ARCDATA= data set. However, these characteristics of PROC NETFLOW are not helpful when a Linear Programming problem is being solved and all data is provided in a single data set specified by the CONDATA= data set, and that data set is not the last data set created before PROC NETFLOW starts. In this case, you must specify that an ARCDATA= data set and a CONDATA= data set are both equal to the input data set. PROC NETFLOW then knows that a Linear Programming problem is to be solved, and the data reside in one data set.

The CONDATA= data set describes the constraints and their right-hand-sides. These data are elements of the matrix Q and the vector r.

Constraint types are also specified in the CONDATA= data set. You can include in this data set variable data such as upper bound values, lower value bounds, and objective function coefficients. It is possible to give all information about some or all variables in the CONDATA= data set.

A variable is identified in this data set by its name. If you specify a variable's name in the ARCDATA= data set, then this name is used to associate data in the CONDATA= data set with that variable.

If you use the dense constraint input format, these variable names are names of SAS variables in the VAR list of the CONDATA= data set.

If you use the sparse constraint input format, these variable names are values of the COLUMN list SAS variable of CONDATA= data set.

When using the Interior Point algorithm, the execution of PROC NETFLOW has two stages. In the preliminary (zeroth) stage, the data are read from the ARCDATA= data set (if used) and the CONDATA= data set. Error checking is performed. In the next stage, the Linear Program is preprocessed, then the optimal optimal solution to the Linear Program is found. The solution is saved in the CONOUT= data set. This data set is also named in the PROC NETFLOW, RESET, and SAVE statements.

See the "Getting Started: Network Models: Interior Point algorithm" section on page 305 for a fuller description of the stages of the Interior Point algorithm.

Introductory Example: Linear Programming Models: Interior Point algorithm

Consider the Linear Programming problem in the "An Introductory Example" section on page 58 in the chapter on the LP procedure.

```
data dcon1;
   input _id_ $14.
         a_light a_heavy brega naphthal naphthai
         heatingo jet_1 jet_2
         _type_ $ _rhs_;
   datalines;
profit         -175 -165 -205  0  0  0 300 300 max     .
naphtha_l_conv .035 .030 .045 -1  0  0   0   0  eq     0
naphtha_i_conv .100 .075 .135  0 -1  0   0   0  eq     0
heating_o_conv .390 .300 .430  0  0 -1   0   0  eq     0
recipe_1          0    0    0  0 .3 .7  -1   0  eq     0
recipe_2          0    0    0 .2  0 .8   0  -1  eq     0
available       110  165   80  .  .  .   .   . upperbd .
;
```

To find the minimum cost solution and to examine all or parts of the optimum, you use PRINT statements.

- **print problem/short;** outputs information for all variables and all constraint coefficients. See Figure 4.19.

- **print some_variables(j:)/short;** is information about a set of variables, (in this case, those with names that start with the character string (here, the single character "j" (without the quotes)) preceeding the colon. See Figure 4.20.
- **print some_cons(recipe_1)/short;** is information about a set of constraints (here, that set only has one member, the constraint called recipe_1). See Figure 4.21.
- **print con_variables(_all_,brega)/short;** lists the constraint information for a set of variables (here, that set only has one member, the variable called brega). See Figure 4.22.
- **print con_variables(recipe:,n: jet_1)/short;** coefficient information for those in a set of constraints belonging to a set of variables. See Figure 4.23.

```
proc netflow
   condata=dcon1
   conout=solutn1;
run;
print problem/short;
print some_variables(j:)/short;
print some_cons(recipe_1)/short;
print con_variables(_all_,brega)/short;
print con_variables(recipe:,n: jet_1)/short;
```

The following messages, that appear on the SAS log, summarize the model as read by PROC NETFLOW and note the progress toward a solution:

```
NOTE: Number of variables= 8 .
NOTE: Number of <= constraints= 0 .
NOTE: Number of == constraints= 5 .
NOTE: Number of >= constraints= 0 .
NOTE: Number of constraint coefficients= 18 .
NOTE: 0 columns, 0 rows and 0 coefficients were added
      to the problem to handle unrestricted variables,
      variables that are split, and constraint slack or
      surplus variables.
NOTE: There are 8 nonzero elements in A * A transpose.
NOTE: Number of fill-ins=0.
NOTE: Of the 5 rows and columns, 0 are sparse.
NOTE: There are 0 nonzero elements in the sparse rows of
      the factored A * A transpose. This includes fill-ins
      in the sparse rows.
NOTE: There are 0 operations of the form
      u[i,j]=u[i,j]-u[q,j]*u[q,i]/u[q,q] to factorize the
      sparse rows of A * A transpose.
NOTE: Constraint feasibility attained by iteration 3.
NOTE: Bound feasibility attained by iteration 3.
NOTE: Dual feasibility attained by iteration 3.
NOTE: Primal-Dual Predictor-Corrector Interior point
      algorithm performed 8 iterations.
NOTE: Objective = 1544.0000121.
NOTE: The data set WORK.SOLUTN1 has 8 observations and
      6 variables.
```

NETFLOW PROCEDURE

_N_	_NAME_	_OBJFN_	_UPPERBD	_LOWERBD	_VALUE_
1	a_heavy	-165	165	0	5.583E-7
2	a_light	-175	110	0	110
3	brega	-205	80	0	80
4	heatingo	0	99999999	0	77.3
5	jet_1	300	99999999	0	60.65
6	jet_2	300	99999999	0	63.33
7	naphthai	0	99999999	0	21.8
8	naphthal	0	99999999	0	7.45

NETFLOW PROCEDURE

_N_	_id_	_type_	_rhs_	_NAME_	_COST_	_CAPAC_
1	heating_o_conv	EQ	0	a_heavy	-165	165
2	heating_o_conv	EQ	0	a_light	-175	110
3	heating_o_conv	EQ	0	brega	-205	80
4	heating_o_conv	EQ	0	heatingo	0	99999999
5	naphtha_i_conv	EQ	0	a_heavy	-165	165
6	naphtha_i_conv	EQ	0	a_light	-175	110
7	naphtha_i_conv	EQ	0	brega	-205	80
8	naphtha_i_conv	EQ	0	naphthai	0	99999999
9	naphtha_l_conv	EQ	0	a_heavy	-165	165
10	naphtha_l_conv	EQ	0	a_light	-175	110
11	naphtha_l_conv	EQ	0	brega	-205	80
12	naphtha_l_conv	EQ	0	naphthal	0	99999999
13	recipe_1	EQ	0	heatingo	0	99999999
14	recipe_1	EQ	0	jet_1	300	99999999
15	recipe_1	EQ	0	naphthai	0	99999999
16	recipe_2	EQ	0	heatingo	0	99999999
17	recipe_2	EQ	0	jet_2	300	99999999
18	recipe_2	EQ	0	naphthal	0	99999999

_N_	_LO_	_VALUE_	_COEF_
1	0	5.583E-7	0.3
2	0	110	0.39
3	0	80	0.43
4	0	77.3	-1
5	0	5.583E-7	0.075
6	0	110	0.1
7	0	80	0.135
8	0	21.8	-1
9	0	5.583E-7	0.03
10	0	110	0.035
11	0	80	0.045
12	0	7.45	-1
13	0	77.3	0.7
14	0	60.65	-1
15	0	21.8	0.3
16	0	77.3	0.8
17	0	63.33	-1
18	0	7.45	0.2

Figure 4.19. print problem/short;

NETFLOW PROCEDURE

_N_	_NAME_	_COST_	_CAPAC_	_LO_	_VALUE_
1	jet_1	300	99999999	0	60.65
2	jet_2	300	99999999	0	63.33

Figure 4.20. print some_variables(j:)/short;

NETFLOW PROCEDURE

_N_	_id_	_type_	_rhs_	_NAME_	_COST_	_CAPAC_
1	recipe_1	EQ	0	heatingo	0	99999999
2	recipe_1	EQ	0	jet_1	300	99999999
3	recipe_1	EQ	0	naphthai	0	99999999

_N_	_LO_	_VALUE_	_COEF_
1	0	77.3	0.7
2	0	60.65	-1
3	0	21.8	0.3

Figure 4.21. print some_cons(recipe_1)/short;

NETFLOW PROCEDURE

_N_	_id_	_type_	_rhs_	_NAME_	_COST_	_CAPAC_
1	heating_o_conv	EQ	0	brega	-205	80
2	naphtha_i_conv	EQ	0	brega	-205	80
3	naphtha_l_conv	EQ	0	brega	-205	80

_N_	_LO_	_VALUE_	_COEF_
1	0	80	0.43
2	0	80	0.135
3	0	80	0.045

Figure 4.22. print con_variables(_all_,brega)/short;

NETFLOW PROCEDURE

_N_	_id_	_type_	_rhs_	_NAME_	_COST_	_CAPAC_
1	recipe_1	EQ	0	jet_1	300	99999999
2	recipe_1	EQ	0	naphthai	0	99999999
3	recipe_2	EQ	0	naphthal	0	99999999

_N_	_LO_	_VALUE_	_COEF_
1	0	60.65	-1
2	0	21.8	0.3
3	0	7.45	0.2

Figure 4.23. print con_variables(recipe:,n: jet_1)/short;

Unlike PROC LP, that displays the solution and other information as output, PROC NETFLOW saves the optimum in output SAS data sets you specify. For this example, the solution is saved in the SOLUTION data set. It can be displayed with PROC PRINT as

```
proc print data=solutn1;
   var _name_ _cost_ _capac_ _lo_ _flow_ _fcost_;
   sum _fcost_;
   title3 'LP Optimum'; run;
```

Notice, in the CONOUT=SOLUTION (Figure 4.24), the optimal value through each variable in the Linear program is given in the variable named _FLOW_, and the cost of value for each variable is given in the variable _FCOST_.

```
                              LP Optimum

   Obs     _NAME_      _COST_      _CAPAC_     _LO_      _FLOW_       _FCOST_

     1     a_heavy      -165          165       0        0.000        -0.00
     2     a_light      -175          110       0      110.000    -19250.00
     3     brega        -205           80       0       80.000    -16400.00
     4     heatingo        0     99999999       0       77.300         0.00
     5     jet_1         300     99999999       0       60.650     18195.00
     6     jet_2         300     99999999       0       63.330     18999.00
     7     naphthai        0     99999999       0       21.800         0.00
     8     naphthal        0     99999999       0        7.450         0.00
                                                                  =========
                                                                    1544.00
```

Figure 4.24. CONOUT=SOLUTN1

The same model can be specified in the sparse format as in the following scon2 dataset. This format enables you to omit the zero coefficients.

```
data scon2;
   input _type_ $ @10 _col_ $13. @24 _row_ $16. _coef_;
   datalines;
max      .             profit                      .
eq       .             napha_l_conv                .
eq       .             napha_i_conv                .
eq       .             heating_oil_conv            .
eq       .             recipe_1                    .
eq       .             recipe_2                    .
upperbd  .             available                   .
.        a_light       profit                   -175
.        a_light       napha_l_conv             .035
.        a_light       napha_i_conv             .100
.        a_light       heating_oil_conv         .390
.        a_light       available                 110
.        a_heavy       profit                   -165
.        a_heavy       napha_l_conv             .030
.        a_heavy       napha_i_conv             .075
.        a_heavy       heating_oil_conv         .300
.        a_heavy       available                 165
.        brega         profit                   -205
.        brega         napha_l_conv             .045
.        brega         napha_i_conv             .135
.        brega         heating_oil_conv         .430
.        brega         available                  80
.        naphthal      napha_l_conv               -1
.        naphthal      recipe_2                   .2
.        naphthai      napha_i_conv               -1
.        naphthai      recipe_1                   .3
.        heatingo      heating_oil_conv           -1
.        heatingo      recipe_1                   .7
.        heatingo      recipe_2                   .8
.        jet_1         profit                    300
.        jet_1         recipe_1                   -1
.        jet_2         profit                    300
.        jet_2         recipe_2                   -1
;
```

To find the minimum cost solution, invoke PROC NETFLOW (note the SPARSECONDATA option which must be specified) as follows:

```
proc netflow
   sparsecondata
   condata=scon2
   conout=solutn2;
run;
```

A data set that is used as an ARCDATA= data set can be initialized as follows:

```
data vars3;
   input _name_ $ profit available;
   datalines;
a_heavy  -165 165
a_light  -175 110
brega    -205  80
heatingo    0   .
jet_1     300   .
jet_2     300   .
naphthai    0   .
naphthal    0   .
;
```

The following CONDATA= data set is the original dense format CONDATA= dcon1 data set with the variable information removed. (You could have left some or all of that information in CONDATA as PROC NETFLOW "merges" data, but doing that and checking for consistency uses time.)

```
data dcon3;
   input _id_ $14.
         a_light a_heavy brega naphthal naphthai
         heatingo jet_1 jet_2
         _type_ $ _rhs_;
   datalines;
naphtha_l_conv .035 .030 .045 -1  0  0   0   0  eq     0
naphtha_i_conv .100 .075 .135  0 -1  0   0   0  eq     0
heating_o_conv .390 .300 .430  0  0 -1   0   0  eq     0
recipe_1          0    0    0  0 .3 .7  -1   0  eq     0
recipe_2          0    0    0 .2  0 .8   0  -1  eq     0
;
```

It is important to note that it is now necessary to specify the MAXIMIZE option; otherwise, PROC NETFLOW will optimize to the minimum (which, incidently, has a total objective = -3539.25). You must indicate that the SAS variable profit in the ARCDATA= vars3 data set has values that are objective function coefficients, by specifying the OBJFN statement. The UPPERBD must be specified as the SAS variable available that has as values upper bounds.

```
proc netflow
   maximize             /* ***** necessary ***** */
```

```
      arcdata=vars3
      condata=dcon3
      conout=solutn3;
   objfn profit;
   upperbd available;
   run;
```

The ARCDATA=vars3 data set can become more concise by noting that the model variables heatingo, naphthai, and naphthal have zero objective function coefficients (the default) and default upper bounds, so those observations need not be present.

```
   data vars4;
      input _name_ $ profit available;
      datalines;
   a_heavy  -165 165
   a_light  -175 110
   brega    -205  80
   jet_1     300   .
   jet_2     300   .
   ;
```

The CONDATA=dcon3 data set can become more concise by noting that all the constraints have the same type (eq) and zero (the default) rhs values. This model is a good candidate for using the DEFCONTYPE= options.

The DEFCONTYPE= option can be useful not only when *all* constraints have the same type as is the case here, but also when *most* constraints have the same type or, if when you prefer to change the default type from $\leq$ to = or $\geq$. The essential constraint type data in CONDATA= data set is that which overrides the DEFCONTYPE= type you specified.

```
   data dcon4;
      input _id_ $14.
            a_light a_heavy brega naphthal naphthai
            heatingo jet_1 jet_2;
      datalines;
   naphtha_l_conv .035 .030 .045 -1  0  0   0   0
   naphtha_i_conv .100 .075 .135  0 -1  0   0   0
   heating_o_conv .390 .300 .430  0  0 -1   0   0
   recipe_1          0    0    0  0 .3 .7  -1   0
   recipe_2          0    0    0 .2  0 .8   0  -1
   ;

   proc netflow
      maximize defcontype=eq
      arcdata=vars3
      condata=dcon3
      conout=solutn3;
   objfn profit;
   upperbd available;
   run;
```

Several different ways of using an ARCDATA= data set and a sparse format CONDATA= data set for this Linear Program follows. The following CONDATA= data set is the result of removing the profit and available data from the original sparse format CONDATA=scon2 data set.

```
data scon5;
   input _type_ $ @10 _col_ $13. @24 _row_ $16. _coef_;
   datalines;
eq        .             napha_l_conv            .
eq        .             napha_i_conv            .
eq        .             heating_oil_conv        .
eq        .             recipe_1                .
eq        .             recipe_2                .
.         a_light       napha_l_conv         .035
.         a_light       napha_i_conv         .100
.         a_light       heating_oil_conv     .390
.         a_heavy       napha_l_conv         .030
.         a_heavy       napha_i_conv         .075
.         a_heavy       heating_oil_conv     .300
.         brega         napha_l_conv         .045
.         brega         napha_i_conv         .135
.         brega         heating_oil_conv     .430
.         naphthal      napha_l_conv           -1
.         naphthal      recipe_2               .2
.         naphthai      napha_i_conv           -1
.         naphthai      recipe_1               .3
.         heatingo      heating_oil_conv       -1
.         heatingo      recipe_1               .7
.         heatingo      recipe_2               .8
.         jet_1         recipe_1               -1
.         jet_2         recipe_2               -1
;

proc netflow
   maximize
   sparsecondata
   arcdata=vars3        /* or arcdata=vars4 */
   condata=scon5
   conout=solutn5;
objfn profit;
upperbd available;
run;
```

The CONDATA=scon5 data set can become more concise by noting that all the constraints have the same type (eq) and zero (the default) rhs values. Use the DEFCONTYPE= option again. Once the first 5 observations of the CONDATA=scon5 data set are removed, the _type_ SAS variable has values that are missing in the remaining observations. Therefore, this SAS variable can be removed.

```
data scon6;
   input _col_ $ _row_&$16. _coef_;
   datalines;
```

```
a_light  napha_l_conv          .035
a_light  napha_i_conv          .100
a_light  heating_oil_conv      .390
a_heavy  napha_l_conv          .030
a_heavy  napha_i_conv          .075
a_heavy  heating_oil_conv      .300
brega    napha_l_conv          .045
brega    napha_i_conv          .135
brega    heating_oil_conv      .430
naphthal napha_l_conv            -1
naphthal recipe_2                .2
naphthai napha_i_conv            -1
naphthai recipe_1                .3
heatingo heating_oil_conv        -1
heatingo recipe_1                .7
heatingo recipe_2                .8
jet_1    recipe_1                -1
jet_2    recipe_2                -1
;

proc netflow
   maximize
   defcontype=eq
   sparsecondata
   arcdata=vars4        /* or arcdata=vars4 */
   condata=scon6
   conout=solutn6;
objfn profit;
upperbd available;
run;
```

Interactivity: Linear Programming Models: Interior Point algorithm

PROC NETFLOW can be used interactively. You begin by giving the PROC NETFLOW statement, and you must specify the CONDATA= data set. If necessary, specify the ARCDATA= data set.

The variable lists should be given next. If you have variables in the input data sets that have special names (for example, a variable in the ARCDATA= data set named _COST_ that has objective function coefficients as values), it may not be necessary to have many or any variable lists.

The PRINT, QUIT, SAVE, SHOW, RESET, and RUN statements follow and can be listed in any order. The QUIT statements can be used only once. The others can be used as many times as needed.

