Chapter 5

The Linear Programming Solver

Contents

Overview: LP Solver

The OPTMODEL procedure provides a framework for specifying and solving linear programs (LPs). A standard linear program has the following formulation:

$$\begin{array}{rl} \min & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} & \mathbf{A}\mathbf{x} \; \{\geq, =, \leq\} \; \mathbf{b} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{array}$$

where

$\mathbf{x}$	$\in$	$\mathbb{R}^n$	is the vector of decision variables
$\mathbf{A}$	$\in$	$\mathbb{R}^{m \times n}$	is the matrix of constraints
$\mathbf{c}$	$\in$	$\mathbb{R}^n$	is the vector of objective function coefficients
$\mathbf{b}$	$\in$	$\mathbb{R}^m$	is the vector of constraints right-hand sides (RHS)
$\mathbf{l}$	$\in$	$\mathbb{R}^n$	is the vector of lower bounds on variables
$\mathbf{u}$	$\in$	$\mathbb{R}^n$	is the vector of upper bounds on variables

The following LP solvers are available in the OPTMODEL procedure:

- primal simplex solver
- dual simplex solver
- network simplex solver
- interior point solver

The primal and dual simplex solvers implement the two-phase simplex method. In phase I, the solver tries to find a feasible solution. If no feasible solution is found, the LP is infeasible; otherwise, the solver enters phase II to solve the original LP. The network simplex solver extracts a network substructure, solves this using network simplex, and then constructs an advanced basis to feed to either primal or dual simplex. The interior point solver implements a primal-dual predictor-corrector interior point algorithm. If any of the decision variables are constrained to be integer-valued, then the relaxed version of the problem is solved.

Getting Started: LP Solver

The following example illustrates how you can use the OPTMODEL procedure to solve linear programs. Suppose you want to solve the following problem:

$$\begin{array}{rrcrcrcl} \max & x_1 & + & x_2 & + & x_3 & & \\ \text{subject to} & 3x_1 & + & 2x_2 & - & x_3 & \leq & 1 \\ & -2x_1 & - & 3x_2 & + & 2x_3 & \leq & 1 \\ & & & x_1, & & x_2, \; x_3 & \geq & 0 \end{array}$$

You can use the following statements to call the OPTMODEL procedure for solving linear programs:

```
proc optmodel;
   var x{i in 1..3} >= 0;
   max f =     x[1] +   x[2] +    x[3];
   con c1:  3*x[1] + 2*x[2] -    x[3] <= 1;
   con c2: -2*x[1] - 3*x[2] + 2*x[3] <= 1;
   solve with lp / solver = ps presolver = none printfreq = 1;
   print x;
quit;
```

The optimal solution and the optimal objective value are displayed in Figure 5.1.

Figure 5.1 Solution Summary

The OPTMODEL Procedure

Problem Summary

Objective Sense	Maximization
Objective Function	f
Objective Type	Linear
Number of Variables	3
Bounded Above	0
Bounded Below	3
Bounded Below and Above	0
Free	0
Fixed	0
Number of Constraints	2
Linear LE (<=)	2
Linear EQ (=)	0
Linear GE (>=)	0
Linear Range	0
Constraint Coefficients	6

Solution Summary

Solver	Primal Simplex
Objective Function	f
Solution Status	Optimal
Objective Value	8
Iterations	2
Primal Infeasibility	0
Dual Infeasibility	0
Bound Infeasibility	0

[1]	x
1	0
2	3
3	5

The iteration log displaying problem statistics, progress of the solution, and the optimal objective value is shown in Figure 5.2.

Figure 5.2 Log

```
NOTE: The problem has 3 variables (0 free, 0 fixed).
NOTE: The problem has 2 linear constraints (2 LE, 0 EQ, 0 GE, 0 range).
NOTE: The problem has 6 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The OPTLP presolver value NONE is applied.
NOTE: The PRIMAL SIMPLEX solver is called.
                       Objective     Entering     Leaving
      Phase Iteration  Value         Variable     Variable
        2           1      0.500000  x[3]         c2       (S)
        2           2      8.000000  x[2]         c1       (S)
NOTE: Optimal.
NOTE: Objective = 8.
```

Syntax: LP Solver

The following statement is available in the OPTMODEL procedure:

SOLVE WITH LP < / *options* > ;

Functional Summary

Table 5.1 summarizes the list of options available for the SOLVE WITH LP statement, classified by function.

Table 5.1 Options for the LP Solver

Description	Option
Solver Options:	
Enables or disables IIS detection	IIS=
Specifies the type of solver	SOLVER=
Specifies the type of solver called after network simplex	SOLVER2=
Presolve Option:	
Specifies the type of presolve	PRESOLVER=
Control Options:	
Specifies the feasibility tolerance	FEASTOL=
Specifies the maximum number of iterations	MAXITER=
Specifies the upper limit on real time used to solve the problem	MAXTIME=
Specifies the optimality tolerance	OPTTOL=
Specifies the frequency of printing solution progress	PRINTFREQ=

Table 5.1 (continued)

Description	Option
Specifies the detail of solution progress printed in log	PRINTLEVEL2=
Specifies units of CPU time or real time	TIMETYPE=
Simplex Algorithm Options:	
Specifies the type of initial basis	BASIS=
Specifies the type of pricing strategy	PRICETYPE=
Specifies the queue size for determining entering variable	QUEUESIZE=
Enables or disables scaling of the problem	SCALE=
Interior Point Algorithm Options:	
Enables or disables interior crossover (Experimental)	CROSSOVER=
Specifies the stopping criterion based on duality gap	STOP_DG=
Specifies the stopping criterion based on dual infeasibility	STOP_DI=
Specifies the stopping criterion based on primal infeasibility	STOP_PI=

LP Solver Options

This section describes the options recognized by the LP solver. These options can be specified after a forward slash (/) in the SOLVE statement, provided that the LP solver is explicitly specified using a WITH clause.

If the LP solver terminates before reaching an optimal solution, an intermediate solution is available. You can access this solution by using the .sol variable suffix in the OPTMODEL procedure. See the section "Suffixes" on page 121 for details.

Solver Options

IIS=*option* | *num*

specifies whether the LP solver attempts to identify a set of constraints and variables that form an irreducible infeasible set (IIS). Table 5.2 describes the valid values of the IIS= option.

Table 5.2 Values for IIS= Option

num	*option*	**Description**
0	OFF	Disables IIS detection.
1	ON	Enables IIS detection.

If an IIS is found, information about the infeasibilities can be found in the .status values of the constraints and variables. If no IIS is detected, then a solver is called to continue solving the problem. The default value of this option is OFF. The OPTMODEL option PRESOLVER=NONE should be specified when IIS=ON is specified; otherwise, the IIS results can be incomplete. See the section

"Irreducible Infeasible Set" on page 186 for details about the IIS= option. See "Suffixes" on page 121 for details about the .status suffix.

SOLVER=*option*

specifies one of the following LP solvers:

Option	Description
PRIMAL (PS)	Use primal simplex solver.
DUAL (DS)	Use dual simplex solver.
NETWORK (NS)	Use network simplex solver.
ITERATIVE (II)	Use interior point solver.

The valid abbreviated value for each option is indicated in parentheses. By default, the dual simplex solver is used.

SOLVER2=*option*

specifies one of the following LP solvers if SOLVER=NS:

Option	Description
PRIMAL (PS)	Use primal simplex solver (after network simplex).
DUAL (DS)	Use dual simplex solver (after network simplex).

The valid abbreviated value for each option is indicated in parentheses. By default, the LP solver decides which algorithm is best to use after calling the network simplex solver on the extracted network.

Presolve Options

PRESOLVER=*option* **|** *num*

specifies one of the following presolve options:

num	*option*	Description
0	NONE	Disable presolver.
–1	AUTOMATIC	Apply presolver by using default setting.
1	BASIC	Perform basic presolve like removing empty rows, columns, and fixed variables.
2	MODERATE	Perform basic presolve and apply other inexpensive presolve techniques.
3	AGGRESSIVE	Perform moderate presolve and apply other aggressive (but expensive) presolve techniques.

The default option is AUTOMATIC. See the section "Presolve" on page 176 for details.

Control Options

FEASTOL=ϵ

specifies the feasibility tolerance, $\epsilon \in$[1E–9, 1E–4], for determining the feasibility of a variable. The default value is 1E–6.

MAXITER=k

specifies the maximum number of iterations. The value k can be any integer between one and the largest four-byte signed integer, which is $2^{31} - 1$. If you do not specify this option, the procedure does not stop based on the number of iterations performed. For network simplex, this iteration limit corresponds to the solver called after network simplex (either primal or dual simplex).

MAXTIME=k

specifies an upper limit of k seconds of time for the optimization process. The timer used by this option is determined by the value of the TIMETYPE= option. If you do not specify this option, the procedure does not stop based on the amount of time elapsed.

OPTTOL=ϵ

specifies the optimality tolerance, $\epsilon \in$ [1E–9, 1E–4], for declaring optimality. The default value is 1E–6.

PRINTFREQ=k

specifies that the printing of the solution progress to the iteration log is to occur after every k iterations. The print frequency, k, is an integer between zero and the largest four-byte signed integer, which is $2^{31} - 1$.

The value $k = 0$ disables the printing of the progress of the solution. If the primal or dual simplex algorithms are used, the default value of this option is determined dynamically according to the problem size. If the network simplex algorithm is used, the default value of this option is 10,000. If the interior point algorithm is used, the default value of this option is 1.

PRINTLEVEL2=*option* **|** *num*

controls the amount of information displayed in the SAS log by the LP solver, from a short description of presolve information and summary to details at each iteration. Table 5.6 describes the valid values for this option.

Table 5.6 Values for PRINTLEVEL2= Option

num	*option*	**Description**
0	NONE	Turns off all solver-related messages to SAS log.
1	BASIC	Displays a solver summary after stopping.
2	MODERATE	Prints a solver summary and an iteration log by using the interval dictated by the PRINTFREQ= option.
3	AGGRESSIVE	Prints a detailed solver summary and an iteration log by using the interval dictated by the PRINTFREQ= option.