The CONOPT and PIVOT are not relevant to the Interior Point algorithm and should not be used.

Use the RESET or SAVE statement to change the name of the output data set. There is only one output data set, the CONOUT= data set. With the RESET statement, you can also indicate the reasons why optimization should stop, (for example, you can indicate the maximum number of iterations that can be performed). PROC NETFLOW then has a chance to either execute the next statement or, if the next statement is one that PROC NETFLOW does not recognize (the next PROC or DATA step in the SAS session), do any allowed optimization and finish. If no new statement has been submitted, you are prompted for one. Some options of the RESET statement enable you to control aspects of the Interior Point algorithm. Specifying certain values for these options can reduce the time it takes to solve a problem. Note that any of the RESET options can be specified in the PROC NETFLOW statement.

The RUN statement starts optimization. Once the optimization has started, it runs until the optimum is reached. The RUN statement should be specified at most once.

The QUIT statement immediately stops PROC NETFLOW. The SAVE statement has options that enable you to name the output data set; information about the current solution is saved in this output data set. Use the SHOW statement if you want to examine the values of options of other statements. Information about the amount of optimization that has been done and the STATUS of the current solution can also be displayed using the SHOW statement.

The PRINT statement instructs PROC NETFLOW to display parts of the problem. The ways the PRINT statements are specified are identical whether the Interior Point algorithm or the Simplex algorithm is used; however, there are minor differences in what is displayed for each variable or constraint coefficient.

PRINT VARIABLES produces information on all arcs. PRINT SOME_VARIABLES limits this output to a subset of variables. There are similar PRINT statements for variables and constraints:

```
PRINT CONSTRAINTS;
PRINT SOME_CONS;
```

PRINT CON_VARIABLES enables you to limit constraint information that is obtained to members of a set of variables that have nonzero constraint coefficients in a set of constraints.

For example, an interactive PROC NETFLOW run might look something like this:

```
proc netflow
       condata=data set
       other options;
   variable list specifications;     /* if necessary */
   reset options;
   print options;    /* look at problem              */
run;                 /* do some optimization         */
   print options;    /* look at the optimal solution */
   save options;     /* keep optimal solution        */
```

If you are interested only in finding the optimal solution, have used SAS variables that have special names in the input data sets, and want to use default setting for everything, then the following statement is all you need:

proc netflow condata=*data set options* **;**

Functional Summary: Linear Programming Models: Interior Point algorithm

The following tables outline the options available for the NETFLOW procedure when the Interior Point algorithm is being used to solve a linear programming problem, classified by function.

Table 4.36. Input Data Set Options

Description	Statement	Option
arcs input data set	NETFLOW	ARCDATA=
constraint input data set	NETFLOW	CONDATA=

Table 4.37. Options for Networks

Description	Statement	Option
default variable objective function coefficient	NETFLOW	DEFCOST=
default variable upper bound	NETFLOW	DEFCAPACITY=
default variable lower bound	NETFLOW	DEFMINFLOW=

Table 4.38. Miscellaneous Options

Description	Statement	Option
infinity value	NETFLOW	INFINITY=
do constraint row and/or variable column coefficient scaling, or neither	NETFLOW	SCALE=
maximization instead of minimization	NETFLOW	MAXIMIZE

Table 4.39. Data Set Read Options

Description	Statement	Option
CONDATA has sparse data format	NETFLOW	SPARSECONDATA
default constraint type	NETFLOW	DEFCONTYPE=
special COLUMN variable value	NETFLOW	TYPEOBS=
special COLUMN variable value	NETFLOW	RHSOBS=
data for an constraint found in only one obs of CONDATA	NETFLOW	CON_SINGLE_OBS
data for a coefficient found once in CONDATA	NETFLOW	NON_REPLIC=
data is grouped, exploited during data read	NETFLOW	GROUPED=

Table 4.40. Problem Size (approx.) Options

Description	Statement	Option
number of variables	NETFLOW	NNAS=
number of coefficients	NETFLOW	NCOEFS=
number of constraints	NETFLOW	NCONS=

Table 4.41. Memory Control Options

Description	Statement	Option
issue memory usage messages to SASLOG	NETFLOW	MEMREP
number of bytes to use for main memory	NETFLOW	BYTES=
proportion of memory used by frequently accessed arrays	NETFLOW	COREFACTOR=
maximum bytes for a single array	NETFLOW	MAXARRAYBYTES=

Table 4.42. Output Data Set Options: RESET

Description	Statement	Option
solution data set	RESET	CONOUT=

Table 4.43. PRINT Statement Options

Description	Statement	Option
display everything	PRINT	PROBLEM
display variables information	PRINT	VARIABLES
display constraint information	PRINT	CONSTRAINTS
display information for some variables	PRINT	SOME_VARIABLES
display information for some constraints	PRINT	SOME_CONS
display information for some constraints associated with some variables	PRINT	CON_VARIABLES

Table 4.44. PRINT Statement Qualifiers

Description	Statement	Option
produce a short report	PRINT	/ SHORT
produce a long report	PRINT	/ LONG
only variables with zero value	PRINT	/ ZERO
only variables with nonzero value	PRINT	/ NONZERO

Table 4.45. SHOW Statement Options

Description	Statement	Option
show problem, optimization status	SHOW	STATUS
show LP model parameters	SHOW	NETSTMT
show data sets that have, will be created	SHOW	DATA SETS

Table 4.46. Output Data Set Options: SAVE

Description	Statement	Option
constrained solution data set	SAVE	CONOUT=

Table 4.47. Interior Point algorithm Options

Description	Statement	Option
use Interior Point algorithm	NETFLOW	INTPOINT
allowed amount of dual infeasibility	RESET	TOLDINF=
allowed amount of primal infeasibility	RESET	TOLPINF=
cut-off tolerance for Cholesky factorization	RESET	CHOLTINYTOL=
density threshold for Cholesky processing	RESET	DENSETHR=
maximum number of Interior Point algorithm iterations	RESET	MAXITERB=
Primal-Dual (Duality) gap tolerance	RESET	PDGAPTOL=
step-length multiplier	RESET	PDSTEPMULT=
preprocessing type	RESET	PRSLTYPE=

Syntax: Linear Programming Models: Interior Point algorithm

Below are statements used in PROC NETFLOW, listed in alphabetical order as they appear in the text that follows.

PROC NETFLOW *options* **;**
  CAPACITY *variable* **;**
  COEF *variables* **;**
  COLUMN *variable* **;**
  COST *variable* **;**
  DEMAND *variable* **;**
  ID *variables* **;**
  LO *variable* **;**
  NAME *variable* **;**
  PRINT *options* **;**
  QUIT;
  RESET *options* **;**
  ROW *variables* **;**
  RHS *variables* **;**
  RUN;
  SAVE *options* **;**
  TYPE *variable* **;**
  VAR *variables* **;**

PROC NETFLOW *options*;	required statement
COST *variable*; **CAPACITY** *variable*; **LO** *variable*; **NAME** *variable*; **ID** *variables*;	optional ARCDATA lists
COLUMN *variable*; **ROW** *variables*; **COEF** *variables*; **TYPE** *variable*; **VAR** *variables*; **RHS** *variable*;	optional CONDATA lists
RESET *options*; **SAVE** *options*; **SHOW** *options*; **PRINT** *options*; **RUN**; **QUIT**;	optional interactive statements

Examples

The following examples illustrate some of the capabilities of PROC NETFLOW. These examples, together with the other SAS/OR examples, can be found in the SAS sample library.

Example 4.1. Shortest Path Problem

Whole pineapples are served in a restaurant in London. To ensure freshness, the pineapples are purchased in Hawaii and air freighted from Honolulu to Heathrow in London. The following network diagram (Figure 4.25) outlines the different routes that the pineapples could take.

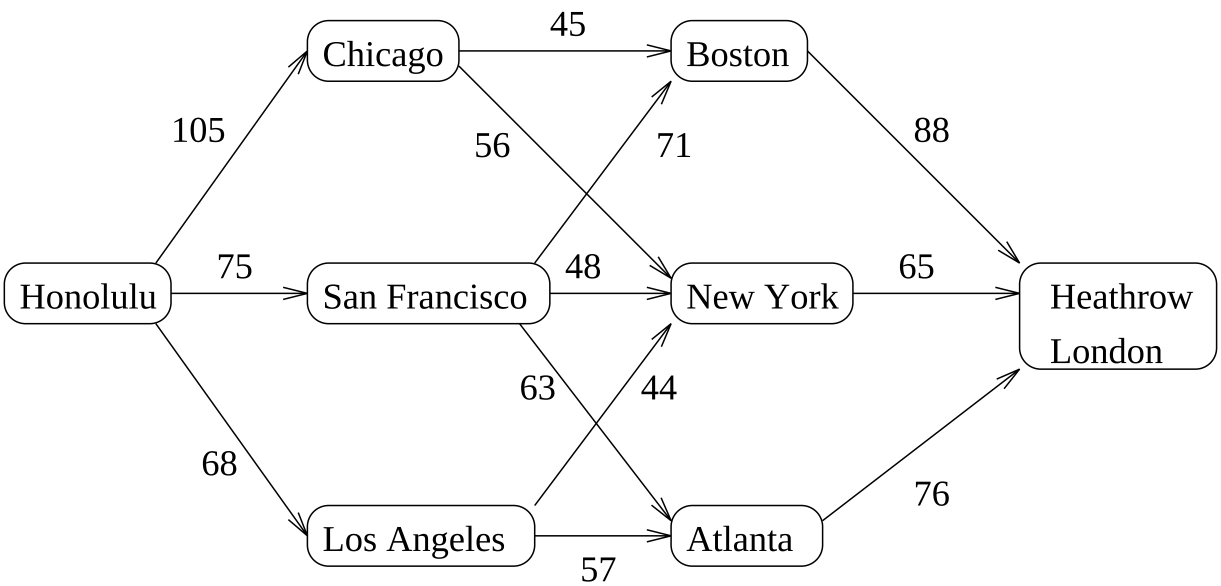


Figure 4.25. Pineapple routes

The cost to freight a pineapple is known for each arc. You can use PROC NETFLOW to determine what routes should be used to minimize total shipping cost. The shortest path is the least cost path that all pineapples should use. The SHORTPATH option indicates this type of network problem.

The SINK= option value HEATHROW LONDON is not a valid SAS variable name so it must be enclosed in single quotes. The TAILNODE list variable is FFROM. Because the name of this variable is not _TAIL_ or _FROM_, the TAILNODE list must be specified in the PROC NETFLOW statement. The HEADNODE list must also be explicitly specified because the variable that belongs to this list does not have the name _HEAD_ or _TO_, but is TTO.

```
title 'Shortest Path Problem';
title2 'How to get Hawaiian Pineapples to a London Restaurant';
data aircost1;
   input   ffrom&$13. tto&$15. _cost_ ;
```

```
   datalines;
Honolulu       Chicago          105
Honolulu       San Francisco     75
Honolulu       Los Angeles       68
Chicago        Boston            45
Chicago        New York          56
San Francisco  Boston            71
San Francisco  New York          48
San Francisco  Atlanta           63
Los Angeles    New York          44
Los Angeles    Atlanta           57
Boston         Heathrow London   88
New York       Heathrow London   65
Atlanta        Heathrow London   76
;

proc netflow
   shortpath
   sourcenode=Honolulu
   sinknode='Heathrow London'   /* Quotes for embedded blank */
      ARCDATA=aircost1
      arcout=spath;
         tail    ffrom;
         head    tto;
proc print data=spath; sum _fcost_;
run;
```

The length at optimality is written to the SAS log as

```
NOTE: Number of nodes= 8 .
NOTE: Number of arcs= 13 .
NOTE: Number of iterations performed (neglecting any
      constraints)= 5 .
NOTE: Of these, 4 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Shortest path= 177 .
NOTE: The data set WORK.SPATH has 13 observations and 13
      variables.
```

The output data set ARCOUT=SPATH in Output 4.1.1 shows that the best route for the pineapples is from Honolulu to Los Angeles to New York to Heathrow London.

Output 4.1.1. ARCOUT=SPATH

```
                                Shortest Path Problem
                How to get Hawaiian Pineapples to a London Restaurant

Obs  ffrom          tto              _cost_   _CAPAC_  _LO_  _SUPPLY_  _DEMAND_

  1  San Francisco  Atlanta            63    99999999    0       .         .
  2  Los Angeles    Atlanta            57    99999999    0       .         .
  3  Chicago        Boston             45    99999999    0       .         .
  4  San Francisco  Boston             71    99999999    0       .         .
  5  Honolulu       Chicago           105    99999999    0       1         .
  6  Boston         Heathrow London    88    99999999    0       .         1
  7  New York       Heathrow London    65    99999999    0       .         1
  8  Atlanta        Heathrow London    76    99999999    0       .         1
  9  Honolulu       Los Angeles        68    99999999    0       1         .
 10  Chicago        New York           56    99999999    0       .         .
 11  San Francisco  New York           48    99999999    0       .         .
 12  Los Angeles    New York           44    99999999    0       .         .
 13  Honolulu       San Francisco      75    99999999    0       1         .


Obs  _FLOW_    _FCOST_    _RCOST_    _ANUMB_    _TNUMB_        _STATUS_

  1     0          0         13          9         3       LOWERBD NONBASIC
  2     0          0          .         10         4       KEY_ARC    BASIC
  3     0          0          4          4         2       LOWERBD NONBASIC
  4     0          0          .          5         3       KEY_ARC    BASIC
  5     0          0          .          1         1       KEY_ARC    BASIC
  6     0          0         57         11         5       LOWERBD NONBASIC
  7     1         65          .         12         6       KEY_ARC    BASIC
  8     0          0         24         13         7       LOWERBD NONBASIC
  9     1         68          .          3         1       KEY_ARC    BASIC
 10     0          0         49          6         2       LOWERBD NONBASIC
 11     0          0         11          7         3       LOWERBD NONBASIC
 12     1         44          .          8         4       KEY_ARC    BASIC
 13     0          0          .          2         1       KEY_ARC    BASIC
              =======
                177
```

Example 4.2. Minimum Cost Flow Problem

You can continue to use the pineapple example in Example 4.1 by supposing that the airlines now stipulate that no more than 350 pineapples per week can be handled in any single leg of the journey. The restaurant uses 500 pineapples each week. How many pineapples should take each route between Hawaii and London?

You will probably have more minimum cost flow problems because they are more general than maximal flow and shortest path problems. A shortest path formulation is no longer valid because the sink node does not demand one flow unit.

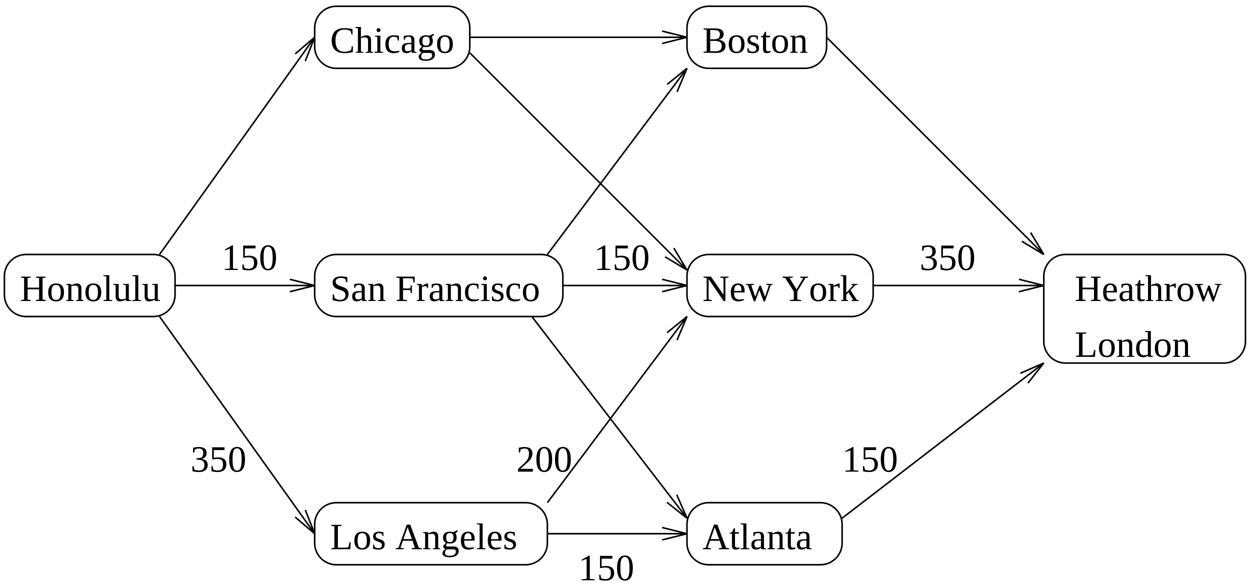
All arcs have the same capacity of 350 pineapples. Because of this, the DEFCAPACITY= option can be specified in the PROC NETFLOW statement, rather than having a CAPACITY list variable in ARCDATA=aircost1. You can have a CAPACITY list variable, but the value of this variable would be 350 in all observations, so using the DEFCAPACITY= option is more convenient. You would have to use the CAPACITY list variable if arcs had differing capacities. You can use both the DEFCAPACITY= option and a CAPACITY list variable.

There is only one supply node and one demand node. These can be named in the SOURCE= and SINK= options. DEMAND=500 is specified for the restaurant demand. There is no need to specify SUPPLY=500, as this is assumed.

```
title 'Minimum Cost Flow Problem';
title2 'How to get Hawaiian Pineapples to a London Restaurant';
proc netflow
   defcapacity=350
   sourcenode='Honolulu'
   sinknode='Heathrow London'  /* Quotes for embedded blank */
   demand=500
      arcdata=aircost1
      arcout=arcout1
      nodeout=nodeout1;
         tail    ffrom;
         head    tto;
   set future1;
proc print data=arcout1; sum _fcost_;
proc print data=nodeout1;
run;
```

The following notes appear on the SAS log:

```
NOTE: Sourcenode was assigned supply of the total network
      demand= 500 .
NOTE: Number of nodes= 8 .
NOTE: Number of supply nodes= 1 .
NOTE: Number of demand nodes= 1 .
NOTE: Total supply= 500 , total demand= 500 .
NOTE: Number of arcs= 13 .
NOTE: Number of iterations performed (neglecting any
      constraints)= 6 .
NOTE: Of these, 4 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= 93750 .
NOTE: The data set WORK.ARCOUT1 has 13 observations and
      13 variables.
NOTE: The data set WORK.NODEOUT1 has 9 observations and
      10 variables.
```



The routes and numbers of pineapples in each arc can be seen in the output data set ARCOUT=arcout1 in Output 4.2.1.