The default value is MODERATE.

TIMETYPE=*option* | *num*

specifies the units of time used by the MAXTIME= option and reported by the PRESOLVE_TIME and SOLUTION_TIME terms in the _OROPTMODEL_ macro variable. Table 5.7 describes the valid values of the TIMETYPE= option.

Table 5.7 Values for TIMETYPE= Option

num	*option*	**Description**
0	CPU	Specifies units of CPU time.
1	REAL	Specifies units of real time.

The "Optimization Statistics" table, an output of PROC OPTMODEL if option PRINTLEVEL=2 is specified in the PROC OPTMODEL statement, also includes the same time units for "Presolver Time" and "Solver Time." The other times (such as "Problem Generation Time") in the "Optimization Statistics" table are always CPU times. The default value of the TIMETYPE= option is CPU.

Simplex Algorithm Options

BASIS=*option* | *num*

specifies the following options for generating an initial basis:

num	*option*	**Description**
0	CRASH	Generate an initial basis by using crash techniques (Maros 2003). The procedure creates a triangular basic matrix consisting of both decision variables and slack variables.
1	SLACK	Generate an initial basis by using all slack variables.
2	WARMSTART	Start the primal and dual simplex solvers with available basis.

The default option for the primal simplex solver is CRASH (0). The default option for the dual simplex solver is SLACK(1). For network simplex, this option has no effect.

PRICETYPE=*option* | *num*

specifies one of the following pricing strategies for the primal and dual simplex solvers:

num	*option*	**Description**
0	HYBRID	Use hybrid Devex and steepest-edge pricing strategies. Available for primal simplex solver only.
1	PARTIAL	Use partial pricing strategy. Optionally, you can specify QUEUESIZE=. Available for primal simplex solver only.
2	FULL	Use the most negative reduced cost pricing strategy.
3	DEVEX	Use Devex pricing strategy.
4	STEEPESTEDGE	Use steepest-edge pricing strategy.

The default pricing strategy for the primal simplex solver is HYBRID and that for the dual simplex solver is STEEPESTEDGE. For the network simplex solver, this option applies only to the solver specified by the SOLVER2= option. See the section "Pricing Strategies for the Primal and Dual Simplex Solvers" on page 176 for details.

QUEUESIZE=*k*

specifies the queue size, $k \in [1, n]$, where n is the number of decision variables. This queue is used for finding an entering variable in the simplex iteration. The default value is chosen adaptively based on the number of decision variables. This option is used only when PRICETYPE=PARTIAL.

SCALE=*option* | *num*

specifies one of the following scaling options:

num	*option*	**Description**
0	NONE	Disable scaling.
–1	AUTOMATIC	Automatically apply scaling procedure if necessary.

The default option is AUTOMATIC.

Interior Point Algorithm Options

CROSSOVER=*option* | *num*

specifies whether to convert the interior point solution to a basic simplex solution. The values of this option are: *Experimental*

num	*option*	**Description**
0	OFF	Disable crossover.
1	ON	Apply the crossover algorithm to the interior point solution.

If the interior point algorithm terminates with a solution, the crossover algorithm uses the interior point solution to create an initial basic solution. After performing primal fixing and dual fixing, the crossover algorithm calls a simplex algorithm to locate an optimal basic solution. The default value of the CROSSOVER= option is OFF.

STOP_DG=δ

specifies the desired relative duality gap, $\delta \in$[1E–9, 1E–4]. This is the relative difference between the primal and dual objective function values and is the primary solution quality parameter. The default value is 1E–6. See the section "The Interior Point Algorithm" on page 177 for details.

STOP_DI=β

specifies the maximum allowed relative dual constraints violation, $\beta \in$ [1E–9, 1E–4]. The default value is 1E–6. See the section "The Interior Point Algorithm" on page 177 for details.

STOP_PI=α

specifies the maximum allowed relative bound and primal constraints violation, $\alpha \in$[1E–9, 1E–4]. The default value is 1E–6. See the section "The Interior Point Algorithm" on page 177 for details.

Details: LP Solver

Presolve

Presolve in the simplex LP solvers of PROC OPTMODEL uses a variety of techniques to reduce the problem size, improve numerical stability, and detect infeasibility or unboundedness (Andersen and Andersen 1995; Gondzio 1997). During presolve, redundant constraints and variables are identified and removed. Presolve can further reduce the problem size by substituting variables. Variable substitution is a very effective technique, but it might occasionally increase the number of nonzero entries in the constraint matrix.

In most cases, using presolve is very helpful in reducing solution times. You can enable presolve at different levels or disable it by specifying the PRESOLVER= option.

Pricing Strategies for the Primal and Dual Simplex Solvers

Several pricing strategies for the primal and dual simplex solvers are available. Pricing strategies determine which variable enters the basis at each simplex pivot. These can be controlled by specifying the PRICETYPE= option.

The primal simplex solver has the following five pricing strategies:

PARTIAL
: scans a queue of decision variables to find an entering variable. You can optionally specify the QUEUESIZE= option to control the length of this queue.

FULL
: uses Dantzig's most violated reduced cost rule (Dantzig 1963). It compares the reduced cost of all decision variables, and selects the variable with the most violated reduced cost as the entering variable.

DEVEX
: implements the Devex pricing strategy developed by Harris (1973).

STEEPESTEDGE
: uses the steepest-edge pricing strategy developed by Forrest and Goldfarb (1992).

HYBRID
: uses a hybrid of the Devex and steepest-edge pricing strategies.

The dual simplex solver has only three pricing strategies available: FULL, DEVEX, and STEEPESTEDGE.

The Network Simplex Algorithm

The network simplex solver in PROC OPTMODEL attempts to leverage the speed of the network simplex algorithm to more efficiently solve linear programs by using the following process:

1. It heuristically extracts the largest possible network substructure from the original problem.

2. It uses the network simplex algorithm to solve for an optimal solution to this substructure.

3. It uses this solution to construct an advanced basis to warm-start either the primal or dual simplex solver on the original linear programming problem.

The network simplex algorithm is a specialized version of the simplex algorithm that uses spanning-tree bases to more efficiently solve linear programming problems that have a pure network form. Such LPs can be modeled using a formulation over a directed graph, as a minimum-cost flow problem. Let $G = (N, A)$ be a directed graph, where N denotes the nodes and A denotes the arcs of the graph. The decision variable x_{ij} denotes the amount of flow sent between node i and node j. The cost per unit of flow on the arcs is designated by c_{ij}, and the amount of flow sent across each arc is bounded to be within $[l_{ij}, u_{ij}]$. The demand (or supply) at each node is designated as b_i, where $b_i > 0$ denotes a supply node and $b_i < 0$ denotes a demand node. The corresponding linear programming problem is as follows:

$$
\begin{array}{rcccl}
\min & \sum_{(i,j)\in A} c_{ij} x_{ij} & & & \\
\text{subject to} & \sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} & = & b_i & \forall i \in N \\
 & x_{ij} & \leq & u_{ij} & \forall (i,j) \in A \\
 & x_{ij} & \geq & l_{ij} & \forall (i,j) \in A.
\end{array}
$$

The network simplex algorithm used in PROC OPTMODEL is the primal network simplex algorithm. This algorithm finds the optimal primal feasible solution and a dual solution that satisfies complementary slackness. Sometimes the directed graph G is disconnected. In this case, the problem can be decomposed into its weakly connected components, and each minimum-cost flow problem can be solved separately. After solving each component, the optimal basis for the network substructure is augmented with the non-network variables and constraints from the original problem. This advanced basis is then used as a starting point for the primal or dual simplex method. The solver automatically selects the solver to use after network simplex. However, you can override this selection with the SOLVER2= option.

The network simplex algorithm can be more efficient than the other solvers on problems that have a large network substructure. The size of this network structure can be seen in the log.

The Interior Point Algorithm

The interior point LP solver in PROC OPTMODEL implements an infeasible primal-dual predictor-corrector interior point algorithm. To illustrate the algorithm and the concepts of duality and dual infeasibility, consider the following LP formulation (the primal):

$$
\begin{array}{rl}
\min & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\
\text{subject to} & \mathbf{A}\mathbf{x} \geq \mathbf{b} \\
 & \mathbf{x} \geq \mathbf{0}
\end{array}
$$

The corresponding dual is as follows:

$$
\begin{array}{rlcccl}
\max & \mathbf{b}^{\mathrm{T}}\mathbf{y} & & & & \\
\text{subject to} & \mathbf{A}^{\mathrm{T}}\mathbf{y} & + & \mathbf{w} & = & \mathbf{c} \\
 & & & \mathbf{y} & \geq & \mathbf{0} \\
 & & & \mathbf{w} & \geq & \mathbf{0}
\end{array}
$$

where $\mathbf{y} \in \mathbb{R}^m$ refers to the vector of dual variables and $\mathbf{w} \in \mathbb{R}^n$ refers to the vector of dual slack variables.

The dual makes an important contribution to the certificate of optimality for the primal. The primal and dual constraints combined with complementarity conditions define the first-order optimality conditions, also known as KKT (Karush-Kuhn-Tucker) conditions, which can be stated as follows:

$$\begin{array}{rcll} \mathbf{Ax} - \mathbf{s} & = & \mathbf{b} & \text{(Primal Feasibility)} \\ \mathbf{A}^{\mathrm{T}}\mathbf{y} + \mathbf{w} & = & \mathbf{c} & \text{(Dual Feasibility)} \\ \mathbf{WXe} & = & \mathbf{0} & \text{(Complementarity)} \\ \mathbf{SYe} & = & \mathbf{0} & \text{(Complementarity)} \\ \mathbf{x},\, \mathbf{y},\, \mathbf{w},\, \mathbf{s} & \geq & \mathbf{0} & \end{array}$$

where $\mathbf{e} \equiv (1, \ldots, 1)^{\mathrm{T}}$ of appropriate dimension and $\mathbf{s} \in \mathbb{R}^m$ is the vector of primal *slack* variables.

NOTE: Slack variables (the s vector) are automatically introduced by the solver when necessary; it is therefore recommended that you not introduce any slack variables explicitly. This enables the solver to handle slack variables much more efficiently.

The letters **X**, **Y**, **W**, and **S** denote matrices with corresponding x, y, w, and s on the main diagonal and zero elsewhere, as in the following example:

$$\mathbf{X} \equiv \begin{bmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_n \end{bmatrix}$$

If $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{w}^*, \mathbf{s}^*)$ is a solution of the previously defined system of equations representing the KKT conditions, then $\mathbf{x}^*$ is also an optimal solution to the original LP model.

At each iteration the interior point algorithm solves a large, sparse system of linear equations as follows:

$$\begin{bmatrix} \mathbf{Y}^{-1}\mathbf{S} & \mathbf{A} \\ \mathbf{A}^{\mathrm{T}} & -\mathbf{X}^{-1}\mathbf{W} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{y} \\ \Delta\mathbf{x} \end{bmatrix} = \begin{bmatrix} \Xi \\ \Theta \end{bmatrix}$$

where $\Delta\mathbf{x}$ and $\Delta\mathbf{y}$ denote the vector of *search directions* in the primal and dual spaces, respectively; Θ and Ξ constitute the vector of the right-hand sides.

The preceding system is known as the reduced KKT system. The interior point solver uses a preconditioned quasi-minimum residual algorithm to solve this system of equations efficiently.

An important feature of the interior point solver is that it takes full advantage of the sparsity in the constraint matrix, thereby enabling it to efficiently solve large-scale linear programs.

The interior point algorithm works simultaneously in the primal and dual spaces. It attains optimality when both primal and dual feasibility are achieved and when complementarity conditions hold. Therefore it is of interest to observe the following four measures:

- Relative primal infeasibility measure α:

$$\alpha = \frac{\|\mathbf{Ax} - \mathbf{b} - \mathbf{s}\|_2}{\|\mathbf{b}\|_2 + 1}$$

- Relative dual infeasibility measure β:

$$\beta = \frac{\|\mathbf{c} - \mathbf{A}^{\mathrm{T}}\mathbf{y} - \mathbf{w}\|_2}{\|\mathbf{c}\|_2 + 1}$$

- Relative duality gap δ:

$$\delta = \frac{|\mathbf{c}^{\mathrm{T}}\mathbf{x} - \mathbf{b}^{\mathrm{T}}\mathbf{y}|}{|\mathbf{c}^{\mathrm{T}}\mathbf{x}| + 1}$$

- Absolute complementarity γ:

$$\gamma = \sum_{i=1}^{n} x_i w_i + \sum_{i=1}^{m} y_i s_i$$

where $\|v\|_2$ is the Euclidean norm of the vector v. These measures are displayed in the iteration log.

Macro Variable _OROPTMODEL_

The OPTMODEL procedure always creates and initializes a SAS macro called _OROPTMODEL_. This variable contains a character string. After each PROC OROPTMODEL run, you can examine this macro by specifying `%put &_OROPTMODEL_;` and check the execution of the most recently invoked solver from the value of the macro variable. The various terms of the variable after the LP solver is called are interpreted as follows.

STATUS
indicates the solver status at termination. It can take one of the following values:

OK	The solver terminated normally.
SYNTAX_ERROR	Incorrect syntax was used.
DATA_ERROR	The input data were inconsistent.
OUT_OF_MEMORY	Insufficient memory was allocated to the procedure.
IO_ERROR	A problem occurred in reading or writing data.
SEMANTIC_ERROR	An evaluation error, such as an invalid operand type, occurred.
ERROR	The status cannot be classified into any of the preceding categories.

SOLUTION_STATUS
indicates the solution status at termination. It can take one of the following values:

OPTIMAL	The solution is optimal.
CONDITIONAL_OPTIMAL	The solution is optimal, but some infeasibilities (primal, dual or bound) exceed tolerances due to scaling or pre-processing.
INFEASIBLE	The problem is infeasible.
UNBOUNDED	The problem is unbounded.

INFEASIBLE_OR_UNBOUNDED	The problem is infeasible or unbounded.
BAD_PROBLEM_TYPE	The problem type is unsupported by the solver.
ITERATION_LIMIT_REACHED	The maximum allowable number of iterations was reached.
TIME_LIMIT_REACHED	The solver reached its execution time limit.
FUNCTION_CALL_LIMIT_REACHED	The solver reached its limit on function evaluations.
FAILED	The solver failed to converge, possibly due to numerical issues.

When SOLUTION_STATUS has a value of OPTIMAL, CONDITIONAL_OPTIMAL, ITERATION_LIMIT_REACHED, or TIME_LIMIT_REACHED, all terms of the _OROPTMODEL_ macro variable are present; for other values of SOLUTION_STATUS, some terms do not appear.

OBJECTIVE
indicates the objective value obtained by the solver at termination.

PRIMAL_INFEASIBILITY
indicates, for the primal simplex and dual simplex solvers, the maximum (absolute) violation of the primal constraints by the primal solution. For the interior point solver, this term indicates the relative violation of the primal constraints by the primal solution.

DUAL_INFEASIBILITY
indicates, for the primal simplex and dual simplex solvers, the maximum (absolute) violation of the dual constraints by the dual solution. For the interior point solver, this term indicates the relative violation of the dual constraints by the dual solution.

BOUND_INFEASIBILITY
indicates, for the primal simplex and dual simplex solvers, the maximum (absolute) violation of the lower or upper bounds by the primal solution. For the interior point solver, this term indicates the relative violation of the lower or upper bounds by the primal solution.

DUALITY_GAP
indicates the (relative) duality gap. This term appears only if the option SOLVER= ITERATIVE is specified in the SOLVE statement.

COMPLEMENTARITY
indicates the (absolute) complementarity. This term appears only if the option SOLVER= ITERATIVE is specified in the SOLVE statement.

ITERATIONS
indicates the number of iterations taken to solve the problem. When the network simplex solver is used, this term indicates the number of network simplex iterations taken to solve the network relaxation. When crossover is enabled, this term indicates the number of interior point iterations taken to solve the problem.

ITERATIONS2
indicates the number of simplex iterations performed by the secondary solver. The network simplex solver selects the secondary solver automatically unless a value has been specified for the SOLVER2=

option. When crossover is enabled, the secondary solver is selected automatically. This term appears only if the network simplex solver is used or if crossover is enabled.

PRESOLVE_TIME
indicates the time (in seconds) used in preprocessing.

SOLUTION_TIME
indicates the time (in seconds) taken to solve the problem, including preprocessing time.

NOTE: The time reported in PRESOLVE_TIME and SOLUTION_TIME is either CPU time (default) or real time. The type is determined by the TIMETYPE= option.

When SOLUTION_STATUS has a value of OPTIMAL, CONDITIONAL_OPTIMAL, ITERATION_LIMIT_REACHED, or TIME_LIMIT_REACHED, all terms of the _OROPTMODEL_ macro variable are present; for other values of SOLUTION_STATUS, some terms do not appear.

Iteration Log for the Primal and Dual Simplex Solvers

The primal and dual simplex solvers implement a two-phase simplex algorithm. Phase I finds a feasible solution, which phase II improves to an optimal solution.

When the PRINTFREQ= option has a value of 1, the following information is printed in the iteration log:

Phase	indicates whether the solver is in phase I or phase II of the simplex method.
Iteration	indicates the iteration number.
Objective Value	indicates the current amount of infeasibility in phase I and the objective value of the current solution in phase II.
Entering Variable	indicates the entering pivot variable. A slack variable entering the basis is indicated by the corresponding row name followed by '(S)'. If the entering nonbasic variable has distinct, finite lower and upper bounds, then a "bound swap" takes place. In other words, if the entering variable is at its upper bound, then it is "flipped" to its lower bound and is indicated in the log as "To lower."
Leaving Variable	indicates the leaving pivot variable. A slack variable leaving the basis is indicated by the corresponding row name followed by '(S)'.

When the PRINTFREQ= option is omitted or specified with a value larger than 1, only the phase, iteration, and objective value information is printed in the iteration log.

The behavior of objective values in the iteration log depends on both the current phase and the chosen solver. In phase I, both simplex methods have artificial objective values that decrease to 0 when a feasible solution is found. For the dual simplex method, phase II maintains a dual feasible solution, so a minimization problem has increasing objective values in the iteration log. For the primal simplex method, phase II maintains a primal feasible solution, so a minimization problem has decreasing objective values in the iteration log.

During the solution process, some elements of the LP model might be perturbed to improve performance. After reaching optimality for the perturbed problem, the LP solver solves the original problem by using the optimal basis for the perturbed problem. This can occasionally cause the primal or dual simplex solver to repeat phase I and phase II in several passes.

Iteration Log for the Network Simplex Solver

After finding the embedded network and formulating the appropriate relaxation, the network simplex solver uses a primal network simplex algorithm. In the case of a connected network, with one (weakly connected) component, the log will show the progress of the simplex algorithm. The following information is displayed in the iteration log:

Iteration	indicates the iteration number.
PrimalObj	indicates the primal objective value of the current solution.
Primal Infeas	indicates the maximum primal infeasibility of the current solution.
Time	indicates the time spent on the current component by network simplex.

The frequency of the simplex iteration log is controlled by the PRINTFREQ= option. The default value of the PRINTFREQ= option is 10,000.

If the network relaxation is disconnected, the information in the iteration log shows progress at the component level. The following information is displayed in the iteration log:

Component	indicates the component number being processed.
Nodes	indicates the number of nodes in this component.
Arcs	indicates the number of arcs in this component.
Iterations	indicates the number of simplex iterations needed to solve this component.
Time	indicates the time spent so far in network simplex.

The frequency of the component iteration log is controlled by the PRINTFREQ= option. In this case, the default value of the PRINTFREQ= option is determined by the size of the network.