Output 4.2.1. ARCOUT=ARCOUT1

Minimum Cost Flow Problem
How to get Hawaiian Pineapples to a London Restaurant

Obs	ffrom	tto	_cost_	_CAPAC_	_LO_	_SUPPLY_	_DEMAND_
1	San Francisco	Atlanta	63	350	0	.	.
2	Los Angeles	Atlanta	57	350	0	.	.
3	Chicago	Boston	45	350	0	.	.
4	San Francisco	Boston	71	350	0	.	.
5	Honolulu	Chicago	105	350	0	500	.
6	Boston	Heathrow London	88	350	0	.	500
7	New York	Heathrow London	65	350	0	.	500
8	Atlanta	Heathrow London	76	350	0	.	500
9	Honolulu	Los Angeles	68	350	0	500	.
10	Chicago	New York	56	350	0	.	.
11	San Francisco	New York	48	350	0	.	.
12	Los Angeles	New York	44	350	0	.	.
13	Honolulu	San Francisco	75	350	0	500	.

Obs	_FLOW_	_FCOST_	_RCOST_	_ANUMB_	_TNUMB_	_STATUS_
1	0	0	2	9	3	LOWERBD NONBASIC
2	150	8550	.	10	4	KEY_ARC BASIC
3	0	0	4	4	2	LOWERBD NONBASIC
4	0	0	.	5	3	KEY_ARC BASIC
5	0	0	.	1	1	KEY_ARC BASIC
6	0	0	22	11	5	LOWERBD NONBASIC
7	350	22750	-24	12	6	UPPERBD NONBASIC
8	150	11400	.	13	7	KEY_ARC BASIC
9	350	23800	-11	3	1	UPPERBD NONBASIC
10	0	0	38	6	2	LOWERBD NONBASIC
11	150	7200	.	7	3	KEY_ARC BASIC
12	200	8800	.	8	4	KEY_ARC BASIC
13	150	11250	.	2	1	KEY_ARC BASIC
		=======				
		93750				

NODEOUT=NODEOUT1 is shown in Output 4.2.2.

Output 4.2.2. NODEOUT=NODEOUT1

Obs	_NODE_	_SUPDEM_	_DUAL_	_NNUMB_	_PRED_	_TRAV_	_SCESS_	_ARCID_	_FLOW_	_FBQ_
1	_ROOT_	0	0	9	0	1	0	-1	81	-14
2	Atlanta	.	-136	7	4	8	2	10	150	9
3	Boston	.	-146	5	3	9	1	5	0	4
4	Chicago	.	-105	2	1	3	1	1	0	1
5	Heathrow London	-500	-212	8	7	5	1	13	150	11
6	Honolulu	500	0	1	9	2	8	-14	0	-1
7	Los Angeles	.	-79	4	6	7	3	-8	200	3
8	New York	.	-123	6	3	4	4	7	150	6
9	San Francisco	.	-75	3	1	6	6	2	150	2

Example 4.3. Using a Warm Start

Suppose that the airlines state that the freight cost per pineapple in flights that leave Chicago has been reduced by 30. How many pineapples should take each route between Hawaii and London? This example illustrates how PROC NETFLOW uses a warm start.

In Example 4.2, the RESET statement of PROC NETFLOW is used to specify FUTURE1. A NODEOUT= data set is also specified. The warm start information is saved in the arcout1 and nodeout1 data sets.

In the following DATA step, the costs, reduced costs, and flows in the arcout1 data set are saved in variables called oldcost, oldflow, and oldfc. These variables form an implicit ID list in the following PROC NETFLOW run and will appear in ARCOUT=arcout2. Thus, it is easy to compare the previous optimum and the new optimum.

```
title 'Minimum Cost Flow Problem - Warm Start';
title2 'How to get Hawaiian Pineapples to a London Restaurant';
data aircost2;
   set arcout1;
      oldcost=_cost_;
      oldflow=_flow_;
      oldfc=_fcost_;
      if ffrom='Chicago' then _cost_=_cost_-30;
proc netflow
   warm
      arcdata=aircost2
      nodedata=nodeout1
      arcout=arcout2;
         tail    ffrom;
         head    tto;
proc print data=arcout2;
   var ffrom tto _cost_ oldcost _capac_ _lo_
       _flow_ oldflow _fcost_ oldfc;
```

```
   sum _fcost_ oldfc;
run;
```

The following notes appear on the SAS log:

```
NOTE: Number of nodes= 8 .
NOTE: Number of supply nodes= 1 .
NOTE: Number of demand nodes= 1 .
NOTE: Total supply= 500 , total demand= 500 .
NOTE: Number of iterations performed (neglecting any
      constraints)= 3 .
NOTE: Of these, 1 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= 93150 .
```

ARCOUT=arcout2 is shown in Output 4.3.1.

Output 4.3.1. ARCOUT=ARCOUT2

Minimum Cost Flow Problem - Warm Start
How to get Hawaiian Pineapples to a London Restaurant

Obs	ffrom	tto	_cost_	oldcost	_CAPAC_	_LO_	_FLOW_	oldflow	_FCOST_	oldfc
1	San Francisco	Atlanta	63	63	350	0	0	0	0	0
2	Los Angeles	Atlanta	57	57	350	0	0	150	0	8550
3	Chicago	Boston	15	45	350	0	150	0	2250	0
4	San Francisco	Boston	71	71	350	0	0	0	0	0
5	Honolulu	Chicago	105	105	350	0	150	0	15750	0
6	Boston	Heathrow London	88	88	350	0	150	0	13200	0
7	New York	Heathrow London	65	65	350	0	350	350	22750	22750
8	Atlanta	Heathrow London	76	76	350	0	0	150	0	11400
9	Honolulu	Los Angeles	68	68	350	0	350	350	23800	23800
10	Chicago	New York	26	56	350	0	0	0	0	0
11	San Francisco	New York	48	48	350	0	0	150	0	7200
12	Los Angeles	New York	44	44	350	0	350	200	15400	8800
13	Honolulu	San Francisco	75	75	350	0	0	150	0	11250
									=====	=====
									93150	93750

Example 4.4. Production, Inventory, Distribution Problem

Example 4.4 through Example 4.8 use data from a company that produces two sizes of televisions in order to illustrate variations in the use the NETFLOW procedure. The company makes televisions with a diagonal screen measurement of either 19 inches or 25 inches. These televisions are made between March and May at both of the company's two factories. Each factory has a limit on the total number of televisions of each screen dimension that can be made during those months.

The televisions are distributed to one of two shops, stored at the factory where they were made and sold later, or shipped to the other factory. Some sets can be used to fill back orders from the previous months. Each shop demands a number of each type of TV for the months March through May. The following network in Figure 4.26 illustrates the model. Arc costs can be interpreted as production costs, storage costs, back order penalty costs, inter-factory transportation costs, and sales profits. The arcs can have capacities and lower flow bounds.

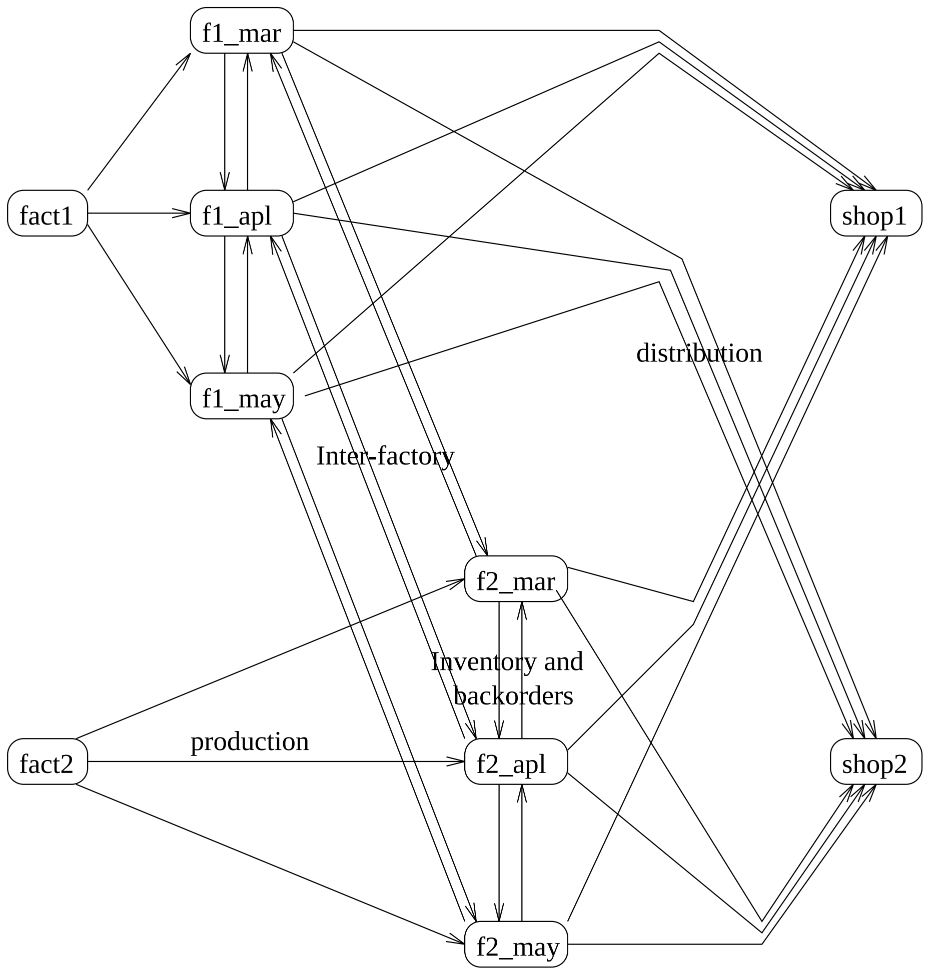


Figure 4.26. TV problem

There are two similarly structured networks, one for the 19-inch televisions and the other for the 25-inch screen TVs. The minimum cost production, inventory, and distribution plan for both TV types can be determined in the same run of PROC NETFLOW. To ensure that node names are unambiguous, the names of nodes in the 19-inch network have suffix _1, and the node names in the 25-inch network have suffix _2.

The FUTURE1 option is specified because further processing could be required. Information concerning an optimal solution is retained so it can be used to warm start later optimizations. Warm start information is mostly in variables named _NNUMB_, _PRED_, _TRAV_, _SCESS_, _ARCID_, and _FBQ_ and in observations for nodes named _EXCESS_ and _ROOT_, that are in the NODEOUT=NODE2 output data set. (PROC NETFLOW uses similar devices to store warm start information in the DUALOUT= data set when the FUTURE2 option is specified.) Variables _ANUMB_ and _TNUMB_ and observations for arcs directed from or toward a node called _EXCESS_ are present in ARCOUT=arc1. (PROC

NETFLOW uses similar devices to store warm start information in the CONOUT= data set when the FUTURE2 option is specified.)

The following code shows how to save the problem data in data sets and solve the model with PROC NETFLOW:

```
   title 'Minimum Cost Flow problem';
   title2 'Production Planning/Inventory/Distribution';
   data node0;
      input _node_ $ _supdem_ ;
      datalines;
   fact1_1   1000
   fact2_1    850
   fact1_2   1000
   fact2_2   1500
   shop1_1   -900
   shop2_1   -900
   shop1_2   -900
   shop2_2  -1450
   ;

   data arc0;
      input _tail_ $ _head_ $ _cost_ _capac_ _lo_ diagonal factory
            key_id $10. mth_made $ _name_&$17. ;
      datalines;
fact1_1  f1_mar_1  127.9  500 50 19 1 production March prod f1 19 mar
fact1_1  f1_apr_1   78.6  600 50 19 1 production April prod f1 19 apl
fact1_1  f1_may_1   95.1  400 50 19 1 production May   .
f1_mar_1 f1_apr_1   15     50  . 19 1 storage    March .
f1_apr_1 f1_may_1   12     50  . 19 1 storage    April .
f1_apr_1 f1_mar_1   28     20  . 19 1 backorder  April back f1 19 apl
f1_may_1 f1_apr_1   28     20  . 19 1 backorder  May   back f1 19 may
f1_mar_1 f2_mar_1   11      .  . 19 . f1_to_2    March .
f1_apr_1 f2_apr_1   11      .  . 19 . f1_to_2    April .
f1_may_1 f2_may_1   16      .  . 19 . f1_to_2    May   .
f1_mar_1 shop1_1  -327.65 250  . 19 1 sales      March .
f1_apr_1 shop1_1  -300    250  . 19 1 sales      April .
f1_may_1 shop1_1  -285    250  . 19 1 sales      May   .
f1_mar_1 shop2_1  -362.74 250  . 19 1 sales      March .
f1_apr_1 shop2_1  -300    250  . 19 1 sales      April .
f1_may_1 shop2_1  -245    250  . 19 1 sales      May   .
fact2_1  f2_mar_1   88.0  450 35 19 2 production March prod f2 19 mar
fact2_1  f2_apr_1   62.4  480 35 19 2 production April prod f2 19 apl
fact2_1  f2_may_1  133.8  250 35 19 2 production May   .
f2_mar_1 f2_apr_1   18     30  . 19 2 storage    March .
f2_apr_1 f2_may_1   20     30  . 19 2 storage    April .
f2_apr_1 f2_mar_1   17     15  . 19 2 backorder  April back f2 19 apl
f2_may_1 f2_apr_1   25     15  . 19 2 backorder  May   back f2 19 may
f2_mar_1 f1_mar_1   10     40  . 19 . f2_to_1    March .
f2_apr_1 f1_apr_1   11     40  . 19 . f2_to_1    April .
f2_may_1 f1_may_1   13     40  . 19 . f2_to_1    May   .
f2_mar_1 shop1_1  -297.4  250  . 19 2 sales      March .
f2_apr_1 shop1_1  -290    250  . 19 2 sales      April .
f2_may_1 shop1_1  -292    250  . 19 2 sales      May   .
```

```
f2_mar_1 shop2_1  -272.7  250  . 19 2 sales      March .
f2_apr_1 shop2_1  -312    250  . 19 2 sales      April .
f2_may_1 shop2_1  -299    250  . 19 2 sales      May   .
fact1_2  f1_mar_2  217.9  400 40 25 1 production March prod f1 25 mar
fact1_2  f1_apr_2  174.5  550 50 25 1 production April prod f1 25 apl
fact1_2  f1_may_2  133.3  350 40 25 1 production May   .
f1_mar_2 f1_apr_2   20     40  . 25 1 storage    March .
f1_apr_2 f1_may_2   18     40  . 25 1 storage    April .
f1_apr_2 f1_mar_2   32     30  . 25 1 backorder  April back f1 25 apl
f1_may_2 f1_apr_2   41     15  . 25 1 backorder  May   back f1 25 may
f1_mar_2 f2_mar_2   23      .  . 25 . f1_to_2    March .
f1_apr_2 f2_apr_2   23      .  . 25 . f1_to_2    April .
f1_may_2 f2_may_2   26      .  . 25 . f1_to_2    May   .
f1_mar_2 shop1_2  -559.76   .  . 25 1 sales      March .
f1_apr_2 shop1_2  -524.28   .  . 25 1 sales      April .
f1_may_2 shop1_2  -475.02   .  . 25 1 sales      May   .
f1_mar_2 shop2_2  -623.89   .  . 25 1 sales      March .
f1_apr_2 shop2_2  -549.68   .  . 25 1 sales      April .
f1_may_2 shop2_2  -460.00   .  . 25 1 sales      May   .
fact2_2  f2_mar_2  182.0  650 35 25 2 production March prod f2 25 mar
fact2_2  f2_apr_2  196.7  680 35 25 2 production April prod f2 25 apl
fact2_2  f2_may_2  201.4  550 35 25 2 production May   .
f2_mar_2 f2_apr_2   28     50  . 25 2 storage    March .
f2_apr_2 f2_may_2   38     50  . 25 2 storage    April .
f2_apr_2 f2_mar_2   31     15  . 25 2 backorder  April back f2 25 apl
f2_may_2 f2_apr_2   54     15  . 25 2 backorder  May   back f2 25 may
f2_mar_2 f1_mar_2   20     25  . 25 . f2_to_1    March .
f2_apr_2 f1_apr_2   21     25  . 25 . f2_to_1    April .
f2_may_2 f1_may_2   43     25  . 25 . f2_to_1    May   .
f2_mar_2 shop1_2  -567.83 500  . 25 2 sales      March .
f2_apr_2 shop1_2  -542.19 500  . 25 2 sales      April .
f2_may_2 shop1_2  -461.56 500  . 25 2 sales      May   .
f2_mar_2 shop2_2  -542.83 500  . 25 2 sales      March .
f2_apr_2 shop2_2  -559.19 500  . 25 2 sales      April .
f2_may_2 shop2_2  -489.06 500  . 25 2 sales      May   .
   ;

   proc netflow
          nodedata=node0
          arcdata=arc0;
      set future1
          nodeout=node2
          arcout=arc1;
   proc print data=arc1; sum _fcost_;
   proc print data=node2;
   run;
```

The following notes appear on the SAS log:

```
NOTE: Number of nodes= 20 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 4350 , total demand= 4150 .
NOTE: Number of arcs= 64 .
```

```
NOTE: Number of iterations performed (neglecting any
      constraints)= 74 .
NOTE: Of these, 1 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= -1281110.35 .
NOTE: The data set WORK.ARC1 has 68 observations and
      18 variables.
NOTE: The data set WORK.NODE2 has 22 observations and
      10 variables.
```

The solution is given in the NODEOUT=node2 and ARCOUT=arc1 data sets. In the ARCOUT= data set, shown in Output 4.4.1, the variables diagonal, factory, key_id, and mth_made form an implicit ID list. The diagonal variable has one of two values, 19 or 25. factory also has one of two values, 1 or 2, to denote the factory where either production or storage occurs, from where TVs are either sold to shops or satisfy back orders. PRODUCTION, STORAGE, SALES, and BACKORDER are values of the key_id variable.

Other values of this variable, F1_TO_2 and F2_TO_1, are used when flow through arcs represents the transportation of TVs between factories. The mth_made variable has values MARCH, APRIL, and MAY, the months when TVs that are modeled as flow through an arc were made (assuming that no televisions are stored for more than one month and none manufactured in May are used to fill March back orders).

These ID variables can be used after the PROC NETFLOW run to produce reports and perform analysis on particular parts of the company's operation. For example, reports can be generated for production numbers for each factory; optimal sales figures for each shop; and how many TVs should be stored, used to fill back orders, sent to the other factory, or any combination of these, for TVs with a particular screen, those produced in a particular month, or both.

Output 4.4.1. ARCOUT=ARC1

Minimum Cost Flow problem
Production Planning/Inventory/Distribution

OBS	_TAIL_	_HEAD_	_COST_	_CAPAC_	_LO_	_NAME_	_SUPPLY_	_DEMAND_	_FLOW_	_FCOST_
1	fact1_1	_EXCESS_	0.00	99999999	0		1000	200	5	0.00
2	fact2_1	_EXCESS_	0.00	99999999	0		850	200	45	0.00
3	fact1_2	_EXCESS_	0.00	99999999	0		1000	200	10	0.00
4	fact2_2	_EXCESS_	0.00	99999999	0		1500	200	140	0.00
5	fact1_1	f1_apr_1	78.60	600	50	prod f1 19 apl	1000	.	600	47160.00
6	f1_mar_1	f1_apr_1	15.00	50	0		.	.	0	0.00
7	f1_may_1	f1_apr_1	28.00	20	0	back f1 19 may	.	.	0	0.00
8	f2_apr_1	f1_apr_1	11.00	40	0		.	.	0	0.00
9	fact1_2	f1_apr_2	174.50	550	50	prod f1 25 apl	1000	.	550	95975.00
10	f1_mar_2	f1_apr_2	20.00	40	0		.	.	0	0.00
11	f1_may_2	f1_apr_2	41.00	15	0	back f1 25 may	.	.	15	615.00
12	f2_apr_2	f1_apr_2	21.00	25	0		.	.	0	0.00
13	fact1_1	f1_mar_1	127.90	500	50	prod f1 19 mar	1000	.	345	44125.50
14	f1_apr_1	f1_mar_1	28.00	20	0	back f1 19 apl	.	.	20	560.00
15	f2_mar_1	f1_mar_1	10.00	40	0		.	.	40	400.00
16	fact1_2	f1_mar_2	217.90	400	40	prod f1 25 mar	1000	.	400	87160.00
17	f1_apr_2	f1_mar_2	32.00	30	0	back f1 25 apl	.	.	30	960.00
18	f2_mar_2	f1_mar_2	20.00	25	0		.	.	25	500.00
19	fact1_1	f1_may_1	95.10	400	50		1000	.	50	4755.00
20	f1_apr_1	f1_may_1	12.00	50	0		.	.	50	600.00
21	f2_may_1	f1_may_1	13.00	40	0		.	.	0	0.00
22	fact1_2	f1_may_2	133.30	350	40		1000	.	40	5332.00
23	f1_apr_2	f1_may_2	18.00	40	0		.	.	0	0.00
24	f2_may_2	f1_may_2	43.00	25	0		.	.	0	0.00
25	f1_apr_1	f2_apr_1	11.00	99999999	0		.	.	30	330.00
26	fact2_1	f2_apr_1	62.40	480	35	prod f2 19 apl	850	.	480	29952.00
27	f2_mar_1	f2_apr_1	18.00	30	0		.	.	0	0.00
28	f2_may_1	f2_apr_1	25.00	15	0	back f2 19 may	.	.	0	0.00
29	f1_apr_2	f2_apr_2	23.00	99999999	0		.	.	0	0.00
30	fact2_2	f2_apr_2	196.70	680	35	prod f2 25 apl	1500	.	680	133756.00
31	f2_mar_2	f2_apr_2	28.00	50	0		.	.	0	0.00
32	f2_may_2	f2_apr_2	54.00	15	0	back f2 25 may	.	.	15	810.00
33	f1_mar_1	f2_mar_1	11.00	99999999	0		.	.	0	0.00
34	fact2_1	f2_mar_1	88.00	450	35	prod f2 19 mar	850	.	290	25520.00
35	f2_apr_1	f2_mar_1	17.00	15	0	back f2 19 apl	.	.	0	0.00
36	f1_mar_2	f2_mar_2	23.00	99999999	0		.	.	0	0.00
37	fact2_2	f2_mar_2	182.00	650	35	prod f2 25 mar	1500	.	645	117390.00
38	f2_apr_2	f2_mar_2	31.00	15	0	back f2 25 apl	.	.	0	0.00
39	f1_may_1	f2_may_1	16.00	99999999	0		.	.	100	1600.00
40	fact2_1	f2_may_1	133.80	250	35		850	.	35	4683.00
41	f2_apr_1	f2_may_1	20.00	30	0		.	.	15	300.00
42	f1_may_2	f2_may_2	26.00	99999999	0		.	.	0	0.00
43	fact2_2	f2_may_2	201.40	550	35		1500	.	35	7049.00
44	f2_apr_2	f2_may_2	38.00	50	0		.	.	0	0.00
45	f1_mar_1	shop1_1	-327.65	250	0		.	900	155	-50785.75
46	f1_apr_1	shop1_1	-300.00	250	0		.	900	250	-75000.00
47	f1_may_1	shop1_1	-285.00	250	0		.	900	0	0.00
48	f2_mar_1	shop1_1	-297.40	250	0		.	900	250	-74350.00
49	f2_apr_1	shop1_1	-290.00	250	0		.	900	245	-71050.00
50	f2_may_1	shop1_1	-292.00	250	0		.	900	0	0.00
51	f1_mar_2	shop1_2	-559.76	99999999	0		.	900	0	0.00
52	f1_apr_2	shop1_2	-524.28	99999999	0		.	900	0	0.00
53	f1_may_2	shop1_2	-475.02	99999999	0		.	900	25	-11875.50
54	f2_mar_2	shop1_2	-567.83	500	0		.	900	500	-283915.00
55	f2_apr_2	shop1_2	-542.19	500	0		.	900	375	-203321.25
56	f2_may_2	shop1_2	-461.56	500	0		.	900	0	0.00
57	f1_mar_1	shop2_1	-362.74	250	0		.	900	250	-90685.00
58	f1_apr_1	shop2_1	-300.00	250	0		.	900	250	-75000.00
59	f1_may_1	shop2_1	-245.00	250	0		.	900	0	0.00
60	f2_mar_1	shop2_1	-272.70	250	0		.	900	0	0.00
61	f2_apr_1	shop2_1	-312.00	250	0		.	900	250	-78000.00
62	f2_may_1	shop2_1	-299.00	250	0		.	900	150	-44850.00
63	f1_mar_2	shop2_2	-623.89	99999999	0		.	1450	455	-283869.95
64	f1_apr_2	shop2_2	-549.68	99999999	0		.	1450	535	-294078.80
65	f1_may_2	shop2_2	-460.00	99999999	0		.	1450	0	0.00
66	f2_mar_2	shop2_2	-542.83	500	0		.	1450	120	-65139.60
67	f2_apr_2	shop2_2	-559.19	500	0		.	1450	320	-178940.80
68	f2_may_2	shop2_2	-489.06	500	0		.	1450	20	-9781.20
										===========
										-1281110.35

OBS	_RCOST_	_ANUMB_	_TNUMB_	_STATUS_	DIAGONAL	FACTORY	KEY_ID	MTH_MADE
1	.	65	1	KEY_ARC BASIC	.	.		
2	.	66	10	KEY_ARC BASIC	.	.		
3	.	67	11	KEY_ARC BASIC	.	.		
4	.	68	20	KEY_ARC BASIC	.	.		
5	-0.650	4	1	UPPERBD NONBASIC	19	1	production	April
6	63.650	5	2	LOWERBD NONBASIC	19	1	storage	March
7	43.000	6	4	LOWERBD NONBASIC	19	1	backorder	May
8	22.000	7	6	LOWERBD NONBASIC	19	.	f2_to_1	April
9	-14.350	36	11	UPPERBD NONBASIC	25	1	production	April
10	94.210	37	12	LOWERBD NONBASIC	25	1	storage	March
11	-16.660	38	14	UPPERBD NONBASIC	25	1	backorder	May
12	30.510	39	16	LOWERBD NONBASIC	25	.	f2_to_1	April
13	.	1	1	KEY_ARC BASIC	19	1	production	March
14	-20.650	2	3	UPPERBD NONBASIC	19	1	backorder	April
15	-29.900	3	5	UPPERBD NONBASIC	19	.	f2_to_1	March
16	-45.160	33	11	UPPERBD NONBASIC	25	1	production	March
17	-42.210	34	13	UPPERBD NONBASIC	25	1	backorder	April
18	-61.060	35	15	UPPERBD NONBASIC	25	.	f2_to_1	March
19	0.850	8	1	LOWERBD NONBASIC	19	1	production	May
20	-3.000	9	3	UPPERBD NONBASIC	19	1	storage	April
21	29.000	10	7	LOWERBD NONBASIC	19	.	f2_to_1	May
22	2.110	40	11	LOWERBD NONBASIC	25	1	production	May
23	75.660	41	13	LOWERBD NONBASIC	25	1	storage	April
24	40.040	42	17	LOWERBD NONBASIC	25	.	f2_to_1	May
25	.	14	3	KEY_ARC BASIC	19	.	f1_to_2	April
26	-27.850	15	10	UPPERBD NONBASIC	19	2	production	April
27	15.750	16	5	LOWERBD NONBASIC	19	2	storage	March
28	45.000	17	7	LOWERBD NONBASIC	19	2	backorder	May
29	13.490	46	13	LOWERBD NONBASIC	25	.	f1_to_2	April
30	-1.660	47	20	UPPERBD NONBASIC	25	2	production	April
31	11.640	48	15	LOWERBD NONBASIC	25	2	storage	March
32	-16.130	49	17	UPPERBD NONBASIC	25	2	backorder	May
33	50.900	11	2	LOWERBD NONBASIC	19	.	f1_to_2	March
34	.	12	10	KEY_ARC BASIC	19	2	production	March
35	19.250	13	6	LOWERBD NONBASIC	19	2	backorder	April
36	104.060	43	12	LOWERBD NONBASIC	25	.	f1_to_2	March
37	.	44	20	KEY_ARC BASIC	25	2	production	March
38	47.360	45	16	LOWERBD NONBASIC	25	2	backorder	April
39	.	18	4	KEY_ARC BASIC	19	.	f1_to_2	May
40	23.550	19	10	LOWERBD NONBASIC	19	2	production	May
41	.	20	6	KEY_ARC BASIC	19	2	storage	April
42	28.960	50	14	LOWERBD NONBASIC	25	.	f1_to_2	May
43	73.170	51	20	LOWERBD NONBASIC	25	2	production	May
44	108.130	52	16	LOWERBD NONBASIC	25	2	storage	April
45	.	21	2	KEY_ARC BASIC	19	1	sales	March
46	-21.000	22	3	UPPERBD NONBASIC	19	1	sales	April
47	9.000	23	4	LOWERBD NONBASIC	19	1	sales	May
48	-9.650	24	5	UPPERBD NONBASIC	19	2	sales	March
49	.	25	6	KEY_ARC BASIC	19	2	sales	April
50	18.000	26	7	LOWERBD NONBASIC	19	2	sales	May
51	47.130	53	12	LOWERBD NONBASIC	25	1	sales	March
52	8.400	54	13	LOWERBD NONBASIC	25	1	sales	April
53	.	55	14	KEY_ARC BASIC	25	1	sales	May
54	-42.000	56	15	UPPERBD NONBASIC	25	2	sales	March
55	.	57	16	KEY_ARC BASIC	25	2	sales	April
56	10.500	58	17	LOWERBD NONBASIC	25	2	sales	May
57	-46.090	27	2	UPPERBD NONBASIC	19	1	sales	March
58	-32.000	28	3	UPPERBD NONBASIC	19	1	sales	April
59	38.000	29	4	LOWERBD NONBASIC	19	1	sales	May
60	4.050	30	5	LOWERBD NONBASIC	19	2	sales	March
61	-33.000	31	6	UPPERBD NONBASIC	19	2	sales	April
62	.	32	7	KEY_ARC BASIC	19	2	sales	May
63	.	59	12	KEY_ARC BASIC	25	1	sales	March
64	.	60	13	KEY_ARC BASIC	25	1	sales	April
65	32.020	61	14	LOWERBD NONBASIC	25	1	sales	May
66	.	62	15	KEY_ARC BASIC	25	2	sales	March
67	.	63	16	KEY_ARC BASIC	25	2	sales	April
68	.	64	17	KEY_ARC BASIC	25	2	sales	May

Output 4.4.2. NODEOUT=NODE2

Obs	_node_	_supdem_	_DUAL_	_NNUMB_	_PRED_	_TRAV_	_SCESS_	_ARCID_	_FLOW_	_FBQ_
1	_ROOT_	238	0.00	22	0	8	0	3	166	-69
2	_EXCESS_	-200	-100000198.75	21	1	11	13	65	5	65
3	f1_apr_1	.	-100000278.00	3	6	7	1	-14	30	4
4	f1_apr_2	.	-100000387.60	13	19	17	1	-60	535	36
5	f1_mar_1	.	-100000326.65	2	8	1	15	-21	155	1
6	f1_mar_2	.	-100000461.81	12	19	13	1	-59	455	33
7	f1_may_1	.	-100000293.00	4	7	2	1	-18	100	8
8	f1_may_2	.	-100000329.94	14	18	12	1	-55	25	40
9	f2_apr_1	.	-100000289.00	6	8	3	5	-25	245	14
10	f2_apr_2	.	-100000397.11	16	19	18	3	-63	320	46
11	f2_mar_1	.	-100000286.75	5	10	22	1	12	255	11
12	f2_mar_2	.	-100000380.75	15	20	19	8	44	610	43
13	f2_may_1	.	-100000309.00	7	6	9	3	20	15	18
14	f2_may_2	.	-100000326.98	17	19	10	1	-64	20	50
15	fact1_1	1000	-100000198.75	1	2	21	14	-1	295	-1
16	fact1_2	1000	-100000198.75	11	21	20	1	-67	10	-33
17	fact2_1	850	-100000198.75	10	21	5	2	-66	45	-33
18	fact2_2	1500	-100000198.75	20	21	15	9	-68	140	-65
19	shop1_1	-900	-99999999.00	8	22	6	21	0	0	21
20	shop1_2	-900	-99999854.92	18	16	14	2	57	375	53
21	shop2_1	-900	-100000010.00	9	7	4	1	32	150	27
22	shop2_2	-1450	-99999837.92	19	15	16	7	62	120	59

Example 4.5. Using an Unconstrained Solution Warm Start

This example examines the effect of changing some of the arc costs. The back order penalty costs are increased by twenty percent. The sales profit of 25-inch TVs sent to the shops in May is increased by thirty units. The backorder penalty costs of 25-inch TVs manufactured in May for April consumption is decreased by thirty units. The production cost of 19- and 25-inch TVs made in May are decreased by five units and twenty units, respectively. How does the optimal solution of the network after these arc cost alterations compare with the optimum of the original network? If you want to use the warm start facilities of PROC NETFLOW to solve this undefined problem, specify the WARM option. Notice that the FUTURE1 option was specified in the last PROC NETFLOW run.

These SAS statements produce the new NODEOUT= and ARCOUT= data sets.

```
title 'Minimum Cost Flow problem- Unconstrained Warm Start';
title2 'Production Planning/Inventory/Distribution';
data arc2;
   set arc1;
      oldcost=_cost_;
      oldfc=_fcost_;
      oldflow=_flow_;
      if key_id='backorder'
         then _cost_=_cost_*1.2;
```

```
          else if _tail_='f2_may_2' then _cost_=_cost_-30;
       if key_id='production' & mth_made='May' then
          if diagonal=19 then _cost_=_cost_-5;
                         else _cost_=_cost_-20;

proc netflow
   warm future1
   nodedata=node2
   arcdata=arc2
   nodeout=node3
   arcout=arc3;

proc print data=arc3 (drop = _status_ _rcost_);
   var _tail_ _head_ _capac_ _lo_ _supply_ _demand_ _name_
       _cost_ _flow_ _fcost_ oldcost oldflow oldfc
       diagonal factory key_id mth_made _anumb_ _tnumb_;
       /* to get this variable order */
           sum oldfc _fcost_;
proc print data=node3;
run;
```

The following notes appear on the SAS log:

```
NOTE: Number of nodes= 21 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 5 .
NOTE: Total supply= 4350 , total demand= 4350 .
NOTE: Number of iterations performed (neglecting any
      constraints)= 8 .
NOTE: Of these, 0 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= -1285086.45 .
NOTE: The data set WORK.ARC3 has 64 observations and
      21 variables.
NOTE: The data set WORK.NODE3 has 20 observations and
      10 variables.
```

The solution is displayed in Output 4.5.1.