The PRINTLEVEL2= option adjusts the amount of detail shown. By default, PRINTLEVEL2=MODERATE and reports as in the preceding description. If PRINTLEVEL2=NONE, no information is shown. If PRINTLEVEL2=BASIC, the only information shown is a summary of the network relaxation and the time spent solving the relaxation. If PRINTLEVEL2=AGGRESSIVE, in the case of one component, the log displays as in the preceding description; in the case of multiple components, for each component, a separate simplex iteration log is displayed.

Iteration Log for the Interior Point Solver

The interior point solver implements an infeasible primal-dual predictor-corrector interior point algorithm. The following information is displayed in the iteration log:

Iter	indicates the iteration number
Complement	indicates the (absolute) complementarity
Duality Gap	indicates the (relative) duality gap
Primal Infeas	indicates the (relative) primal infeasibility measure
Bound Infeas	indicates the (relative) bound infeasibility measure
Dual Infeas	indicates the (relative) dual infeasibility measure

If the sequence of solutions converges to an optimal solution of the problem, you should see all columns in the iteration log converge to zero or very close to zero. If they do not, it can be the result of insufficient iterations being performed to reach optimality. In this case, you might need to increase the value specified in the option MAXITER= or MAXTIME=. If the complementarity and/or the duality gap do not converge, the problem might be infeasible or unbounded. If the infeasibility columns do not converge, the problem might be infeasible.

Problem Statistics

Optimizers can encounter difficulty when solving poorly formulated models. Information about data magnitude provides a simple gauge to determine how well a model is formulated. For example, a model whose constraint matrix contains one very large entry (on the order of 10^9) can cause difficulty when the remaining entries are single-digit numbers. The PRINTLEVEL=2 option in the OPTMODEL procedure causes the ODS table “ProblemStatistics” to be generated when the LP solver is called. This table provides basic data magnitude information that enables you to improve the formulation of your models.

The example output in Figure 5.3 demonstrates the contents of the ODS table “ProblemStatistics.”

Figure 5.3 ODS Table ProblemStatistics

```
                    The OPTMODEL Procedure

                      Problem Statistics

Number of Constraint Matrix Nonzeros                   6
Maximum Constraint Matrix Coefficient                  3
Minimum Constraint Matrix Coefficient                  1
Average Constraint Matrix Coefficient       2.1666666667

Number of Objective Nonzeros                           3
Maximum Objective Coefficient                          1
Minimum Objective Coefficient                          1
Average Objective Coefficient                          1

Number of RHS Nonzeros                                 2
Maximum RHS                                            1
Minimum RHS                                            1
Average RHS                                            1

Maximum Number of Nonzeros per Column                  2
Minimum Number of Nonzeros per Column                  2
Average Number of Nonzeros per Column                  2

Maximum Number of Nonzeros per Row                     3
Minimum Number of Nonzeros per Row                     3
Average Number of Nonzeros per Row                     3
```

Data Magnitude and Variable Bounds

Extremely large numerical values might cause computational difficulties for the LP solver, but the occurrence of such difficulties is hard to predict. For this reason, the LP solver issues a data error message whenever it detects model data that exceeds a specific threshold number. The value of the threshold number depends on your operating environment and is printed in the log as part of the data error message.

The following conditions produce a data error:

- The absolute value of an objective coefficient, constraint coefficient, or range (difference between the upper and lower bounds on a constraint) is greater than the threshold number.
- A variable's lower bound, a $\geq$ or $=$ constraint's right-hand side, or a range constraint's lower bound is greater than the threshold number.
- A variable's upper bound, a $\leq$ or $=$ constraint's right-hand side, or a range constraint's upper bound is smaller than the negative threshold number.

If a variable's upper bound is larger than 1E20, then the LP solver treats the bound as ∞. Similarly, if a variable's lower bound is smaller than –1E20, then the LP solver treats the bound as $-\infty$.

Variable and Constraint Status

Upon termination of the LP solver, the .status suffix of each decision variable and constraint stores information about the status of that variable or constraint. For more information about suffixes in the OPTMODEL procedure, see the section "Suffixes" on page 121.

Variable Status

The .status suffix of a decision variable specifies the status of that decision variable. The suffix can take one of the following values:

- B basic variable
- L nonbasic variable at its lower bound
- U nonbasic variable at its upper bound
- F free variable
- I LP model infeasible (all decision variables have .status equal to I)

For the interior point solver with IIS= OFF, .status is blank.

The following values can appear only if IIS= ON. See the section "Irreducible Infeasible Set" on page 186 for details.

- I_L the lower bound of the variable is violated
- I_U the upper bound of the variable is violated
- I_F the fixed bound of the variable is violated

Constraint Status

The .status suffix of a constraint specifies the status of the slack variable for that constraint. The suffix can take one of the following values:

- B basic variable
- L nonbasic variable at its lower bound
- U nonbasic variable at its upper bound
- F free variable
- I LP model infeasible (all decision variables have .status equal to I)

The following values can appear only if option IIS= ON. See the section "Irreducible Infeasible Set" on page 186 for details.

I_L the "GE" ($\geq$) condition of the constraint is violated

I_U the "LE" ($\leq$) condition of the constraint is violated

I_F the "EQ" ($=$) condition of the constraint is violated

Irreducible Infeasible Set

For a linear programming problem, an irreducible infeasible set (IIS) is an infeasible subset of constraints and variable bounds that will become feasible if any single constraint or variable bound is removed. It is possible to have more than one IIS in an infeasible LP. Identifying an IIS can help to isolate the structural infeasibility in an LP.

The IIS=ON option directs the LP solver to search for an IIS in a given LP. The OPTMODEL option PRESOLVER=NONE should be specified when IIS=ON is specified; otherwise, the IIS results can be incomplete. The LP presolver is not applied to the problem during the IIS search. If the LP solver detects an IIS, it updates the .status suffix of the decision variables and constraints, then stops. Otherwise, the problem is sent on to the LP presolver, followed by the specified solver.

The IIS= option can add special values to the .status suffixes of variables and constraints. (See the section "Variable and Constraint Status" on page 185 for more information.) For constraints, a status of "I_L", "I_U", or "I_F" indicates, respectively, the "GE" ($\geq$), "LE" ($\leq$), or "EQ" ($=$) condition is violated. For range constraints, a status of "I_L" or "I_U" indicates, respectively, that the lower or upper bound of the constraint is violated. For variables, a status of "I_L", "I_U", or "I_F" indicates, respectively, the lower, upper, or fixed bound of the variable is violated. From this information, you can identify names of the constraints (variables) in the IIS as well as the corresponding bound where infeasibility occurs.

Making any one of the constraints or variable bounds in the IIS nonbinding will remove the infeasibility from the IIS. In some cases, changing a right-hand side or bound by a finite amount will remove the infeasibility; however, the only way to guarantee removal of the infeasibility is to set the appropriate right-hand side or bound to ∞ or $-\infty$. Since it is possible for an LP to have multiple irreducible infeasible sets, simply removing the infeasibility from one set might not make the entire problem feasible.

See Example 5.4 for an example demonstrating the use of the IIS= option in locating and removing infeasibilities.

Examples: LP Solver

Example 5.1: Diet Problem

Consider the problem of diet optimization. There are six different foods: bread, milk, cheese, potato, fish, and yogurt. The cost and nutrition values per unit are displayed in Table 5.12.

Table 5.12 Cost and Nutrition Values

	Bread	Milk	Cheese	Potato	Fish	Yogurt
Cost	2.0	3.5	8.0	1.5	11.0	1.0
Protein, g	4.0	8.0	7.0	1.3	8.0	9.2
Fat, g	1.0	5.0	9.0	0.1	7.0	1.0
Carbohydrates, g	15.0	11.7	0.4	22.6	0.0	17.0
Calories	90	120	106	97	130	180

The following SAS code creates the data set fooddata of Table 5.12:

```
data fooddata;
infile datalines;
input    name $  cost  prot  fat  carb  cal;
datalines;
         Bread   2     4     1    15    90
         Milk    3.5   8     5    11.7  120
         Cheese  8     7     9    0.4   106
         Potato  1.5   1.3   0.1  22.6  97
         Fish    11    8     7    0     130
         Yogurt  1     9.2   1    17    180
;
```

The objective is to find a minimum-cost diet that contains at least 300 calories, not more than 10 grams of protein, not less than 10 grams of carbohydrates, and not less than 8 grams of fat. In addition, the diet should contain at least 0.5 unit of fish and no more than 1 unit of milk.

You can model the problem and solve it by using PROC OPTMODEL as follows:

```
proc optmodel;
   /* declare index set */
   set<str> FOOD;

   /* declare variables */
   var diet{FOOD} >= 0;

   /* objective function */
   num cost{FOOD};
   min f=sum{i in FOOD}cost[i]*diet[i];
```

```
/* constraints */
num prot{FOOD};
num fat{FOOD};
num carb{FOOD};
num cal{FOOD};
num min_cal, max_prot, min_carb, min_fat;
con cal_con: sum{i in FOOD}cal[i]*diet[i] >= 300;
con prot_con: sum{i in FOOD}prot[i]*diet[i] <= 10;
con carb_con: sum{i in FOOD}carb[i]*diet[i] >= 10;
con fat_con: sum{i in FOOD}fat[i]*diet[i] >= 8;

/* read parameters */
read data fooddata into FOOD=[name] cost prot fat carb cal;

/* bounds on variables */
diet['Fish'].lb = 0.5;
diet['Milk'].ub = 1.0;

/* solve and print the optimal solution */
solve with lp/printfreq=1; /* print each iteration to log */
print diet;
```

The optimal solution and the optimal objective value are displayed in Output 5.1.1.

Output 5.1.1 Optimal Solution to the Diet Problem

The OPTMODEL Procedure

Problem Summary

Objective Sense	Minimization
Objective Function	f
Objective Type	Linear
Number of Variables	6
Bounded Above	0
Bounded Below	5
Bounded Below and Above	1
Free	0
Fixed	0
Number of Constraints	4
Linear LE (<=)	1
Linear EQ (=)	0
Linear GE (>=)	3
Linear Range	0
Constraint Coefficients	23

Output 5.1.1 *continued*

```
                    Solution Summary

          Solver                     Dual Simplex
          Objective Function                    f
          Solution Status                 Optimal
          Objective Value            12.081337881
          Iterations                            4

          Primal Infeasibility       8.881784E-16
          Dual Infeasibility                    0
          Bound Infeasibility                   0

                   [1]           diet

                   Bread     0.000000
                   Cheese    0.449499
                   Fish      0.500000
                   Milk      0.053599
                   Potato    1.865168
                   Yogurt    0.000000
```

Example 5.2: Reoptimizing the Diet Problem Using BASIS=WARMSTART

After an LP is solved, you might want to change a set of the parameters of the LP and solve the problem again. This can be done efficiently in PROC OPTMODEL. The warm start technique uses the optimal solution of the solved LP as a starting point and solves the modified LP problem faster than it can be solved again from scratch. This example illustrates reoptimizing the diet problem described in Example 5.1.

Assume the optimal solution is found by the SOLVE statement. Instead of quitting the OPTMODEL procedure, you can continue to solve several variations of the original problem.

Suppose the cost of cheese increases from 8 to 10 per unit and the cost of fish decreases from 11 to 7 per serving unit. You can change the parameters and solve the modified problem by submitting the following code:

```
cost['Cheese']=10; cost['Fish']=7;
solve with lp/presolver=none
               basis=warmstart
               solver=ps
               printfreq=1;
print diet;
```

Note that the primal simplex solver is preferred because the primal solution to the last-solved LP is still feasible for the modified problem in this case. The solution is shown in Output 5.2.1.

Output 5.2.1 Optimal Solution to the Diet Problem with Modified Objective Function

```
                    The OPTMODEL Procedure

                       Problem Summary

               Objective Sense            Minimization
               Objective Function                    f
               Objective Type                   Linear

               Number of Variables                   6
               Bounded Above                         0
               Bounded Below                         5
               Bounded Below and Above               1
               Free                                  0
               Fixed                                 0

               Number of Constraints                 4
               Linear LE (<=)                        1
               Linear EQ (=)                         0
               Linear GE (>=)                        3
               Linear Range                          0

               Constraint Coefficients              23

                       Solution Summary

                Solver                  Primal Simplex
                Objective Function                   f
                Solution Status                Optimal
                Objective Value            10.980335514
                Iterations                           4

                Primal Infeasibility      8.881784E-16
                Dual Infeasibility                   0
                Bound Infeasibility                  0

                          [1]           diet

                          Bread     0.000000
                          Cheese    0.449499
                          Fish      0.500000
                          Milk      0.053599
                          Potato    1.865168
                          Yogurt    0.000000
```

The following iteration log indicates that it takes the LP solver no more iterations to solve the modified problem by using BASIS=WARMSTART, since the optimal solution to the original problem remains optimal after the objective function is changed.

Output 5.2.2 Log

```
NOTE: There were 6 observations read from the data set WORK.FOODDATA.
NOTE: The problem has 6 variables (0 free, 0 fixed).
NOTE: The problem has 4 linear constraints (1 LE, 0 EQ, 3 GE, 0 range).
NOTE: The problem has 23 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: No basis information is available.  The BASIS=WARMSTART option is ignored.
NOTE: The OPTLP presolver value NONE is applied.
NOTE: The PRIMAL SIMPLEX solver is called.
                       Objective     Entering        Leaving
      Phase Iteration  Value         Variable        Variable
        1           1      2.333259  diet[Cheese]    prot_con    (S)
        2           2     11.340909  diet[Potato]    fat_con     (S)
        2           3     11.065455  diet[Yogurt]    cal_con     (S)
        2           4     10.980336  diet[Milk]      diet[Yogurt](S)
NOTE: Optimal.
NOTE: Objective = 10.9803355.
```

Next, restore the original coefficients of the objective function and consider the case that you need a diet that supplies at least 150 calories. You can change the parameters and solve the modified problem by submitting the following code:

```
cost['Cheese']=8; cost['Fish']=11;cal_con.lb=150;
solve with lp/presolver=none
              basis=warmstart
              solver=ds
              printfreq=1;
print diet;
```

Note that the dual simplex solver is preferred because the dual solution to the last-solved LP is still feasible for the modified problem in this case. The solution is shown in Output 5.2.3.

Output 5.2.3 Optimal Solution to the Diet Problem with Modified RHS

```
                    The OPTMODEL Procedure

                       Problem Summary

           Objective Sense              Minimization
           Objective Function                      f
           Objective Type                     Linear

           Number of Variables                     6
           Bounded Above                           0
           Bounded Below                           5
           Bounded Below and Above                 1
           Free                                    0
           Fixed                                   0

           Number of Constraints                   4
           Linear LE (<=)                          1
           Linear EQ (=)                           0
           Linear GE (>=)                          3
           Linear Range                            0

           Constraint Coefficients                23

                       Solution Summary

             Solver                    Dual Simplex
             Objective Function                   f
             Solution Status                Optimal
             Objective Value           9.1744131985
             Iterations                           3

             Primal Infeasibility                 0
             Dual Infeasibility                   0
             Bound Infeasibility                  0

                      [1]          diet

                      Bread     0.00000
                      Cheese    0.18481
                      Fish      0.50000
                      Milk      0.56440
                      Potato    0.14702
                      Yogurt    0.00000
```

The following iteration log indicates that it takes the LP solver just one more phase II iteration to solve the modified problem by using BASIS=WARMSTART.

Output 5.2.4 Log

```
NOTE: There were 6 observations read from the data set WORK.FOODDATA.
NOTE: The problem has 6 variables (0 free, 0 fixed).
NOTE: The problem has 4 linear constraints (1 LE, 0 EQ, 3 GE, 0 range).
NOTE: The problem has 23 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: No basis information is available.  The BASIS=WARMSTART option is ignored.
NOTE: The OPTLP presolver value NONE is applied.
NOTE: The DUAL SIMPLEX solver is called.
                      Objective     Entering        Leaving
      Phase Iteration Value         Variable        Variable
        2           1      8.650000  diet[Milk]      fat_con      (S)
        2           2      8.925676  diet[Cheese]    prot_con     (S)
        2           3      9.174413  diet[Potato]    carb_con     (S)
NOTE: Optimal.
NOTE: Objective = 9.1744132.
```

Next, restore the original constraint on calories and consider the case that you need a diet that supplies no more than 550 mg of sodium per day. The following row is appended to Table 5.12.

	Bread	Milk	Cheese	Potato	Fish	Yogurt
sodium, mg	148	122	337	186	56	132

You can change the parameters, add the new constraint, and solve the modified problem by submitting the following code:

```
cal_con.lb=300;
num sod{FOOD}=[148 122 337 186 56 132];
con sodium: sum{i in FOOD}sod[i]*diet[i] <= 550;
solve with lp/presolver=none
              basis=warmstart
              printfreq=1;
print diet;
```

The solution is shown in Output 5.2.5.

Output 5.2.5 Optimal Solution to the Diet Problem with Additional Constraint

```
                    The OPTMODEL Procedure

                       Problem Summary

           Objective Sense            Minimization
           Objective Function                    f
           Objective Type                   Linear

           Number of Variables                   6
           Bounded Above                         0
           Bounded Below                         5
           Bounded Below and Above               1
           Free                                  0
           Fixed                                 0

           Number of Constraints                 5
           Linear LE (<=)                        2
           Linear EQ (=)                         0
           Linear GE (>=)                        3
           Linear Range                          0

           Constraint Coefficients              29

                       Solution Summary

             Solver                   Dual Simplex
             Objective Function                  f
             Solution Status               Optimal
             Objective Value          12.081337881
             Iterations                          4

             Primal Infeasibility                0
             Dual Infeasibility                  0
             Bound Infeasibility                 0

                      [1]           diet

                      Bread     0.000000
                      Cheese    0.449499
                      Fish      0.500000
                      Milk      0.053599
                      Potato    1.865168
                      Yogurt    0.000000
```

The following iteration log indicates that it takes the LP solver no more iterations to solve the modified problem by using the BASIS=WARMSTART option, since the optimal solution to the original problem remains optimal after one more constraint is added.

Output 5.2.6 Log

```
NOTE: There were 6 observations read from the data set WORK.FOODDATA.
NOTE: The problem has 6 variables (0 free, 0 fixed).
NOTE: The problem has 5 linear constraints (2 LE, 0 EQ, 3 GE, 0 range).
NOTE: The problem has 29 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: No basis information is available.  The BASIS=WARMSTART option is ignored.
NOTE: The OPTLP presolver value NONE is applied.
NOTE: The DUAL SIMPLEX solver is called.
                          Objective     Entering         Leaving
      Phase Iteration     Value         Variable         Variable
        2           1       8.650000    diet[Milk]       fat_con      (S)
        2           2       8.894231    diet[Yogurt]     cal_con      (S)
        2           3      11.552207    diet[Potato]     prot_con     (S)
        2           4      12.081338    diet[Cheese]     diet[Yogurt](S)
NOTE: Optimal.
NOTE: Objective = 12.0813379.
```

Example 5.3: Two-Person Zero-Sum Game

Consider a two-person zero-sum game (where one person wins what the other person loses). The players make moves simultaneously, and each has a choice of actions. There is a *payoff* matrix that indicates the amount one player gives to the other under each combination of actions:

$$\begin{array}{rc} & \text{Player II plays } j \\ & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \text{Player I plays } i \quad \begin{array}{c} 1 \\ 2 \\ 3 \end{array} & \begin{pmatrix} -5 & 3 & 1 & 8 \\ 5 & 5 & 4 & 6 \\ -4 & 6 & 0 & 5 \end{pmatrix} \end{array}$$

If player I makes move i and player II makes move j, then player I wins (and player II loses) a_{ij}. What is the best strategy for the two players to adopt? This example is simple enough to be analyzed from observation. Suppose player I plays 1 or 3; the best response of player II is to play 1. In both cases, player I loses and player II wins. So the best action for player I is to play 2. In this case, the best response for player II is to play 3, which minimizes the loss. In this case, (2, 3) is a *pure-strategy Nash equilibrium* in this game.