Output 4.5.1. ARCOUT=ARC3

Minimum Cost Flow problem- Unconstrained Warm Start
Production Planning/Inventory/Distribution

OBS	_TAIL_	_HEAD_	_CAPAC_	_LO_	_SUPPLY_	_DEMAND_	_NAME_	_COST_	_FLOW_	_FCOST_
1	fact1_1	_EXCESS_	99999999	0	1000	200		0.00	5	0.00
2	fact2_1	_EXCESS_	99999999	0	850	200		0.00	45	0.00
3	fact1_2	_EXCESS_	99999999	0	1000	200		0.00	0	0.00
4	fact2_2	_EXCESS_	99999999	0	1500	200		0.00	150	0.00
5	fact1_1	f1_apr_1	600	50	1000	.	prod f1 19 apl	78.60	540	42444.00
6	f1_mar_1	f1_apr_1	50	0	.	.		15.00	0	0.00
7	f1_may_1	f1_apr_1	20	0	.	.	back f1 19 may	33.60	0	0.00
8	f2_apr_1	f1_apr_1	40	0	.	.		11.00	0	0.00
9	fact1_2	f1_apr_2	550	50	1000	.	prod f1 25 apl	174.50	250	43625.00
10	f1_mar_2	f1_apr_2	40	0	.	.		20.00	0	0.00
11	f1_may_2	f1_apr_2	15	0	.	.	back f1 25 may	49.20	15	738.00
12	f2_apr_2	f1_apr_2	25	0	.	.		21.00	0	0.00
13	fact1_1	f1_mar_1	500	50	1000	.	prod f1 19 mar	127.90	340	43486.00
14	f1_apr_1	f1_mar_1	20	0	.	.	back f1 19 apl	33.60	20	672.00
15	f2_mar_1	f1_mar_1	40	0	.	.		10.00	40	400.00
16	fact1_2	f1_mar_2	400	40	1000	.	prod f1 25 mar	217.90	400	87160.00
17	f1_apr_2	f1_mar_2	30	0	.	.	back f1 25 apl	38.40	30	1152.00
18	f2_mar_2	f1_mar_2	25	0	.	.		20.00	25	500.00
19	fact1_1	f1_may_1	400	50	1000	.		90.10	115	10361.50
20	f1_apr_1	f1_may_1	50	0	.	.		12.00	0	0.00
21	f2_may_1	f1_may_1	40	0	.	.		13.00	0	0.00
22	fact1_2	f1_may_2	350	40	1000	.		113.30	350	39655.00
23	f1_apr_2	f1_may_2	40	0	.	.		18.00	0	0.00
24	f2_may_2	f1_may_2	25	0	.	.		13.00	0	0.00
25	f1_apr_1	f2_apr_1	99999999	0	.	.		11.00	20	220.00
26	fact2_1	f2_apr_1	480	35	850	.	prod f2 19 apl	62.40	480	29952.00
27	f2_mar_1	f2_apr_1	30	0	.	.		18.00	0	0.00
28	f2_may_1	f2_apr_1	15	0	.	.	back f2 19 may	30.00	0	0.00
29	f1_apr_2	f2_apr_2	99999999	0	.	.		23.00	0	0.00
30	fact2_2	f2_apr_2	680	35	1500	.	prod f2 25 apl	196.70	680	133756.00
31	f2_mar_2	f2_apr_2	50	0	.	.		28.00	0	0.00
32	f2_may_2	f2_apr_2	15	0	.	.	back f2 25 may	64.80	0	0.00
33	f1_mar_1	f2_mar_1	99999999	0	.	.		11.00	0	0.00
34	fact2_1	f2_mar_1	450	35	850	.	prod f2 19 mar	88.00	290	25520.00
35	f2_apr_1	f2_mar_1	15	0	.	.	back f2 19 apl	20.40	0	0.00
36	f1_mar_2	f2_mar_2	99999999	0	.	.		23.00	0	0.00
37	fact2_2	f2_mar_2	650	35	1500	.	prod f2 25 mar	182.00	635	115570.00
38	f2_apr_2	f2_mar_2	15	0	.	.	back f2 25 apl	37.20	0	0.00
39	f1_may_1	f2_may_1	99999999	0	.	.		16.00	115	1840.00
40	fact2_1	f2_may_1	250	35	850	.		128.80	35	4508.00
41	f2_apr_1	f2_may_1	30	0	.	.		20.00	0	0.00
42	f1_may_2	f2_may_2	99999999	0	.	.		26.00	335	8710.00
43	fact2_2	f2_may_2	550	35	1500	.		181.40	35	6349.00
44	f2_apr_2	f2_may_2	50	0	.	.		38.00	0	0.00
45	f1_mar_1	shop1_1	250	0	.	900		-327.65	150	-49147.50
46	f1_apr_1	shop1_1	250	0	.	900		-300.00	250	-75000.00
47	f1_may_1	shop1_1	250	0	.	900		-285.00	0	0.00
48	f2_mar_1	shop1_1	250	0	.	900		-297.40	250	-74350.00
49	f2_apr_1	shop1_1	250	0	.	900		-290.00	250	-72500.00
50	f2_may_1	shop1_1	250	0	.	900		-292.00	0	0.00
51	f1_mar_2	shop1_2	99999999	0	.	900		-559.76	0	0.00
52	f1_apr_2	shop1_2	99999999	0	.	900		-524.28	0	0.00
53	f1_may_2	shop1_2	99999999	0	.	900		-475.02	0	0.00
54	f2_mar_2	shop1_2	500	0	.	900		-567.83	500	-283915.00
55	f2_apr_2	shop1_2	500	0	.	900		-542.19	400	-216876.00
56	f2_may_2	shop1_2	500	0	.	900		-491.56	0	0.00
57	f1_mar_1	shop2_1	250	0	.	900		-362.74	250	-90685.00
58	f1_apr_1	shop2_1	250	0	.	900		-300.00	250	-75000.00
59	f1_may_1	shop2_1	250	0	.	900		-245.00	0	0.00
60	f2_mar_1	shop2_1	250	0	.	900		-272.70	0	0.00
61	f2_apr_1	shop2_1	250	0	.	900		-312.00	250	-78000.00
62	f2_may_1	shop2_1	250	0	.	900		-299.00	150	-44850.00
63	f1_mar_2	shop2_2	99999999	0	.	1450		-623.89	455	-283869.95
64	f1_apr_2	shop2_2	99999999	0	.	1450		-549.68	235	-129174.80
65	f1_may_2	shop2_2	99999999	0	.	1450		-460.00	0	0.00
66	f2_mar_2	shop2_2	500	0	.	1450		-542.83	110	-59711.30
67	f2_apr_2	shop2_2	500	0	.	1450		-559.19	280	-156573.20
68	f2_may_2	shop2_2	500	0	.	1450		-519.06	370	-192052.20
										===========
										-1285086.45

OBS	OLDCOST	OLDFLOW	OLDFC	DIAGONAL	FACTORY	KEY_ID	MTH_MADE	_ANUMB_	_TNUMB_
1	0.00	5	0.00	.	.			65	1
2	0.00	45	0.00	.	.			66	10
3	0.00	10	0.00	.	.			67	11
4	0.00	140	0.00	.	.			68	20
5	78.60	600	47160.00	19	1	production	April	4	1
6	15.00	0	0.00	19	1	storage	March	5	2
7	28.00	0	0.00	19	1	backorder	May	6	4
8	11.00	0	0.00	19	.	f2_to_1	April	7	6
9	174.50	550	95975.00	25	1	production	April	36	11
10	20.00	0	0.00	25	1	storage	March	37	12
11	41.00	15	615.00	25	1	backorder	May	38	14
12	21.00	0	0.00	25	.	f2_to_1	April	39	16
13	127.90	345	44125.50	19	1	production	March	1	1
14	28.00	20	560.00	19	1	backorder	April	2	3
15	10.00	40	400.00	19	.	f2_to_1	March	3	5
16	217.90	400	87160.00	25	1	production	March	33	11
17	32.00	30	960.00	25	1	backorder	April	34	13
18	20.00	25	500.00	25	.	f2_to_1	March	35	15
19	95.10	50	4755.00	19	1	production	May	8	1
20	12.00	50	600.00	19	1	storage	April	9	3
21	13.00	0	0.00	19	.	f2_to_1	May	10	7
22	133.30	40	5332.00	25	1	production	May	40	11
23	18.00	0	0.00	25	1	storage	April	41	13
24	43.00	0	0.00	25	.	f2_to_1	May	42	17
25	11.00	30	330.00	19	.	f1_to_2	April	14	3
26	62.40	480	29952.00	19	2	production	April	15	10
27	18.00	0	0.00	19	2	storage	March	16	5
28	25.00	0	0.00	19	2	backorder	May	17	7
29	23.00	0	0.00	25	.	f1_to_2	April	46	13
30	196.70	680	133756.00	25	2	production	April	47	20
31	28.00	0	0.00	25	2	storage	March	48	15
32	54.00	15	810.00	25	2	backorder	May	49	17
33	11.00	0	0.00	19	.	f1_to_2	March	11	2
34	88.00	290	25520.00	19	2	production	March	12	10
35	17.00	0	0.00	19	2	backorder	April	13	6
36	23.00	0	0.00	25	.	f1_to_2	March	43	12
37	182.00	645	117390.00	25	2	production	March	44	20
38	31.00	0	0.00	25	2	backorder	April	45	16
39	16.00	100	1600.00	19	.	f1_to_2	May	18	4
40	133.80	35	4683.00	19	2	production	May	19	10
41	20.00	15	300.00	19	2	storage	April	20	6
42	26.00	0	0.00	25	.	f1_to_2	May	50	14
43	201.40	35	7049.00	25	2	production	May	51	20
44	38.00	0	0.00	25	2	storage	April	52	16
45	-327.65	155	-50785.75	19	1	sales	March	21	2
46	-300.00	250	-75000.00	19	1	sales	April	22	3
47	-285.00	0	0.00	19	1	sales	May	23	4
48	-297.40	250	-74350.00	19	2	sales	March	24	5
49	-290.00	245	-71050.00	19	2	sales	April	25	6
50	-292.00	0	0.00	19	2	sales	May	26	7
51	-559.76	0	0.00	25	1	sales	March	53	12
52	-524.28	0	0.00	25	1	sales	April	54	13
53	-475.02	25	-11875.50	25	1	sales	May	55	14
54	-567.83	500	-283915.00	25	2	sales	March	56	15
55	-542.19	375	-203321.25	25	2	sales	April	57	16
56	-461.56	0	0.00	25	2	sales	May	58	17
57	-362.74	250	-90685.00	19	1	sales	March	27	2
58	-300.00	250	-75000.00	19	1	sales	April	28	3
59	-245.00	0	0.00	19	1	sales	May	29	4
60	-272.70	0	0.00	19	2	sales	March	30	5
61	-312.00	250	-78000.00	19	2	sales	April	31	6
62	-299.00	150	-44850.00	19	2	sales	May	32	7
63	-623.89	455	-283869.95	25	1	sales	March	59	12
64	-549.68	535	-294078.80	25	1	sales	April	60	13
65	-460.00	0	0.00	25	1	sales	May	61	14
66	-542.83	120	-65139.60	25	2	sales	March	62	15
67	-559.19	320	-178940.80	25	2	sales	April	63	16
68	-489.06	20	-9781.20	25	2	sales	May	64	17
			===========						
			-1281110.35						

The associated NODEOUT data set is in Output 4.5.2

Output 4.5.2. NODEOUT=NODE3

Obs	_node_	_supdem_	_DUAL_	_NNUMB_	_PRED_	_TRAV_	_SCESS_	_ARCID_	_FLOW_	_FBQ_
1	_ROOT_	238	0.00	22	0	8	0	3	166	-69
2	_EXCESS_	-200	-100000198.75	21	1	20	13	65	5	65
3	f1_apr_1	.	-100000277.35	3	1	6	2	4	490	4
4	f1_apr_2	.	-100000387.60	13	19	11	2	-60	235	36
5	f1_mar_1	.	-100000326.65	2	8	1	20	-21	150	1
6	f1_mar_2	.	-100000461.81	12	19	13	1	-59	455	33
7	f1_may_1	.	-100000288.85	4	1	7	3	8	65	8
8	f1_may_2	.	-100000330.98	14	17	10	1	-50	335	40
9	f2_apr_1	.	-100000288.35	6	3	4	1	14	20	14
10	f2_apr_2	.	-100000397.11	16	19	18	2	-63	280	46
11	f2_mar_1	.	-100000286.75	5	10	22	1	12	255	11
12	f2_mar_2	.	-100000380.75	15	20	19	9	44	600	43
13	f2_may_1	.	-100000304.85	7	4	9	2	18	115	18
14	f2_may_2	.	-100000356.98	17	19	14	2	-64	370	50
15	fact1_1	1000	-100000198.75	1	2	3	19	-1	290	-1
16	fact1_2	1000	-100000213.10	11	13	17	1	-36	200	-33
17	fact2_1	850	-100000198.75	10	21	5	2	-66	45	-33
18	fact2_2	1500	-100000198.75	20	21	15	10	-68	150	-65
19	shop1_1	-900	-99999999.00	8	22	2	21	0	0	21
20	shop1_2	-900	-99999854.92	18	16	12	1	57	400	53
21	shop2_1	-900	-100000005.85	9	7	21	1	32	150	27
22	shop2_2	-1450	-99999837.92	19	15	16	8	62	110	59

Example 4.6. Adding Side Constraints, Using a Warm Start

The manufacturer of Gizmo chips, which are parts needed to make televisions, can supply only 2600 chips to factory 1 and 3750 chips to factory 2 in time for production in each of the months March and April. However, Gizmo chips will not be in short supply in May. Three chips are required to make each 19-inch TV while the 25-inch TVs require four chips each. To limit the production of televisions produced at factory 1 in March so that the TVs have the correct number of chips, a side constraint called FACT1 MAR GIZMO is used. The form of this constraint is

```
3 * prod f1 19 mar   +   4 * prod f1 25 mar   <=   2600
```

"prod f1 19 mar" is the name of the arc directed from the node fact1_1 toward node f1_mar_1 and, in the previous constraint, designates the flow assigned to this arc. The ARCDATA= and ARCOUT= data sets have arc names in a variable called _name_.

The other side constraints (shown below) are called FACT2 MAR GIZMO , FACT1 APL GIZMO and FACT2 APL GIZMO.

```
3 * prod f2 19 mar   +   4 * prod f2 25 mar   <=   3750
3 * prod f1 19 apl   +   4 * prod f1 25 apl   <=   2600
3 * prod f2 19 apl   +   4 * prod f2 25 apl   <=   3750
```

To maintain customer goodwill, the total number of backorders is not to exceed 50 sets. The side constraint TOTAL BACKORDER that models this restriction is:

```
back f1 19 apl  +  back f1 25 apl  +
back f2 19 apl  +  back f2 25 apl  +
back f1 19 may  +  back f1 25 may  +
back f2 19 may  +  back f2 25 may   <=   50
```

The sparse CONDATA= data set format is used. All side constraints are less than or equal type. Because this is the default type value for the DEFCONTYPE= option, type information is not necessary in the following CONDATA=CON3. Also, DEFCONTYPE='<=' does not have to be specified in the PROC NETFLOW statement that follows. Notice that the _column_ variable value CHIP/BO LIMIT indicates that an observation of the CON3 data set contains rhs information. Therefore, specify RHSOBS='CHIP/BO LIMIT' .

```
title 'Adding Side Constraints and Using a Warm Start';
title2 'Production Planning/Inventory/Distribution';
data con3;
   input _column_ &$14. _row_ &$15. _coef_ ;
   datalines;
prod f1 19 mar  FACT1 MAR GIZMO  3
prod f1 25 mar  FACT1 MAR GIZMO  4
CHIP/BO LIMIT   FACT1 MAR GIZMO  2600
prod f2 19 mar  FACT2 MAR GIZMO  3
prod f2 25 mar  FACT2 MAR GIZMO  4
CHIP/BO LIMIT   FACT2 MAR GIZMO  3750
prod f1 19 apl  FACT1 APL GIZMO  3
prod f1 25 apl  FACT1 APL GIZMO  4
CHIP/BO LIMIT   FACT1 APL GIZMO  2600
prod f2 19 apl  FACT2 APL GIZMO  3
prod f2 25 apl  FACT2 APL GIZMO  4
CHIP/BO LIMIT   FACT2 APL GIZMO  3750
back f1 19 apl  TOTAL BACKORDER  1
back f1 25 apl  TOTAL BACKORDER  1
back f2 19 apl  TOTAL BACKORDER  1
back f2 25 apl  TOTAL BACKORDER  1
back f1 19 may  TOTAL BACKORDER  1
back f1 25 may  TOTAL BACKORDER  1
back f2 19 may  TOTAL BACKORDER  1
back f2 25 may  TOTAL BACKORDER  1
CHIP/BO LIMIT   TOTAL BACKORDER  50
;
```

The four pairs of data sets that follow can be used as ARCDATA= and NODEDATA= data sets in the following PROC NETFLOW run. The set used depends on which cost information the arcs are to have and whether a warm start is to be used.

```
ARCDATA=arc0    NODEDATA=node0
ARCDATA=arc1    NODEDATA=node2
ARCDATA=arc2    NODEDATA=node2
ARCDATA=arc3    NODEDATA=node3
```

arc0, node0, arc1, and node2 were created in Example 4.4. The first two data sets are the original input data sets. arc1 and node2 were the ARCOUT= and NODE-OUT= data sets of a PROC NETFLOW run with FUTURE1 specified. Now, if you use arc1 and node2 as the ARCDATA= data set and NODEDATA= data set in a PROC NETFLOW run, you can specify WARM, as these data sets contain additional information describing a warm start.

In Example 4.5, arc2 was created by modifying arc1 to reflect different arc costs. arc2 and node2 can also be used as the ARCDATA= and NODEDATA= data sets in a PROC NETFLOW run. Again, specify WARM, as these data sets contain additional information describing a warm start, This start, however, contains the optimal basis using the original costs.

If you are going to continue optimization using the changed arc costs, it is probably best to use arc3 and node3 as the ARCDATA= and NODEDATA= data sets. These data sets, created in Example 4.6 by PROC NETFLOW when the FUTURE1 option was specified, contain an optimal basis that can be used as a warm start.