For illustration, consider the following mixed strategy case. Assume that player I selects i with probability p_i, $i = 1, 2, 3$, and player II selects j with probability q_j, $j = 1, 2, 3, 4$. Consider player II's problem of minimizing the maximum expected payout:

$$\min_{\mathbf{q}} \left\{ \max_{i} \sum_{j=1}^{4} a_{ij} q_j \right\} \quad \text{subject to} \quad \sum_{j=1}^{4} q_{ij} = 1, \quad \mathbf{q} \geq 0$$

This is equivalent to

$$\begin{array}{lrcl} \min\limits_{\mathbf{q},v} \; v \quad \text{subject to} & \sum_{j=1}^{4} a_{ij}q_j & \le & v \quad \forall\, i \\ & \sum_{j=1}^{4} q_j & = & 1 \\ & \mathbf{q} & \ge & 0 \end{array}$$

The problem can be transformed into a more standard format by making a simple change of variables: $x_j = q_j/v$. The preceding LP formulation now becomes

$$\begin{array}{lrcl} \min\limits_{\mathbf{x},v} \; v \quad \text{subject to} & \sum_{j=1}^{4} a_{ij}x_j & \le & 1 \quad \forall\, i \\ & \sum_{j=1}^{4} x_j & = & 1/v \\ & \mathbf{q} & \ge & 0 \end{array}$$

which is equivalent to

$$\max_{\mathbf{x}} \sum_{j=1}^{4} x_j \quad \text{subject to} \quad A\mathbf{x} \le \mathbf{1}, \quad \mathbf{x} \ge 0$$

where A is the payoff matrix and $\mathbf{1}$ is a vector of 1's. It turns out that the corresponding optimization problem from player I's perspective can be obtained by solving the dual problem, which can be written as

$$\min_{\mathbf{y}} \sum_{i=1}^{3} y_i \quad \text{subject to} \quad A^{\mathrm{T}}\mathbf{y} \ge \mathbf{1}, \quad \mathbf{y} \ge 0$$

You can model the problem and solve it by using PROC OPTMODEL as follows:

```
proc optmodel;
   num a{1..3, 1..4}=[-5 3 1 8
                       5 5 4 6
                      -4 6 0 5];
   var x{1..4} >= 0;
   max f = sum{i in 1..4}x[i];
   con c{i in 1..3}: sum{j in 1..4}a[i,j]*x[j] <= 1;
   solve with lp / solver = ps presolver = none printfreq = 1;
   print x;
   print c.dual;
quit;
```

The optimal solution is displayed in Output 5.3.1.

Output 5.3.1 Optimal Solutions to the Two-Person Zero-Sum Game

```
                    The OPTMODEL Procedure

                       Problem Summary

            Objective Sense            Maximization
            Objective Function                    f
            Objective Type                   Linear

            Number of Variables                   4
            Bounded Above                         0
            Bounded Below                         4
            Bounded Below and Above               0
            Free                                  0
            Fixed                                 0

            Number of Constraints                 3
            Linear LE (<=)                        3
            Linear EQ (=)                         0
            Linear GE (>=)                        0
            Linear Range                          0

            Constraint Coefficients              11

                       Solution Summary

             Solver                  Primal Simplex
             Objective Function                   f
             Solution Status                Optimal
             Objective Value                   0.25
             Iterations                           1

             Primal Infeasibility                 0
             Dual Infeasibility                   0
             Bound Infeasibility                  0

                          [1]       x

                           1      0.00
                           2      0.00
                           3      0.25
                           4      0.00

                         [1]     c.DUAL

                          1        0.00
                          2        0.25
                          3        0.00
```

The optimal solution $\mathbf{x}^* = (0, 0, 0.25, 0)$ with an optimal value of 0.25. Therefore the optimal strategy for player II is $\mathbf{q}^* = \mathbf{x}^*/0.25 = (0, 0, 1, 0)$. You can check the optimal solution of the dual problem by using the constraint suffix ".dual". So $\mathbf{y}^* = (0, 0.25, 0)$ and player I's optimal strategy is (0, 1, 0). The solution is consistent with our intuition from observation.

Example 5.4: Finding an Irreducible Infeasible Set

This example demonstrates the use of the IIS= option to locate an irreducible infeasible set. Suppose you want to solve a linear program that has the following simple formulation:

$$
\begin{array}{rrcrcrcrl}
\min & x_1 & + & x_2 & + & x_3 & & & \text{(cost)} \\
\text{subject to} & x_1 & + & x_2 & & & \geq & 10 & \text{(con1)} \\
 & x_1 & & & + & x_3 & \leq & 4 & \text{(con2)} \\
4 \leq & & & x_2 & + & x_3 & \leq & 5 & \text{(con3)} \\
 & & & & x_1, & x_2 & \geq & 0 & \\
 & & & 0 & \leq & x_3 & \leq & 3 &
\end{array}
$$

It is easy to verify that the following three constraints (or rows) and one variable (or column) bound form an IIS for this problem:

$$
\begin{array}{rcrcrcrl}
x_1 & + & x_2 & & & \geq & 10 & \text{(con1)} \\
x_1 & & & + & x_3 & \leq & 4 & \text{(con2)} \\
 & & x_2 & + & x_3 & \leq & 5 & \text{(con3)} \\
 & & & & x_3 & \geq & 0 &
\end{array}
$$

You can formulate the problem and call the LP solver by using the following statements:

```
proc optmodel presolver=none;
   /* declare variables */
   var x{1..3} >=0;

   /* upper bound on variable x[3] */
   x[3].ub = 3;

   /* objective function */
   min obj = x[1] + x[2] + x[3];

   /* constraints */
   con c1: x[1] + x[2] >= 10;
   con c2: x[1] + x[3] <= 4;
   con c3: 4 <= x[2] + x[3] <= 5;

   solve with lp / iis = on;

   print x.status;
   print c1.status c2.status c3.status;
```

The notes printed in the log appear in Output 5.4.1.

Output 5.4.1 Finding an IIS: Log

```
NOTE: The problem has 3 variables (0 free, 0 fixed).
NOTE: The problem has 3 linear constraints (1 LE, 0 EQ, 1 GE, 1 range).
NOTE: The problem has 6 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The IIS option is called.
                         Objective
      Phase Iteration    Value
        1         1        5.000000
        1         2        1.000000
NOTE: Processing rows.
        1         3               0
        1         4               0
NOTE: Processing columns.
        1         5               0
NOTE: The IIS option found an IIS set with 3 rows and 1 columns.
```

The output of the PRINT statements appears in Output 5.4.2. The value of the .status suffix for the variables x[1] and x[2] is "I," which indicates an infeasible problem. The value I is not one of those assigned by the IIS= option to members of the IIS, however, so the variable bounds for x[1] and x[2] are not in the IIS.

Output 5.4.2 Solution Summary, Variable Status, and Constraint Status

```
                              The OPTMODEL Procedure

                                 Solution Summary

                        Solver                    Dual Simplex
                        Objective Function                 obj
                        Solution Status             Infeasible
                        Objective Value                      .
                        Iterations                           5

                                 [1]     x.STATUS

                                 1
                                 2
                                 3       I_L

                       c1.STATUS     c2.STATUS     c3.STATUS

                       I_L           I_U           I_U
```

The value of c3.status is I_U, which indicates that $x_2 + x_3 \leq 5$ is an element of the IIS. The original constraint is c3, a range constraint with a lower bound of 4. If you choose to remove the constraint $x_2 + x_3 \leq 5$, you can change the value of c3.ub to the largest positive number representable in your operating environment. You can specify this number by using the MIN aggregation expression in the OPTMODEL procedure. See "MIN Aggregation Expression" on page 96 for details.

The modified LP problem is specified and solved by adding the following lines to the original PROC OPTMODEL call.

```
/* relax upper bound on constraint c3 */
c3.ub = min{{}}0;

solve with lp / iis = on;

/* print solution */
print x;
```

Because one element of the IIS has been removed, the modified LP problem should no longer contain the infeasible set. Due to the size of this problem, there should be no additional irreducible infeasible sets.

The notes shown in Output 5.4.3 are printed to the log.

Output 5.4.3 Infeasibility Removed: Log

```
NOTE: The problem has 3 variables (0 free, 0 fixed).
NOTE: The problem has 3 linear constraints (1 LE, 0 EQ, 2 GE, 0 range).
NOTE: The problem has 6 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The IIS option is called.
                       Objective
      Phase Iteration  Value
        1           1             0
NOTE: The IIS option found this problem to be feasible.
NOTE: The OPTLP presolver value AUTOMATIC is applied.
NOTE: The OPTLP presolver removed 0 variables and 0 constraints.
NOTE: The OPTLP presolver removed 0 constraint coefficients.
NOTE: The presolved problem has 3 variables, 3 constraints, and 6 constraint
      coefficients.
NOTE: The DUAL SIMPLEX solver is called.
                       Objective
      Phase Iteration  Value
        2           1     10.000000
NOTE: Optimal.
NOTE: Objective = 10.
```

The solution summary and primal solution are displayed in Output 5.4.4.