PROC NETFLOW is used to find the changed cost network solution that obeys the chip limit and backorder side constraints. The FUTURE2 option is specified in case further processing is required. An explicit ID list has also been specified so that the variables oldcost, oldfc and oldflow do not appear in the subsequent output data sets.

```
proc netflow
   nodedata=node3 arcdata=arc3 warm
   condata=con3 sparsecondata rhsobs='CHIP/BO LIMIT'
   future2 dualout=dual4 conout=con4;
      id diagonal factory key_id mth_made;
proc print data=con4;
   sum _fcost_;
proc print data=dual4;
```

The following messages appear on the SAS log:

```
NOTE: The following 3 variables in ARCDATA do not belong to
      any SAS variable list. These will be ignored.
      OLDCOST
      OLDFC
      OLDFLOW
NOTE: The following 1 variables in NODEDATA do not belong to
      any SAS variable list. These will be ignored.
      _DUAL_
NOTE: Number of nodes= 21 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 5 .
NOTE: The greater of total supply and total demand= 4350 .
NOTE: Number of iterations performed (neglecting any
      constraints)= 1 .
NOTE: Of these, 0 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= -1285086.45 .
NOTE: Number of <= side constraints= 5 .
```

```
NOTE: Number of == side constraints= 0 .
NOTE: Number of >= side constraints= 0 .
NOTE: Number of arc and nonarc variable side constraint
      coefficients= 16 .
NOTE: Number of iterations, optimizing with constraints= 10 .
NOTE: Of these, 0 were degenerate.
NOTE: Optimum reached.
NOTE: Minimal total cost= -1282708.625 .
NOTE: The data set WORK.CON4 has 68 observations and 18
      variables.
NOTE: The data set WORK.DUAL4 has 27 observations and 14
      variables.
```

Output 4.6.1. CONOUT=CON4

Adding Side Constraints and Using a Warm Start
Production Planning/Inventory/Distribution

OBS	_TAIL_	_HEAD_	_COST_	_CAPAC_	_LO_	_NAME_	_SUPPLY_	_DEMAND_	_FLOW_	_FCOST_
1	fact1_1	_EXCESS_	0.00	99999999	0		1000	200	5.000	0.00
2	fact2_1	_EXCESS_	0.00	99999999	0		850	200	45.000	0.00
3	fact1_2	_EXCESS_	0.00	99999999	0		1000	200	0.000	0.00
4	fact2_2	_EXCESS_	0.00	99999999	0		1500	200	150.000	0.00
5	fact1_1	f1_apr_1	78.60	600	50	prod f1 19 apl	1000	.	533.333	41920.00
6	f1_mar_1	f1_apr_1	15.00	50	0		.	.	0.000	0.00
7	f1_may_1	f1_apr_1	33.60	20	0	back f1 19 may	.	.	0.000	0.00
8	f2_apr_1	f1_apr_1	11.00	40	0		.	.	0.000	0.00
9	fact1_2	f1_apr_2	174.50	550	50	prod f1 25 apl	1000	.	250.000	43625.00
10	f1_mar_2	f1_apr_2	20.00	40	0		.	.	0.000	0.00
11	f1_may_2	f1_apr_2	49.20	15	0	back f1 25 may	.	.	0.000	0.00
12	f2_apr_2	f1_apr_2	21.00	25	0		.	.	0.000	0.00
13	fact1_1	f1_mar_1	127.90	500	50	prod f1 19 mar	1000	.	333.333	42633.33
14	f1_apr_1	f1_mar_1	33.60	20	0	back f1 19 apl	.	.	20.000	672.00
15	f2_mar_1	f1_mar_1	10.00	40	0		.	.	40.000	400.00
16	fact1_2	f1_mar_2	217.90	400	40	prod f1 25 mar	1000	.	400.000	87160.00
17	f1_apr_2	f1_mar_2	38.40	30	0	back f1 25 apl	.	.	30.000	1152.00
18	f2_mar_2	f1_mar_2	20.00	25	0		.	.	25.000	500.00
19	fact1_1	f1_may_1	90.10	400	50		1000	.	128.333	11562.83
20	f1_apr_1	f1_may_1	12.00	50	0		.	.	0.000	0.00
21	f2_may_1	f1_may_1	13.00	40	0		.	.	0.000	0.00
22	fact1_2	f1_may_2	113.30	350	40		1000	.	350.000	39655.00
23	f1_apr_2	f1_may_2	18.00	40	0		.	.	0.000	0.00
24	f2_may_2	f1_may_2	13.00	25	0		.	.	0.000	0.00
25	f1_apr_1	f2_apr_1	11.00	99999999	0		.	.	13.333	146.67
26	fact2_1	f2_apr_1	62.40	480	35	prod f2 19 apl	850	.	480.000	29952.00
27	f2_mar_1	f2_apr_1	18.00	30	0		.	.	0.000	0.00
28	f2_may_1	f2_apr_1	30.00	15	0	back f2 19 may	.	.	0.000	0.00
29	f1_apr_2	f2_apr_2	23.00	99999999	0		.	.	0.000	0.00
30	fact2_2	f2_apr_2	196.70	680	35	prod f2 25 apl	1500	.	577.500	113594.25
31	f2_mar_2	f2_apr_2	28.00	50	0		.	.	0.000	0.00
32	f2_may_2	f2_apr_2	64.80	15	0	back f2 25 may	.	.	0.000	0.00
33	f1_mar_1	f2_mar_1	11.00	99999999	0		.	.	0.000	0.00
34	fact2_1	f2_mar_1	88.00	450	35	prod f2 19 mar	850	.	290.000	25520.00
35	f2_apr_1	f2_mar_1	20.40	15	0	back f2 19 apl	.	.	0.000	0.00
36	f1_mar_2	f2_mar_2	23.00	99999999	0		.	.	0.000	0.00
37	fact2_2	f2_mar_2	182.00	650	35	prod f2 25 mar	1500	.	650.000	118300.00
38	f2_apr_2	f2_mar_2	37.20	15	0	back f2 25 apl	.	.	0.000	0.00
39	f1_may_1	f2_may_1	16.00	99999999	0		.	.	115.000	1840.00
40	fact2_1	f2_may_1	128.80	250	35		850	.	35.000	4508.00
41	f2_apr_1	f2_may_1	20.00	30	0		.	.	0.000	0.00
42	f1_may_2	f2_may_2	26.00	99999999	0		.	.	350.000	9100.00
43	fact2_2	f2_may_2	181.40	550	35		1500	.	122.500	22221.50
44	f2_apr_2	f2_may_2	38.00	50	0		.	.	0.000	0.00
45	f1_mar_1	shop1_1	-327.65	250	0		.	900	143.333	-46963.17
46	f1_apr_1	shop1_1	-300.00	250	0		.	900	250.000	-75000.00
47	f1_may_1	shop1_1	-285.00	250	0		.	900	13.333	-3800.00
48	f2_mar_1	shop1_1	-297.40	250	0		.	900	250.000	-74350.00
49	f2_apr_1	shop1_1	-290.00	250	0		.	900	243.333	-70566.67
50	f2_may_1	shop1_1	-292.00	250	0		.	900	0.000	0.00
51	f1_mar_2	shop1_2	-559.76	99999999	0		.	900	0.000	0.00
52	f1_apr_2	shop1_2	-524.28	99999999	0		.	900	0.000	0.00
53	f1_may_2	shop1_2	-475.02	99999999	0		.	900	0.000	0.00
54	f2_mar_2	shop1_2	-567.83	500	0		.	900	500.000	-283915.00
55	f2_apr_2	shop1_2	-542.19	500	0		.	900	400.000	-216876.00
56	f2_may_2	shop1_2	-491.56	500	0		.	900	0.000	0.00
57	f1_mar_1	shop2_1	-362.74	250	0		.	900	250.000	-90685.00
58	f1_apr_1	shop2_1	-300.00	250	0		.	900	250.000	-75000.00
59	f1_may_1	shop2_1	-245.00	250	0		.	900	0.000	0.00
60	f2_mar_1	shop2_1	-272.70	250	0		.	900	0.000	0.00
61	f2_apr_1	shop2_1	-312.00	250	0		.	900	250.000	-78000.00
62	f2_may_1	shop2_1	-299.00	250	0		.	900	150.000	-44850.00
63	f1_mar_2	shop2_2	-623.89	99999999	0		.	1450	455.000	-283869.95
64	f1_apr_2	shop2_2	-549.68	99999999	0		.	1450	220.000	-120929.60
65	f1_may_2	shop2_2	-460.00	99999999	0		.	1450	0.000	0.00
66	f2_mar_2	shop2_2	-542.83	500	0		.	1450	125.000	-67853.75
67	f2_apr_2	shop2_2	-559.19	500	0		.	1450	177.500	-99256.23
68	f2_may_2	shop2_2	-519.06	500	0		.	1450	472.500	-245255.85
										===========
										-1282708.63

OBS	_RCOST_	_ANUMB_	_TNUMB_	_STATUS_	DIAGONAL	FACTORY	KEY_ID	MTH_MADE
1	.	65	1	KEY_ARC BASIC	.	.		
2	.	66	10	KEY_ARC BASIC	.	.		
3	30.187	67	11	LOWERBD NONBASIC	.	.		
4	.	68	20	KEY_ARC BASIC	.	.		
5	.	4	1	KEY_ARC BASIC	19	1	production	April
6	63.650	5	2	LOWERBD NONBASIC	19	1	storage	March
7	47.020	6	4	LOWERBD NONBASIC	19	1	backorder	May
8	22.000	7	6	LOWERBD NONBASIC	19	.	f2_to_1	April
9	.	36	11	KEY_ARC BASIC	25	1	production	April
10	94.210	37	12	LOWERBD NONBASIC	25	1	storage	March
11	.	38	14	NONKEY ARC BASIC	25	1	backorder	May
12	30.510	39	16	LOWERBD NONBASIC	25	.	f2_to_1	April
13	.	1	1	KEY_ARC BASIC	19	1	production	March
14	-7.630	2	3	UPPERBD NONBASIC	19	1	backorder	April
15	-34.750	3	5	UPPERBD NONBASIC	19	.	f2_to_1	March
16	-31.677	33	11	UPPERBD NONBASIC	25	1	production	March
17	-28.390	34	13	UPPERBD NONBASIC	25	1	backorder	April
18	-61.060	35	15	UPPERBD NONBASIC	25	.	f2_to_1	March
19	.	8	1	KEY_ARC BASIC	19	1	production	May
20	6.000	9	3	LOWERBD NONBASIC	19	1	storage	April
21	29.000	10	7	LOWERBD NONBASIC	19	.	f2_to_1	May
22	-11.913	40	11	UPPERBD NONBASIC	25	1	production	May
23	74.620	41	13	LOWERBD NONBASIC	25	1	storage	April
24	39.000	42	17	LOWERBD NONBASIC	25	.	f2_to_1	May
25	.	14	3	KEY_ARC BASIC	19	.	f1_to_2	April
26	-14.077	15	10	UPPERBD NONBASIC	19	2	production	April
27	10.900	16	5	LOWERBD NONBASIC	19	2	storage	March
28	48.420	17	7	LOWERBD NONBASIC	19	2	backorder	May
29	13.490	46	13	LOWERBD NONBASIC	25	.	f1_to_2	April
30	.	47	20	KEY_ARC BASIC	25	2	production	April
31	11.640	48	15	LOWERBD NONBASIC	25	2	storage	March
32	32.090	49	17	LOWERBD NONBASIC	25	2	backorder	May
33	55.750	11	2	LOWERBD NONBASIC	19	.	f1_to_2	March
34	.	12	10	KEY_ARC BASIC	19	2	production	March
35	34.920	13	6	LOWERBD NONBASIC	19	2	backorder	April
36	104.060	43	12	LOWERBD NONBASIC	25	.	f1_to_2	March
37	-23.170	44	20	UPPERBD NONBASIC	25	2	production	March
38	60.980	45	16	LOWERBD NONBASIC	25	2	backorder	April
39	.	18	4	KEY_ARC BASIC	19	.	f1_to_2	May
40	22.700	19	10	LOWERBD NONBASIC	19	2	production	May
41	9.000	20	6	LOWERBD NONBASIC	19	2	storage	April
42	.	50	14	KEY_ARC BASIC	25	.	f1_to_2	May
43	.	51	20	NONKEY ARC BASIC	25	2	production	May
44	78.130	52	16	LOWERBD NONBASIC	25	2	storage	April
45	.	21	2	KEY_ARC BASIC	19	1	sales	March
46	-21.000	22	3	UPPERBD NONBASIC	19	1	sales	April
47	.	23	4	NONKEY ARC BASIC	19	1	sales	May
48	-14.500	24	5	UPPERBD NONBASIC	19	2	sales	March
49	.	25	6	NONKEY ARC BASIC	19	2	sales	April
50	9.000	26	7	LOWERBD NONBASIC	19	2	sales	May
51	47.130	53	12	LOWERBD NONBASIC	25	1	sales	March
52	8.400	54	13	LOWERBD NONBASIC	25	1	sales	April
53	1.040	55	14	LOWERBD NONBASIC	25	1	sales	May
54	-42.000	56	15	UPPERBD NONBASIC	25	2	sales	March
55	.	57	16	KEY_ARC BASIC	25	2	sales	April
56	10.500	58	17	LOWERBD NONBASIC	25	2	sales	May
57	-37.090	27	2	UPPERBD NONBASIC	19	1	sales	March
58	-23.000	28	3	UPPERBD NONBASIC	19	1	sales	April
59	38.000	29	4	LOWERBD NONBASIC	19	1	sales	May
60	8.200	30	5	LOWERBD NONBASIC	19	2	sales	March
61	-24.000	31	6	UPPERBD NONBASIC	19	2	sales	April
62	.	32	7	KEY_ARC BASIC	19	2	sales	May
63	.	59	12	KEY_ARC BASIC	25	1	sales	March
64	.	60	13	KEY_ARC BASIC	25	1	sales	April
65	33.060	61	14	LOWERBD NONBASIC	25	1	sales	May
66	.	62	15	KEY_ARC BASIC	25	2	sales	March
67	.	63	16	KEY_ARC BASIC	25	2	sales	April
68	.	64	17	KEY_ARC BASIC	25	2	sales	May

Output 4.6.2. DUALOUT=DUAL4

Obs	_node_	_supdem_	_DUAL_	_NNUMB_	_PRED_	_TRAV_	_SCESS_
1	_ROOT_	238	0.00	22	0	8	5
2	_EXCESS_	-200	-100000193.90	21	1	20	13
3	f1_apr_1	.	-100000278.00	3	1	6	2
4	f1_apr_2	.	-100000405.92	13	19	11	2
5	f1_mar_1	.	-100000326.65	2	8	1	20
6	f1_mar_2	.	-100000480.13	12	19	13	1
7	f1_may_1	.	-100000284.00	4	1	7	3
8	f1_may_2	.	-100000349.30	14	17	15	1
9	f2_apr_1	.	-100000289.00	6	3	4	1
10	f2_apr_2	.	-100000415.43	16	20	18	9
11	f2_mar_1	.	-100000281.90	5	10	3	1
12	f2_mar_2	.	-100000399.07	15	19	10	1
13	f2_may_1	.	-100000300.00	7	4	9	2
14	f2_may_2	.	-100000375.30	17	19	14	2
15	fact1_1	1000	-100000193.90	1	2	21	19
16	fact1_2	1000	-100000224.09	11	13	17	1
17	fact2_1	850	-100000193.90	10	21	5	2
18	fact2_2	1500	-100000193.90	20	21	16	10
19	shop1_1	-900	-99999999.00	8	22	2	21
20	shop1_2	-900	-99999873.24	18	16	19	1
21	shop2_1	-900	-100000001.00	9	7	22	1
22	shop2_2	-1450	-99999856.24	19	16	12	7
23		.	-1.83	2	8	.	.
24		.	-1.62	0	8	.	.
25		.	-6.21	3	17	.	.
26		.	0.00	1	1	.	1
27		.	-7.42	4	13	.	.

Obs	_ARCID_	_FLOW_	_FBQ_	_VALUE_	_RHS_	_TYPE_	_row_
1	3	166.000	-69	0	75		
2	65	5.000	65	.	.		
3	4	483.333	4	.	.		
4	-60	220.000	36	.	.		
5	-21	143.333	1	.	.		
6	-59	455.000	33	.	.		
7	8	78.333	8	.	.		
8	-50	350.000	40	.	.		
9	14	13.333	14	.	.		
10	47	542.500	46	.	.		
11	12	255.000	11	.	.		
12	-62	125.000	43	.	.		
13	18	115.000	18	.	.		
14	-64	472.500	50	.	.		
15	-1	283.333	-1	.	.		
16	-36	200.000	-33	.	.		
17	-66	45.000	-33	.	.		
18	-68	150.000	-65	.	.		
19	0	0.000	21	.	.		
20	57	400.000	53	.	.		
21	32	150.000	27	.	.		
22	63	177.500	59	.	.		
23	25	243.333	.	2600	2600	LE	FACT1 APL GIZMO
24	23	13.333	.	2600	2600	LE	FACT1 MAR GIZMO
25	51	87.500	.	3750	3750	LE	FACT2 APL GIZMO
26	.	280.000	.	3470	3750	LE	FACT2 MAR GIZMO
27	38	0.000	.	50	50	LE	TOTAL BACKORDER

Example 4.7. Using a Constrained Solution Warm Start

Suppose the 25-inch screen TVs produced at factory 1 in May can be sold at either shop with an increased profit of 40 dollars each. What is the new optimal solution? Because only arc costs have been changed, information about the present solution in DUALOUT=dual4 and CONOUT=con4 can be used as a warm start in the following PROC NETFLOW run. It is still necessary to specify CONDATA=con3 SPARSECONDATA RHSOBS='CHIP/BO LIMIT', since the CONDATA= data set is always read.

```
title 'Using a Constrained Solution Warm Start';
title2 'Production Planning/Inventory/Distribution';
data new_con4;
   set con4;
      oldcost=_cost_;
      oldflow=_flow_;
      oldfc=_fcost_;
      if _tail_='f1_may_2'
         & (_head_='shop1_2' | _head_='shop2_2')
         then _cost_=_cost_-40;

proc netflow
   warm arcdata=new_con4 dualin=dual4
   condata=con3 sparsecondata rhsobs='CHIP/BO LIMIT'
   dualout=dual5 conout=con5;
proc print data=con5 (drop = _status_ _rcost_);
   var _tail_ _head_ _capac_ _lo_ _supply_ _demand_ _name_
       _cost_ _flow_ _fcost_ oldcost oldflow oldfc
       diagonal factory key_id mth_made _anumb_ _tnumb_;
          /* to get this variable order */
           sum oldfc _fcost_;
proc print data=dual5;
```

The following messages appear on the SAS log:

```
NOTE: Number of nodes= 21 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 5 .
NOTE: The greater of total supply and total demand= 4350 .
NOTE: Number of <= side constraints= 5 .
NOTE: Number of == side constraints= 0 .
NOTE: Number of >= side constraints= 0 .
NOTE: Number of arc and nonarc variable side constraint
      coefficients= 16 .
NOTE: Number of iterations, optimizing with constraints= 7 .
NOTE: Of these, 1 were degenerate.
NOTE: Optimum reached.
NOTE: Minimal total cost= -1295661.8 .
NOTE: The data set WORK.CON5 has 64 observations and 21
      variables.
NOTE: The data set WORK.DUAL5 has 25 observations and 14
      variables.
```

Output 4.7.1. CONOUT=CON5

Using a Constrained Solution Warm Start 21
Production Planning/Inventory/Distribution 16:23 Friday, May 28, 1993