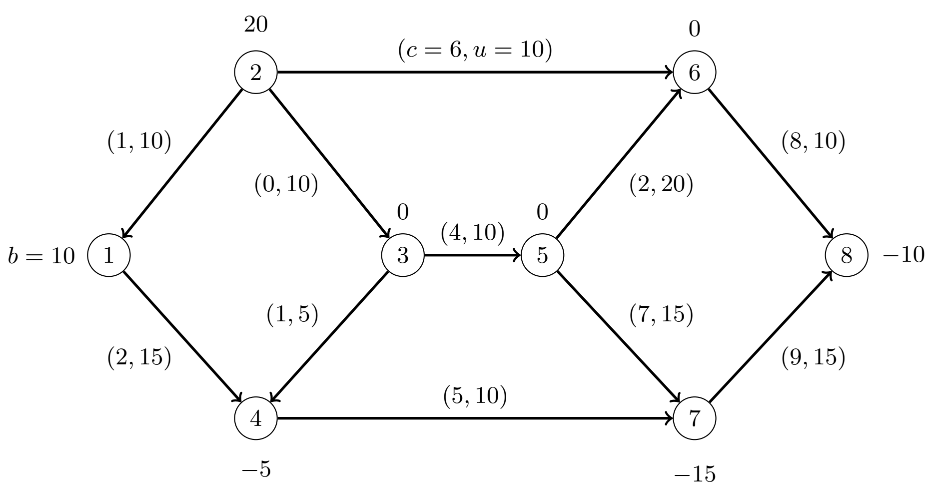
Output 5.4.4 Infeasibility Removed: Solution

```
                    The OPTMODEL Procedure

                       Solution Summary

            Solver                       Dual Simplex
            Objective Function                    obj
            Solution Status                   Optimal
            Objective Value                        10
            Iterations                              1

            Primal Infeasibility                    0
            Dual Infeasibility                      0
            Bound Infeasibility                     0

                         [1]      x

                           1      0
                           2     10
                           3      0
```

Example 5.5: Using the Network Simplex Solver

This example demonstrates how you can use the network simplex solver to find the minimum-cost flow in a directed graph. Consider the directed graph in Figure 5.4, which appears in Ahuja, Magnanti, and Orlin (1993).

Figure 5.4 Minimum Cost Network Flow Problem: Data



You can use the following SAS statements to create the input data sets nodedata and arcdata:

```
data nodedata;
   input _node_ $ _sd_;
   datalines;
```

```
1    10
2    20
3     0
4    -5
5     0
6     0
7   -15
8   -10
;
run;

data arcdata;
   input _tail_ $ _head_ $ _lo_ _capac_ _cost_;
   datalines;
1    4    0    15    2
2    1    0    10    1
2    3    0    10    0
2    6    0    10    6
3    4    0     5    1
3    5    0    10    4
4    7    0    10    5
5    6    0    20    2
5    7    0    15    7
6    8    0    10    8
7    8    0    15    9
;
run;
```

You can use the following call to PROC OPTMODEL to find the minimum-cost flow:

```
proc optmodel;
   set <str> NODES;
   num supply_demand {NODES};

   set <str,str> ARCS;
   num arcLower  {ARCS};
   num arcUpper  {ARCS};
   num arcCost   {ARCS};

   read data arcdata into ARCS=[_tail_ _head_]
      arcLower=_lo_ arcUpper=_capac_ arcCost=_cost_;
   read data nodedata into NODES=[_node_] supply_demand=_sd_;

   var flow {<i,j> in ARCS} >= arcLower[i,j] <= arcUpper[i,j];
   min obj = sum {<i,j> in ARCS} arcCost[i,j] * flow[i,j];
   con balance {i in NODES}:
      sum {<(i),j> in ARCS} flow[i,j] - sum {<j,(i)> in ARCS} flow[j,i]
         = supply_demand[i];
   solve with lp / solver=ns scale=none printfreq=1;
   print flow;
quit;
%put &_OROPTMODEL_;
```

The optimal solution is displayed in Output 5.5.1.

Output 5.5.1 Network Simplex Solver: Primal Solution Output

The OPTMODEL Procedure

Problem Summary

Objective Sense	Minimization
Objective Function	obj
Objective Type	Linear
Number of Variables	11
Bounded Above	0
Bounded Below	0
Bounded Below and Above	11
Free	0
Fixed	0
Number of Constraints	8
Linear LE (<=)	0
Linear EQ (=)	8
Linear GE (>=)	0
Linear Range	0
Constraint Coefficients	22

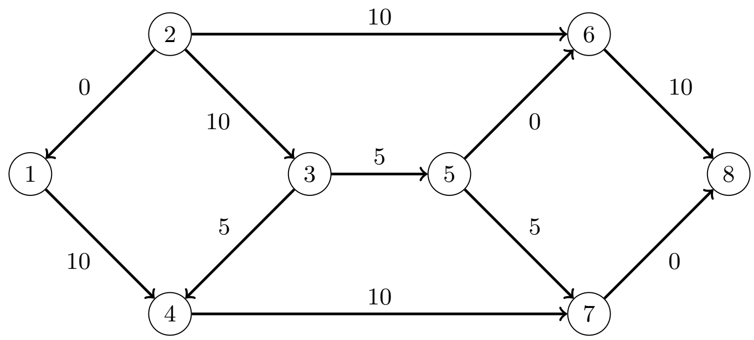
Solution Summary

Solver	Network Simplex
Objective Function	obj
Solution Status	Optimal
Objective Value	270
Iterations	8
Iterations2	1
Primal Infeasibility	0
Dual Infeasibility	0
Bound Infeasibility	0

[1]	[2]	flow
1	4	10
2	1	0
2	3	10
2	6	10
3	4	5
3	5	5
4	7	10
5	6	0
5	7	5
6	8	10
7	8	0

The optimal solution is represented graphically in Figure 5.5.

Figure 5.5 Minimum Cost Network Flow Problem: Optimal Solution



The iteration log is displayed in Output 5.5.2.

Output 5.5.2 Log: Solution Progress

```
NOTE: There were 11 observations read from the data set WORK.ARCDATA.
NOTE: There were 8 observations read from the data set WORK.NODEDATA.
NOTE: The problem has 11 variables (0 free, 0 fixed).
NOTE: The problem has 8 linear constraints (0 LE, 8 EQ, 0 GE, 0 range).
NOTE: The problem has 22 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The PRESOLVER value NONE is applied because a pure network has been found.
NOTE: The OPTLP presolver value NONE is applied.
NOTE: The NETWORK SIMPLEX solver is called.
NOTE: The network has 8 rows (100.00%), 11 columns (100.00%), and 1 component.
NOTE: The network extraction and setup time is 0.00 seconds.
      Iteration      PrimalObj      PrimalInf     Time
              1              0     20.0000000     0.00
              2              0     20.0000000     0.00
              3      5.0000000     15.0000000     0.00
              4      5.0000000     15.0000000     0.00
              5     75.0000000     15.0000000     0.00
              6     75.0000000     15.0000000     0.00
              7    130.0000000     10.0000000     0.00
              8    270.0000000              0     0.00
NOTE: The Network Simplex solve time is 0.00 seconds.
NOTE: The total Network Simplex solve time is 0.00 seconds.
NOTE: Optimal.
NOTE: Objective = 270.
NOTE: The PRIMAL SIMPLEX solver is called.
                        Objective     Entering          Leaving
      Phase Iteration   Value         Variable          Variable
        2           1     270.000000  flow['2','1']     flow['4','7'](S)
NOTE: Optimal.
NOTE: Objective = 270.
STATUS=OK SOLUTION_STATUS=OPTIMAL OBJECTIVE=270 PRIMAL_INFEASIBILITY=0
DUAL_INFEASIBILITY=0 BOUND_INFEASIBILITY=0 ITERATIONS=8 ITERATIONS2=1
PRESOLVE_TIME=0.00 SOLUTION_TIME=0.00
```

Now, suppose there is a budget on the flow that comes out of arc 2: the total arc cost of flow that comes out of arc 2 cannot exceed 50. You can use the following call to PROC OPTMODEL to find the minimum-cost flow:

```
proc optmodel;
   set <str> NODES;
   num supply_demand {NODES};

   set <str,str> ARCS;
   num arcLower  {ARCS};
   num arcUpper  {ARCS};
   num arcCost   {ARCS};

   read data arcdata into ARCS=[_tail_ _head_]
      arcLower=_lo_ arcUpper=_capac_ arcCost=_cost_;
   read data nodedata into NODES=[_node_] supply_demand=_sd_;

   var flow {<i,j> in ARCS} >= arcLower[i,j] <= arcUpper[i,j];
   min obj = sum {<i,j> in ARCS} arcCost[i,j] * flow[i,j];
   con balance {i in NODES}:
      sum {<(i),j> in ARCS} flow[i,j] - sum {<j,(i)> in ARCS} flow[j,i]
         = supply_demand[i];
   con budgetOn2:
      sum {<i,j> in ARCS: i='2'} arcCost[i,j] * flow[i,j] <= 50;
   solve with lp / solver=ns scale=none printfreq=1;
   print flow;
quit;
%put &_OROPTMODEL_;
```

The optimal solution is displayed in Output 5.5.3.

Output 5.5.3 Network Simplex Solver: Primal Solution Output

The OPTMODEL Procedure

Problem Summary

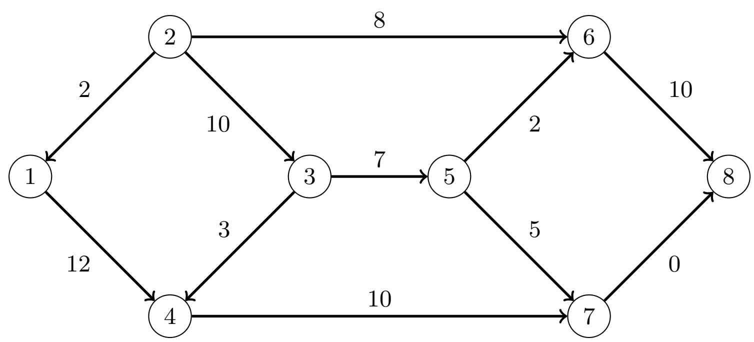
Objective Sense	Minimization
Objective Function	obj
Objective Type	Linear
Number of Variables	11
Bounded Above	0
Bounded Below	0
Bounded Below and Above	11
Free	0
Fixed	0
Number of Constraints	9
Linear LE (<=)	1
Linear EQ (=)	8
Linear GE (>=)	0
Linear Range	0
Constraint Coefficients	24

Output 5.5.3 *continued*

```
                    Solution Summary

         Solver                    Network Simplex
         Objective Function                    obj
         Solution Status                   Optimal
         Objective Value                       274
         Iterations                              5
         Iterations2                             2

         Primal Infeasibility         8.881784E-16
         Dual Infeasibility                      0
         Bound Infeasibility                     0

                    [1]     [2]     flow

                    1       4         12
                    2       1          2
                    2       3         10
                    2       6          8
                    3       4          3
                    3       5          7
                    4       7         10
                    5       6          2
                    5       7          5
                    6       8         10
                    7       8          0
```

The optimal solution is represented graphically in Figure 5.6.