OBS	_TAIL_	_HEAD_	_CAPAC_	_LO_	_SUPPLY_	_DEMAND_	_NAME_	_COST_	_FLOW_	_FCOST_
1	fact1_1	f1_apr_1	600	50	1000	.	prod f1 19 apl	78.60	533.333	41920.00
2	f1_mar_1	f1_apr_1	50	0	.	.		15.00	0.000	0.00
3	f1_may_1	f1_apr_1	20	0	.	.	back f1 19 may	33.60	0.000	0.00
4	f2_apr_1	f1_apr_1	40	0	.	.		11.00	0.000	0.00
5	fact1_2	f1_apr_2	550	50	1000	.	prod f1 25 apl	174.50	250.000	43625.00
6	f1_mar_2	f1_apr_2	40	0	.	.		20.00	0.000	0.00
7	f1_may_2	f1_apr_2	15	0	.	.	back f1 25 may	49.20	0.000	0.00
8	f2_apr_2	f1_apr_2	25	0	.	.		21.00	0.000	0.00
9	fact1_1	f1_mar_1	500	50	1000	.	prod f1 19 mar	127.90	333.333	42633.33
10	f1_apr_1	f1_mar_1	20	0	.	.	back f1 19 apl	33.60	20.000	672.00
11	f2_mar_1	f1_mar_1	40	0	.	.		10.00	40.000	400.00
12	fact1_2	f1_mar_2	400	40	1000	.	prod f1 25 mar	217.90	400.000	87160.00
13	f1_apr_2	f1_mar_2	30	0	.	.	back f1 25 apl	38.40	30.000	1152.00
14	f2_mar_2	f1_mar_2	25	0	.	.		20.00	25.000	500.00
15	fact1_1	f1_may_1	400	50	1000	.		90.10	128.333	11562.83
16	f1_apr_1	f1_may_1	50	0	.	.		12.00	0.000	0.00
17	f2_may_1	f1_may_1	40	0	.	.		13.00	0.000	0.00
18	fact1_2	f1_may_2	350	40	1000	.		113.30	350.000	39655.00
19	f1_apr_2	f1_may_2	40	0	.	.		18.00	0.000	0.00
20	f2_may_2	f1_may_2	25	0	.	.		13.00	0.000	0.00
21	f1_apr_1	f2_apr_1	99999999	0	.	.		11.00	13.333	146.67
22	fact2_1	f2_apr_1	480	35	850	.	prod f2 19 apl	62.40	480.000	29952.00
23	f2_mar_1	f2_apr_1	30	0	.	.		18.00	0.000	0.00
24	f2_may_1	f2_apr_1	15	0	.	.	back f2 19 may	30.00	0.000	0.00
25	f1_apr_2	f2_apr_2	99999999	0	.	.		23.00	0.000	0.00
26	fact2_2	f2_apr_2	680	35	1500	.	prod f2 25 apl	196.70	550.000	108185.00
27	f2_mar_2	f2_apr_2	50	0	.	.		28.00	0.000	0.00
28	f2_may_2	f2_apr_2	15	0	.	.	back f2 25 may	64.80	0.000	0.00
29	f1_mar_1	f2_mar_1	99999999	0	.	.		11.00	0.000	0.00
30	fact2_1	f2_mar_1	450	35	850	.	prod f2 19 mar	88.00	290.000	25520.00
31	f2_apr_1	f2_mar_1	15	0	.	.	back f2 19 apl	20.40	0.000	0.00
32	f1_mar_2	f2_mar_2	99999999	0	.	.		23.00	0.000	0.00
33	fact2_2	f2_mar_2	650	35	1500	.	prod f2 25 mar	182.00	650.000	118300.00
34	f2_apr_2	f2_mar_2	15	0	.	.	back f2 25 apl	37.20	0.000	0.00
35	f1_may_1	f2_may_1	99999999	0	.	.		16.00	115.000	1840.00
36	fact2_1	f2_may_1	250	35	850	.		128.80	35.000	4508.00
37	f2_apr_1	f2_may_1	30	0	.	.		20.00	0.000	0.00
38	f1_may_2	f2_may_2	99999999	0	.	.		26.00	0.000	0.00
39	fact2_2	f2_may_2	550	35	1500	.		181.40	150.000	27210.00
40	f2_apr_2	f2_may_2	50	0	.	.		38.00	0.000	0.00
41	f1_mar_1	shop1_1	250	0	.	900		-327.65	143.333	-46963.17
42	f1_apr_1	shop1_1	250	0	.	900		-300.00	250.000	-75000.00
43	f1_may_1	shop1_1	250	0	.	900		-285.00	13.333	-3800.00
44	f2_mar_1	shop1_1	250	0	.	900		-297.40	250.000	-74350.00
45	f2_apr_1	shop1_1	250	0	.	900		-290.00	243.333	-70566.67
46	f2_may_1	shop1_1	250	0	.	900		-292.00	0.000	0.00
47	f1_mar_2	shop1_2	99999999	0	.	900		-559.76	0.000	0.00
48	f1_apr_2	shop1_2	99999999	0	.	900		-524.28	0.000	0.00
49	f1_may_2	shop1_2	99999999	0	.	900		-515.02	350.000	-180257.00
50	f2_mar_2	shop1_2	500	0	.	900		-567.83	500.000	-283915.00
51	f2_apr_2	shop1_2	500	0	.	900		-542.19	50.000	-27109.50
52	f2_may_2	shop1_2	500	0	.	900		-491.56	0.000	0.00
53	f1_mar_1	shop2_1	250	0	.	900		-362.74	250.000	-90685.00
54	f1_apr_1	shop2_1	250	0	.	900		-300.00	250.000	-75000.00
55	f1_may_1	shop2_1	250	0	.	900		-245.00	0.000	0.00
56	f2_mar_1	shop2_1	250	0	.	900		-272.70	0.000	0.00
57	f2_apr_1	shop2_1	250	0	.	900		-312.00	250.000	-78000.00
58	f2_may_1	shop2_1	250	0	.	900		-299.00	150.000	-44850.00
59	f1_mar_2	shop2_2	99999999	0	.	1450		-623.89	455.000	-283869.95
60	f1_apr_2	shop2_2	99999999	0	.	1450		-549.68	220.000	-120929.60
61	f1_may_2	shop2_2	99999999	0	.	1450		-500.00	0.000	0.00
62	f2_mar_2	shop2_2	500	0	.	1450		-542.83	125.000	-67853.75
63	f2_apr_2	shop2_2	500	0	.	1450		-559.19	500.000	-279595.00
64	f2_may_2	shop2_2	500	0	.	1450		-519.06	150.000	-77859.00
										===========
										-1295661.80

OBS	OLDCOST	OLDFLOW	OLDFC	DIAGONAL	FACTORY	KEY_ID	MTH_MADE	_ANUMB_	_TNUMB_
1	78.60	533.333	41920.00	19	1	production	April	4	1
2	15.00	0.000	0.00	19	1	storage	March	5	2
3	33.60	0.000	0.00	19	1	backorder	May	6	4
4	11.00	0.000	0.00	19	.	f2_to_1	April	7	6
5	174.50	250.000	43625.00	25	1	production	April	36	11
6	20.00	0.000	0.00	25	1	storage	March	37	12
7	49.20	0.000	0.00	25	1	backorder	May	38	14
8	21.00	0.000	0.00	25	.	f2_to_1	April	39	16
9	127.90	333.333	42633.33	19	1	production	March	1	1
10	33.60	20.000	672.00	19	1	backorder	April	2	3
11	10.00	40.000	400.00	19	.	f2_to_1	March	3	5
12	217.90	400.000	87160.00	25	1	production	March	33	11
13	38.40	30.000	1152.00	25	1	backorder	April	34	13
14	20.00	25.000	500.00	25	.	f2_to_1	March	35	15
15	90.10	128.333	11562.83	19	1	production	May	8	1
16	12.00	0.000	0.00	19	1	storage	April	9	3
17	13.00	0.000	0.00	19	.	f2_to_1	May	10	7
18	113.30	350.000	39655.00	25	1	production	May	40	11
19	18.00	0.000	0.00	25	1	storage	April	41	13
20	13.00	0.000	0.00	25	.	f2_to_1	May	42	17
21	11.00	13.333	146.67	19	.	f1_to_2	April	14	3
22	62.40	480.000	29952.00	19	2	production	April	15	10
23	18.00	0.000	0.00	19	2	storage	March	16	5
24	30.00	0.000	0.00	19	2	backorder	May	17	7
25	23.00	0.000	0.00	25	.	f1_to_2	April	46	13
26	196.70	577.500	113594.25	25	2	production	April	47	20
27	28.00	0.000	0.00	25	2	storage	March	48	15
28	64.80	0.000	0.00	25	2	backorder	May	49	17
29	11.00	0.000	0.00	19	.	f1_to_2	March	11	2
30	88.00	290.000	25520.00	19	2	production	March	12	10
31	20.40	0.000	0.00	19	2	backorder	April	13	6
32	23.00	0.000	0.00	25	.	f1_to_2	March	43	12
33	182.00	650.000	118300.00	25	2	production	March	44	20
34	37.20	0.000	0.00	25	2	backorder	April	45	16
35	16.00	115.000	1840.00	19	.	f1_to_2	May	18	4
36	128.80	35.000	4508.00	19	2	production	May	19	10
37	20.00	0.000	0.00	19	2	storage	April	20	6
38	26.00	350.000	9100.00	25	.	f1_to_2	May	50	14
39	181.40	122.500	22221.50	25	2	production	May	51	20
40	38.00	0.000	0.00	25	2	storage	April	52	16
41	-327.65	143.333	-46963.17	19	1	sales	March	21	2
42	-300.00	250.000	-75000.00	19	1	sales	April	22	3
43	-285.00	13.333	-3800.00	19	1	sales	May	23	4
44	-297.40	250.000	-74350.00	19	2	sales	March	24	5
45	-290.00	243.333	-70566.67	19	2	sales	April	25	6
46	-292.00	0.000	0.00	19	2	sales	May	26	7
47	-559.76	0.000	0.00	25	1	sales	March	53	12
48	-524.28	0.000	0.00	25	1	sales	April	54	13
49	-475.02	0.000	0.00	25	1	sales	May	55	14
50	-567.83	500.000	-283915.00	25	2	sales	March	56	15
51	-542.19	400.000	-216876.00	25	2	sales	April	57	16
52	-491.56	0.000	0.00	25	2	sales	May	58	17
53	-362.74	250.000	-90685.00	19	1	sales	March	27	2
54	-300.00	250.000	-75000.00	19	1	sales	April	28	3
55	-245.00	0.000	0.00	19	1	sales	May	29	4
56	-272.70	0.000	0.00	19	2	sales	March	30	5
57	-312.00	250.000	-78000.00	19	2	sales	April	31	6
58	-299.00	150.000	-44850.00	19	2	sales	May	32	7
59	-623.89	455.000	-283869.95	25	1	sales	March	59	12
60	-549.68	220.000	-120929.60	25	1	sales	April	60	13
61	-460.00	0.000	0.00	25	1	sales	May	61	14
62	-542.83	125.000	-67853.75	25	2	sales	March	62	15
63	-559.19	177.500	-99256.23	25	2	sales	April	63	16
64	-519.06	472.500	-245255.85	25	2	sales	May	64	17
			===========						
			-1282708.63						

Output 4.7.2. DUALOUT=DUAL5

Obs	_node_	_supdem_	_DUAL_	_NNUMB_	_PRED_	_TRAV_	_SCESS_
1	f1_apr_1	.	-100000278.00	3	1	6	2
2	f1_apr_2	.	-100000405.92	13	19	11	2
3	f1_mar_1	.	-100000326.65	2	8	1	20
4	f1_mar_2	.	-100000480.13	12	19	13	1
5	f1_may_1	.	-100000284.00	4	1	7	3
6	f1_may_2	.	-100000363.43	14	18	10	1
7	f2_apr_1	.	-100000289.00	6	3	4	1
8	f2_apr_2	.	-100000390.60	16	20	18	3
9	f2_mar_1	.	-100000281.90	5	10	3	1
10	f2_mar_2	.	-100000399.07	15	19	16	1
11	f2_may_1	.	-100000300.00	7	4	9	2
12	f2_may_2	.	-100000375.30	17	20	19	6
13	fact1_1	1000	-100000193.90	1	2	21	19
14	fact1_2	1000	-100000224.09	11	13	15	1
15	fact2_1	850	-100000193.90	10	21	5	2
16	fact2_2	1500	-100000193.90	20	21	17	10
17	shop1_1	-900	-99999999.00	8	22	2	21
18	shop1_2	-900	-99999848.41	18	16	14	2
19	shop2_1	-900	-100000001.00	9	7	22	1
20	shop2_2	-1450	-99999856.24	19	17	12	5
21		.	-1.83	2	8	.	.
22		.	-1.62	0	8	.	.
23		.	0.00	3	3	.	3
24		.	0.00	1	1	.	1
25		.	0.00	4	4	.	4

Obs	_ARCID_	_FLOW_	_FBQ_	_VALUE_	_RHS_	_TYPE_	_row_
1	4	483.333	4	.	.		
2	-60	220.000	36	.	.		
3	-21	143.333	1	.	.		
4	-59	455.000	33	.	.		
5	8	78.333	8	.	.		
6	-55	350.000	40	.	.		
7	14	13.333	14	.	.		
8	47	515.000	46	.	.		
9	12	255.000	11	.	.		
10	-62	125.000	43	.	.		
11	18	115.000	18	.	.		
12	51	115.000	50	.	.		
13	-1	283.333	-1	.	.		
14	-36	200.000	-33	.	.		
15	-66	45.000	-33	.	.		
16	-68	150.000	-65	.	.		
17	0	0.000	21	.	.		
18	57	50.000	53	.	.		
19	32	150.000	27	.	.		
20	64	150.000	59	.	.		
21	25	243.333	.	2600	2600	LE	FACT1 APL GIZMO
22	23	13.333	.	2600	2600	LE	FACT1 MAR GIZMO
23	.	110.000	.	3640	3750	LE	FACT2 APL GIZMO
24	.	280.000	.	3470	3750	LE	FACT2 MAR GIZMO
25	.	0.000	.	50	50	LE	TOTAL BACKORDER

Example 4.8. Nonarc Variables in the Side Constraints

Notice in DUALOUT=dual5 from Example 4.7 the FACT2 MAR GIZMO constraint (observation 24) has a _VALUE_ of 3470, which is not equal to the _RHS_ of this constraint. Not all of the 3750 chips that can be supplied to factory 2 for March production are used. It is suggested that all the possible chips be obtained in March and those not used be saved for April production. Because chips must be kept in an air-controlled environment, it costs 1 dollar to store each chip purchased in March until April. The maximum number of chips that can be stored in this environment at each factory is 150. In addition, a search of the parts inventory at factory 1 turned up 15 chips available for their March production.

Nonarc variables are used in the side constraints that handle the limitations of supply of Gizmo chips. A nonarc variable called “f1 unused mar” has as a value the number of chips that are not used at factory 1 in March. Another nonarc variable, “f2 unused mar”, has as a value the number of chips that are not used at factory 2 in March. “f1 chips from mar” has as a value the number of chips left over from March used for production at factory 1 in April. Similarly, “f2 chips from mar” has as a value the number of chips left over from March used for April production at factory 2 in April. The last two nonarc variables have objective function coefficients of 1 and upper bounds of 150. The Gizmo side constraints are

```
3*prod f1 19 mar + 4*prod f1 25 mar + f1 unused chips = 2615
3*prod f2 19 apl + 4*prod f2 25 apl + f2 unused chips = 3750
3*prod f1 19 apl + 4*prod f1 25 apl - f1 chips from mar = 2600
3*prod f2 19 apl + 4*prod f2 25 apl - f2 chips from mar = 3750
f1 unused chips + f2 unused chips -
f1 chips from mar - f2 chips from mar >= 0
```

The last side constraint states that the number of chips not used in March is not less than the number of chips left over from March and used in April. Here, this constraint is called CHIP LEFTOVER.

The following SAS code creates a new data set containing constraint data. It seems that most of the constraints are now equalities, so you specify DEFCONTYPE=EQ in the PROC NETFLOW statements from now on and provide constraint type data for constraints that are not “equal to” type, using the default TYPEOBS value _TYPE_ as the _COLUMN_ variable value to indicate observations that contain constraint type data. Also, from now on, the default RHSOBS value is used.

```
title 'Nonarc Variables in the Side Constraints';
title2 'Production Planning/Inventory/Distribution';
data con6;
   input _column_ &$17. _row_ &$15. _coef_ ;
   datalines;
prod f1 19 mar     FACT1 MAR GIZMO  3
prod f1 25 mar     FACT1 MAR GIZMO  4
f1 unused chips    FACT1 MAR GIZMO  1
_RHS_              FACT1 MAR GIZMO  2615
prod f2 19 mar     FACT2 MAR GIZMO  3
prod f2 25 mar     FACT2 MAR GIZMO  4
f2 unused chips    FACT2 MAR GIZMO  1
_RHS_              FACT2 MAR GIZMO  3750
prod f1 19 apl     FACT1 APL GIZMO  3
prod f1 25 apl     FACT1 APL GIZMO  4
f1 chips from mar  FACT1 APL GIZMO  -1
_RHS_              FACT1 APL GIZMO  2600
prod f2 19 apl     FACT2 APL GIZMO  3
prod f2 25 apl     FACT2 APL GIZMO  4
f2 chips from mar  FACT2 APL GIZMO  -1
_RHS_              FACT2 APL GIZMO  3750
f1 unused chips    CHIP LEFTOVER    1
f2 unused chips    CHIP LEFTOVER    1
```

```
f1 chips from mar  CHIP LEFTOVER    -1
f2 chips from mar  CHIP LEFTOVER    -1
_TYPE_             CHIP LEFTOVER    1
back f1 19 apl     TOTAL BACKORDER  1
back f1 25 apl     TOTAL BACKORDER  1
back f2 19 apl     TOTAL BACKORDER  1
back f2 25 apl     TOTAL BACKORDER  1
back f1 19 may     TOTAL BACKORDER  1
back f1 25 may     TOTAL BACKORDER  1
back f2 19 may     TOTAL BACKORDER  1
back f2 25 may     TOTAL BACKORDER  1
_TYPE_             TOTAL BACKORDER  -1
_RHS_              TOTAL BACKORDER  50
;
```

The nonarc variables “f1 chips from mar” and “f2 chips from mar” have objective function coefficients of 1 and upper bounds of 150. There are various ways in which this information can be furnished to PROC NETFLOW. If there were a TYPE list variable in the CONDATA= data set, observations could be in the form:

```
_COLUMN_           _TYPE_  _ROW_ _COEF_
f1 chips from mar  objfn     .       1
f1 chips from mar  upperbd   .     150
f2 chips from mar  objfn     .       1
f2 chips from mar  upperbd   .     150
```

It is desireable to assign ID list variable values to all the nonarc variables:

```
data arc6;
   set con5;
   drop oldcost oldfc oldflow _flow_ _fcost_ _status_ _rcost_;
data arc6_b;
   input _name_ &$17. _cost_ _capac_ factory key_id $ ;
   datalines;
f1 unused chips    .   . 1 chips
f2 unused chips    .   . 2 chips
f1 chips from mar  1 150 1 chips
f2 chips from mar  1 150 2 chips
;

proc append
   base=arc6 data=arc6_b;
proc netflow
   nodedata=node0 arcdata=arc6
   condata=con6 defcontype=eq sparsecondata
   dualout=dual7 conout=con7;
run;
print nonarcs/short;
```

The following messages appear on the SAS log:

```
NOTE: Number of nodes= 20 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 4350 , total demand= 4150 .
NOTE: Number of arcs= 64 .
NOTE: Number of nonarc variables= 8 .
NOTE: Number of iterations performed (neglecting any
      constraints)= 70 .
NOTE: Of these, 1 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= -1295730.8 .
NOTE: Number of <= side constraints= 1 .
NOTE: Number of == side constraints= 4 .
NOTE: Number of >= side constraints= 1 .
NOTE: Number of arc and nonarc variable side constraint
      coefficients= 24 .
NOTE: Number of iterations, optimizing with constraints= 13 .
NOTE: Of these, 3 were degenerate.
NOTE: Optimum reached.
NOTE: Minimal total cost= -1295542.742 .
NOTE: The data set WORK.CON7 has 68 observations and 18
      variables.
NOTE: The data set WORK.DUAL7 has 26 observations and 14
      variables.
```

The output in Output 4.8.1 is produced by **print***nonarcs/short;*

Output 4.8.1. Output of PRINT NONARCS/SHORT;

```
                 Nonarc Variables in the Side Constraints
                 Production Planning/Inventory/Distribution

                            NETFLOW PROCEDURE

     _N_    _name_               _cost_     _capac_        _lo_     _VALUE_

      1     f1 chips from mar       1          150          0          20
      2     f1 unused chips         0     99999999          0           0
      3     f2 chips from mar       1          150          0           0
      4     f2 unused chips         0     99999999          0         280
```

The optimal solution data sets, CONOUT=CON7 in Output 4.8.2 and DUALOUT=DUAL7 in Output 4.8.3 follow:

```
proc print data=con7;
   sum _fcost_;
proc print data=dual7;
```

Output 4.8.2. CONOUT=CON7

Nonarc Variables in the Side Constraints
Production Planning/Inventory/Distribution

OBS	_TAIL_	_HEAD_	_COST_	_CAPAC_	_LO_	_NAME_	_SUPPLY_	_DEMAND_	_FLOW_
1	fact1_1	f1_apr_1	78.60	600	50	prod f1 19 apl	1000	.	540.000
2	f1_mar_1	f1_apr_1	15.00	50	0		.	.	0.000
3	f1_may_1	f1_apr_1	33.60	20	0	back f1 19 may	.	.	0.000
4	f2_apr_1	f1_apr_1	11.00	40	0		.	.	0.000
5	fact1_2	f1_apr_2	174.50	550	50	prod f1 25 apl	1000	.	250.000
6	f1_mar_2	f1_apr_2	20.00	40	0		.	.	0.000
7	f1_may_2	f1_apr_2	49.20	15	0	back f1 25 may	.	.	0.000
8	f2_apr_2	f1_apr_2	21.00	25	0		.	.	25.000
9	fact1_1	f1_mar_1	127.90	500	50	prod f1 19 mar	1000	.	338.333
10	f1_apr_1	f1_mar_1	33.60	20	0	back f1 19 apl	.	.	20.000
11	f2_mar_1	f1_mar_1	10.00	40	0		.	.	40.000
12	fact1_2	f1_mar_2	217.90	400	40	prod f1 25 mar	1000	.	400.000
13	f1_apr_2	f1_mar_2	38.40	30	0	back f1 25 apl	.	.	30.000
14	f2_mar_2	f1_mar_2	20.00	25	0		.	.	25.000
15	fact1_1	f1_may_1	90.10	400	50		1000	.	116.667
16	f1_apr_1	f1_may_1	12.00	50	0		.	.	0.000
17	f2_may_1	f1_may_1	13.00	40	0		.	.	0.000
18	fact1_2	f1_may_2	113.30	350	40		1000	.	350.000
19	f1_apr_2	f1_may_2	18.00	40	0		.	.	0.000
20	f2_may_2	f1_may_2	13.00	25	0		.	.	0.000
21	f1_apr_1	f2_apr_1	11.00	99999999	0		.	.	20.000
22	fact2_1	f2_apr_1	62.40	480	35	prod f2 19 apl	850	.	480.000
23	f2_mar_1	f2_apr_1	18.00	30	0		.	.	0.000
24	f2_may_1	f2_apr_1	30.00	15	0	back f2 19 may	.	.	0.000
25	f1_apr_2	f2_apr_2	23.00	99999999	0		.	.	0.000
26	fact2_2	f2_apr_2	196.70	680	35	prod f2 25 apl	1500	.	577.500
27	f2_mar_2	f2_apr_2	28.00	50	0		.	.	0.000
28	f2_may_2	f2_apr_2	64.80	15	0	back f2 25 may	.	.	0.000
29	f1_mar_1	f2_mar_1	11.00	99999999	0		.	.	0.000
30	fact2_1	f2_mar_1	88.00	450	35	prod f2 19 mar	850	.	290.000
31	f2_apr_1	f2_mar_1	20.40	15	0	back f2 19 apl	.	.	0.000
32	f1_mar_2	f2_mar_2	23.00	99999999	0		.	.	0.000
33	fact2_2	f2_mar_2	182.00	650	35	prod f2 25 mar	1500	.	650.000
34	f2_apr_2	f2_mar_2	37.20	15	0	back f2 25 apl	.	.	0.000
35	f1_may_1	f2_may_1	16.00	99999999	0		.	.	115.000
36	fact2_1	f2_may_1	128.80	250	35		850	.	35.000
37	f2_apr_1	f2_may_1	20.00	30	0		.	.	0.000
38	f1_may_2	f2_may_2	26.00	99999999	0		.	.	0.000
39	fact2_2	f2_may_2	181.40	550	35		1500	.	122.500
40	f2_apr_2	f2_may_2	38.00	50	0		.	.	0.000
41	f1_mar_1	shop1_1	-327.65	250	0		.	900	148.333
42	f1_apr_1	shop1_1	-300.00	250	0		.	900	250.000
43	f1_may_1	shop1_1	-285.00	250	0		.	900	1.667
44	f2_mar_1	shop1_1	-297.40	250	0		.	900	250.000
45	f2_apr_1	shop1_1	-290.00	250	0		.	900	250.000
46	f2_may_1	shop1_1	-292.00	250	0		.	900	0.000
47	f1_mar_2	shop1_2	-559.76	99999999	0		.	900	0.000
48	f1_apr_2	shop1_2	-524.28	99999999	0		.	900	0.000
49	f1_may_2	shop1_2	-515.02	99999999	0		.	900	347.500
50	f2_mar_2	shop1_2	-567.83	500	0		.	900	500.000
51	f2_apr_2	shop1_2	-542.19	500	0		.	900	52.500
52	f2_may_2	shop1_2	-491.56	500	0		.	900	0.000
53	f1_mar_1	shop2_1	-362.74	250	0		.	900	250.000
54	f1_apr_1	shop2_1	-300.00	250	0		.	900	250.000
55	f1_may_1	shop2_1	-245.00	250	0		.	900	0.000
56	f2_mar_1	shop2_1	-272.70	250	0		.	900	0.000
57	f2_apr_1	shop2_1	-312.00	250	0		.	900	250.000
58	f2_may_1	shop2_1	-299.00	250	0		.	900	150.000
59	f1_mar_2	shop2_2	-623.89	99999999	0		.	1450	455.000
60	f1_apr_2	shop2_2	-549.68	99999999	0		.	1450	245.000
61	f1_may_2	shop2_2	-500.00	99999999	0		.	1450	2.500
62	f2_mar_2	shop2_2	-542.83	500	0		.	1450	125.000
63	f2_apr_2	shop2_2	-559.19	500	0		.	1450	500.000
64	f2_may_2	shop2_2	-519.06	500	0		.	1450	122.500
65			1.00	150	0	f1 chips from mar	.	.	20.000
66			0.00	99999999	0	f1 unused chips	.	.	0.000
67			1.00	150	0	f2 chips from mar	.	.	0.000
68			0.00	99999999	0	f2 unused chips	.	.	280.000

OBS	_FCOST_	_RCOST_	_ANUMB_	_TNUMB_	_STATUS_	DIAGONAL	FACTORY	KEY_ID	MTH_MADE
1	42444.00	.	1	1	KEY_ARC BASIC	19	1	production	April
2	0.00	66.150	2	3	LOWERBD NONBASIC	19	1	storage	March
3	0.00	42.580	3	4	LOWERBD NONBASIC	19	1	backorder	May
4	0.00	22.000	4	5	LOWERBD NONBASIC	19	.	f2_to_1	April
5	43625.00	.	15	6	KEY_ARC BASIC	25	1	production	April
6	0.00	94.210	16	8	LOWERBD NONBASIC	25	1	storage	March
7	0.00	.	17	9	NONKEY ARC BASIC	25	1	backorder	May
8	525.00	-1.510	18	10	UPPERBD NONBASIC	25	.	f2_to_1	April
9	43272.83	.	5	1	KEY_ARC BASIC	19	1	production	March
10	672.00	-17.070	6	2	UPPERBD NONBASIC	19	1	backorder	April
11	400.00	-34.750	7	11	UPPERBD NONBASIC	19	.	f2_to_1	March
12	87160.00	-28.343	19	6	UPPERBD NONBASIC	25	1	production	March
13	1152.00	-35.330	20	7	UPPERBD NONBASIC	25	1	backorder	April
14	500.00	-61.060	21	12	UPPERBD NONBASIC	25	.	f2_to_1	March
15	10511.67	.	8	1	KEY_ARC BASIC	19	1	production	May
16	0.00	3.500	9	2	LOWERBD NONBASIC	19	1	storage	April
17	0.00	29.000	10	13	LOWERBD NONBASIC	19	.	f2_to_1	May
18	39655.00	-15.520	22	6	UPPERBD NONBASIC	25	1	production	May
19	0.00	67.680	23	7	LOWERBD NONBASIC	25	1	storage	April
20	0.00	32.060	24	14	LOWERBD NONBASIC	25	.	f2_to_1	May
21	220.00	.	11	2	KEY_ARC BASIC	19	.	f1_to_2	April
22	29952.00	-35.592	12	15	UPPERBD NONBASIC	19	2	production	April
23	0.00	13.400	13	11	LOWERBD NONBASIC	19	2	storage	March
24	0.00	43.980	14	13	LOWERBD NONBASIC	19	2	backorder	May
25	0.00	45.510	25	7	LOWERBD NONBASIC	25	.	f1_to_2	April
26	113594.25	.	26	16	KEY_ARC BASIC	25	2	production	April
27	0.00	43.660	27	12	LOWERBD NONBASIC	25	2	storage	March
28	0.00	57.170	28	14	LOWERBD NONBASIC	25	2	backorder	May
29	0.00	55.750	29	3	LOWERBD NONBASIC	19	.	f1_to_2	March
30	25520.00	.	30	15	KEY_ARC BASIC	19	2	production	March
31	0.00	25.480	31	5	LOWERBD NONBASIC	19	2	backorder	April
32	0.00	104.060	32	8	LOWERBD NONBASIC	25	.	f1_to_2	March
33	118300.00	-23.170	33	16	UPPERBD NONBASIC	25	2	production	March
34	0.00	22.020	34	10	LOWERBD NONBASIC	25	2	backorder	April
35	1840.00	.	35	4	KEY_ARC BASIC	19	.	f1_to_2	May
36	4508.00	22.700	36	15	LOWERBD NONBASIC	19	2	production	May
37	0.00	6.500	37	5	LOWERBD NONBASIC	19	2	storage	April
38	0.00	6.940	38	9	LOWERBD NONBASIC	25	.	f1_to_2	May
39	22221.50	.	39	16	KEY_ARC BASIC	25	2	production	May
40	0.00	46.110	40	10	LOWERBD NONBASIC	25	2	storage	April
41	-48601.42	.	41	3	KEY_ARC BASIC	19	1	sales	March
42	-75000.00	-23.500	42	2	UPPERBD NONBASIC	19	1	sales	April
43	-475.00	.	43	4	NONKEY ARC BASIC	19	1	sales	May
44	-74350.00	-14.500	44	11	UPPERBD NONBASIC	19	2	sales	March
45	-72500.00	-2.500	45	5	UPPERBD NONBASIC	19	2	sales	April
46	0.00	9.000	46	13	LOWERBD NONBASIC	19	2	sales	May
47	0.00	79.150	47	8	LOWERBD NONBASIC	25	1	sales	March
48	0.00	40.420	48	7	LOWERBD NONBASIC	25	1	sales	April
49	-178969.45	.	49	9	KEY_ARC BASIC	25	1	sales	May
50	-283915.00	-9.980	50	12	UPPERBD NONBASIC	25	2	sales	March
51	-28464.98	.	51	10	KEY_ARC BASIC	25	2	sales	April
52	0.00	42.520	52	14	LOWERBD NONBASIC	25	2	sales	May
53	-90685.00	-37.090	53	3	UPPERBD NONBASIC	19	1	sales	March
54	-75000.00	-25.500	54	2	UPPERBD NONBASIC	19	1	sales	April
55	0.00	38.000	55	4	LOWERBD NONBASIC	19	1	sales	May
56	0.00	8.200	56	11	LOWERBD NONBASIC	19	2	sales	March
57	-78000.00	-26.500	57	5	UPPERBD NONBASIC	19	2	sales	April
58	-44850.00	.	58	13	KEY_ARC BASIC	19	2	sales	May
59	-283869.95	.	59	8	KEY_ARC BASIC	25	1	sales	March
60	-134671.60	.	60	7	KEY_ARC BASIC	25	1	sales	April
61	-1250.00	.	61	9	NONKEY ARC BASIC	25	1	sales	May
62	-67853.75	.	62	12	KEY_ARC BASIC	25	2	sales	March
63	-279595.00	-32.020	63	10	UPPERBD NONBASIC	25	2	sales	April
64	-63584.85	.	64	14	KEY_ARC BASIC	25	2	sales	May
65	20.00	.	-2	.	NONKEY BASIC	.	1	chips	
66	0.00	1.617	0	.	LOWERBD NONBASIC	.	1	chips	
67	0.00	2.797	-3	.	LOWERBD NONBASIC	.	2	chips	
68	0.00	.	-1	.	NONKEY BASIC	.	2	chips	
	===========								
	-1295542.74								

Output 4.8.3. DUALOUT=DUAL7

Obs	_node_	_supdem_	_DUAL_	_NNUMB_	_PRED_	_TRAV_	_SCESS_
1	f1_apr_1	.	-100000275.50	2	1	5	2
2	f1_apr_2	.	-100000405.92	7	20	6	2
3	f1_mar_1	.	-100000326.65	3	17	1	20
4	f1_mar_2	.	-100000480.13	8	20	7	1
5	f1_may_1	.	-100000284.00	4	1	13	3
6	f1_may_2	.	-100000356.24	9	18	2	1
7	f2_apr_1	.	-100000286.50	5	2	4	1
8	f2_apr_2	.	-100000383.41	10	16	18	3
9	f2_mar_1	.	-100000281.90	11	15	16	1
10	f2_mar_2	.	-100000399.07	12	20	10	1
11	f2_may_1	.	-100000300.00	13	4	19	2
12	f2_may_2	.	-100000375.30	14	16	20	6
13	fact1_1	1000	-100000193.90	1	3	21	19
14	fact1_2	1000	-100000227.42	6	7	12	1
15	fact2_1	850	-100000193.90	15	21	11	2
16	fact2_2	1500	-100000193.90	16	21	14	10
17	shop1_1	-900	-99999999.00	17	22	3	21
18	shop1_2	-900	-99999841.22	18	10	9	2
19	shop2_1	-900	-100000001.00	19	13	22	1
20	shop2_2	-1450	-99999856.24	20	14	8	5
21		.	0.00	4	4	.	4
22		.	-1.00	2	2	.	.
23		.	-1.62	0	17	.	.
24		.	1.80	3	20	.	.
25		.	0.00	1	1	.	.
26		.	-0.48	5	7	.	.

Obs	_ARCID_	_FLOW_	_FBQ_	_VALUE_	_RHS_	_TYPE_	_row_
1	1	490.000	1	.	.		
2	-60	245.000	15	.	.		
3	-41	148.333	5	.	.		
4	-59	455.000	19	.	.		
5	8	66.667	8	.	.		
6	-49	347.500	22	.	.		
7	11	20.000	11	.	.		
8	26	542.500	25	.	.		
9	30	255.000	29	.	.		
10	-62	125.000	32	.	.		
11	35	115.000	35	.	.		
12	39	87.500	38	.	.		
13	-5	288.333	-1	.	.		
14	-15	200.000	-15	.	.		
15	-67	45.000	-41	.	.		
16	-68	150.000	-41	.	.		
17	0	0.000	41	.	.		
18	51	52.500	47	.	.		
19	58	150.000	53	.	.		
20	64	122.500	59	.	.		
21	.	260.000	.	260	0	GE	CHIP LEFTOVER
22	-2	20.000	.	2600	2600	EQ	FACT1 APL GIZMO
23	43	1.667	.	2615	2615	EQ	FACT1 MAR GIZMO
24	61	2.500	.	3750	3750	EQ	FACT2 APL GIZMO
25	-1	280.000	.	3750	3750	EQ	FACT2 MAR GIZMO
26	17	0.000	.	50	50	LE	TOTAL BACKORDER

The optimal value of the nonarc variable “f2 unused chips” is 280. This means that although there are 3750 chips that can be used at factory 2 in March, only 3470 are used. As the optimal value of “f1 unused chips” is zero, all chips available for production in March at factory 1 are used. The nonarc variable “f2 chips from mar” also has zero optimal value. This means that the April production at factory 2 does not need any chips that could have been held in inventory since March. However, the nonarc variable “f1 chips from mar” has value of 20. Thus, 3490 chips should be ordered for factory 2 in March. Twenty of these chips should be held in inventory until April, then sent to factory 1.

References

Bland, R.G. (1977), “New Finite Pivoting Rules for the Simplex Method”, *Mathematics of Operations Research*, 2 (2) 103-107.

Kearney, T.D. (1990), “A Tutorial on the NETFLOW Procedure in SAS/OR”, *Proceedings of the fifteenth Annual SAS Users Group International Conference*, 97-108.

Kennington, J.L. and Helgason, R.V. (1980), “Algorithms for Networking Programming”, New York: Wiley Interscience, John Wiley & Sons.

Reid, J.K. (1975), “A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases”, *Harwell Report CSS 20*, A.E.R.E., Didcot, Oxfordshire, England.

Roos, C. Terlaky, T. Vial, J.-Ph. (1997) “Theory and Algorithms for Linear Optimization”, John Wiley & Sons.

Ryan, D.M. and Osborne, M.R. (1988), “On the Solution of Highly Degenerate Linear Programmes”, *Mathematical Programming*, 41 385-392.

Wright, S.J. (1996) “Primal-Dual Interior Point Algorithms”, Philadelphia: SIAM

Ye, Y. (1996) “Interior Point Algorithms: Theory and Analysis”, Wiley-Interscience Series in Discrete Mathematics and Optimization.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/OR® User's Guide: Mathematical Programming, Version 8*, Cary, NC: SAS Institute Inc., 1999. 566 pp.

SAS/OR® User's Guide: Mathematical Programming, Version 8

ISBN 1–58025–491–8

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999