Figure 5.6 Minimum Cost Network Flow Problem: Optimal Solution (with Budget Constraint)



The iteration log is displayed in Output 5.5.4. Note that the network simplex solver extracts a subnetwork in this case.

Output 5.5.4 Log: Solution Progress

```
NOTE: There were 11 observations read from the data set WORK.ARCDATA.
NOTE: There were 8 observations read from the data set WORK.NODEDATA.
NOTE: The problem has 11 variables (0 free, 0 fixed).
NOTE: The problem has 9 linear constraints (1 LE, 8 EQ, 0 GE, 0 range).
NOTE: The problem has 24 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The OPTLP presolver value AUTOMATIC is applied.
NOTE: The OPTLP presolver removed 6 variables and 7 constraints.
NOTE: The OPTLP presolver removed 15 constraint coefficients.
NOTE: The presolved problem has 5 variables, 2 constraints, and 9 constraint
      coefficients.
NOTE: The NETWORK SIMPLEX solver is called.
NOTE: The network has 1 rows (50.00%), 5 columns (100.00%), and 1 component.
NOTE: The network extraction and setup time is 0.00 seconds.
      Iteration      PrimalObj      PrimalInf     Time
              1    259.9800000      5.0200000     0.00
              2    264.9900000      0.0100000     0.00
              3    265.0300000              0     0.00
              4    255.0300000              0     0.00
              5    270.0000000              0     0.00
NOTE: The Network Simplex solve time is 0.00 seconds.
NOTE: The total Network Simplex solve time is 0.00 seconds.
NOTE: Optimal.
NOTE: Objective = 270.
NOTE: The DUAL SIMPLEX solver is called.
                        Objective     Entering          Leaving
      Phase Iteration   Value         Variable          Variable
        2           1     270.000000  flow['5','6']     budgetOn2    (S)
        2           2     274.000000  flow['3','4']     flow['2','3'](S)
NOTE: Optimal.
NOTE: Objective = 274.
STATUS=OK SOLUTION_STATUS=OPTIMAL OBJECTIVE=274
PRIMAL_INFEASIBILITY=8.881784E-16 DUAL_INFEASIBILITY=0 BOUND_INFEASIBILITY=0
ITERATIONS=5 ITERATIONS2=2 PRESOLVE_TIME=0.00 SOLUTION_TIME=0.00
```

Example 5.6: Migration to OPTMODEL: Generalized Networks

The following example shows how to use PROC OPTMODEL to solve the example “Generalized Networks: Using the EXCESS= Option” in Chapter 6, “The NETFLOW Procedure” (*SAS/OR User’s Guide: Mathematical Programming Legacy Procedures*). The input data sets are the same as in the PROC NETFLOW example.

```
title 'Generalized Networks';

data garcs;
   input _from_ $ _to_ $ _cost_ _mult_;
   datalines;
s1 d1 1 .
s1 d2 8 .
s2 d1 4 2
```

```
s2 d2 2 2
s2 d3 1 2
s3 d2 5 0.5
s3 d3 4 0.5
;

data gnodes;
   input _node_ $ _sd_ ;
   datalines;
s1 5
s2 20
s3 10
d1 -5
d2 -10
d3 -20
;
```

The following PROC OPTMODEL statements read the data sets, build the linear programming model, solve the model, and output the optimal solution to a SAS data set called GENETOUT:

```
proc optmodel;
   set <str> NODES;
   num _sd_ {NODES} init 0;
   read data gnodes into NODES=[_node_] _sd_;

   set <str,str> ARCS;
   num _lo_ {ARCS} init 0;
   num _capac_ {ARCS} init .;
   num _cost_ {ARCS};
   num _mult_ {ARCS} init 1;
   read data garcs nomiss into ARCS=[_from_ _to_] _cost_ _mult_;
   NODES = NODES union (union {<i,j> in ARCS} {i,j});

   var Flow {<i,j> in ARCS} >= _lo_[i,j];
   min obj = sum {<i,j> in ARCS} _cost_[i,j] * Flow[i,j];
   con balance {i in NODES}: sum {<(i),j> in ARCS} Flow[i,j]
      - sum {<j,(i)> in ARCS} _mult_[j,i] * Flow[j,i] = _sd_[i];

   num infinity = min {r in {}} r;
   /* change equality constraint to le constraint for supply nodes */
   for {i in NODES: _sd_[i] > 0} balance[i].lb = -infinity;

   solve;

   num _supply_ {<i,j> in ARCS} = (if _sd_[i] ne 0 then _sd_[i] else .);
   num _demand_ {<i,j> in ARCS} = (if _sd_[j] ne 0 then -_sd_[j] else .);
   num _fcost_ {<i,j> in ARCS} = _cost_[i,j] * Flow[i,j].sol;

   create data gnetout from [_from_ _to_]
      _cost_ _capac_ _lo_ _mult_ _supply_ _demand_ _flow_=Flow _fcost_;
quit;
```

To solve a generalized network flow problem, the usual balance constraint is altered to include the arc multiplier "_mult_[i,j]" in the second sum. The balance constraint is initially declared as an equality, but to mimic the EXCESS=SUPPLY option in PROC NETFLOW, the sense of this constraint is changed to "≤" by relaxing the constraint's lower bound for supply nodes. The output data set is displayed in Output 5.6.1.

Output 5.6.1 Optimal Solution with Excess Supply

```
                         Generalized Networks

Obs _from_ _to_ _cost_ _capac_ _lo_ _mult_ _supply_ _demand_ _flow_ _fcost_

 1    s1    d1     1       .      0    1.0      5        5       5       5
 2    s1    d2     8       .      0    1.0      5       10       0       0
 3    s2    d1     4       .      0    2.0     20        5       0       0
 4    s2    d2     2       .      0    2.0     20       10       5      10
 5    s2    d3     1       .      0    2.0     20       20      10      10
 6    s3    d2     5       .      0    0.5     10       10       0       0
 7    s3    d3     4       .      0    0.5     10       20       0       0
```

The log is displayed in Output 5.6.2.

Output 5.6.2 OPTMODEL Log

```
NOTE: There were 6 observations read from the data set WORK.GNODES.
NOTE: There were 7 observations read from the data set WORK.GARCS.
NOTE: The problem has 7 variables (0 free, 0 fixed).
NOTE: The problem has 6 linear constraints (3 LE, 3 EQ, 0 GE, 0 range).
NOTE: The problem has 14 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The OPTMODEL presolver is disabled for linear problems.
NOTE: The OPTLP presolver value AUTOMATIC is applied.
NOTE: The OPTLP presolver removed 2 variables and 2 constraints.
NOTE: The OPTLP presolver removed 4 constraint coefficients.
NOTE: The presolved problem has 5 variables, 4 constraints, and 10 constraint
      coefficients.
NOTE: The DUAL SIMPLEX solver is called.
                        Objective
      Phase Iteration   Value
        2           1     25.000000
NOTE: Optimal.
NOTE: Objective = 25.
NOTE: The data set WORK.GNETOUT has 7 observations and 10 variables.
```

Now consider the previous example but with a slight modification to the arc multipliers, as in the PROC NETFLOW example:

```
data garcs1;
   input _from_ $ _to_ $ _cost_ _mult_;
   datalines;
s1 d1 1 0.5
s1 d2 8 0.5
s2 d1 4 .
s2 d2 2 .
s2 d3 1 .
s3 d2 5 0.5
s3 d3 4 0.5
;
```

The following PROC OPTMODEL statements are identical to the preceding example, except for the balance constraint. The balance constraint is still initially declared as an equality, but to mimic the PROC NETFLOW EXCESS=DEMAND option, the sense of this constraint is changed to "≥" by relaxing the constraint's upper bound for demand nodes.

```
proc optmodel;
   set <str> NODES;
   num _sd_ {NODES} init 0;
   read data gnodes into NODES=[_node_] _sd_;

   set <str,str> ARCS;
   num _lo_ {ARCS} init 0;
   num _capac_ {ARCS} init .;
   num _cost_ {ARCS};
   num _mult_ {ARCS} init 1;
   read data garcs1 nomiss into ARCS=[_from_ _to_] _cost_ _mult_;
   NODES = NODES union (union {<i,j> in ARCS} {i,j});

   var Flow {<i,j> in ARCS} >= _lo_[i,j];
   for {<i,j> in ARCS: _capac_[i,j] ne .} Flow[i,j].ub = _capac_[i,j];
   min obj = sum {<i,j> in ARCS} _cost_[i,j] * Flow[i,j];
   con balance {i in NODES}: sum {<(i),j> in ARCS} Flow[i,j]
      - sum {<j,(i)> in ARCS} _mult_[j,i] * Flow[j,i] = _sd_[i];

   num infinity = min {r in {}} r;
   /* change equality constraint to ge constraint */
   for {i in NODES: _sd_[i] < 0} balance[i].ub = infinity;

   solve;

   num _supply_ {<i,j> in ARCS} = (if _sd_[i] ne 0 then _sd_[i] else .);
   num _demand_ {<i,j> in ARCS} = (if _sd_[j] ne 0 then -_sd_[j] else .);
   num _fcost_ {<i,j> in ARCS} = _cost_[i,j] * Flow[i,j].sol;

   create data gnetout1 from [_from_ _to_]
      _cost_ _capac_ _lo_ _mult_ _supply_ _demand_ _flow_=Flow _fcost_;
quit;
```

The output data set is displayed in Output 5.6.3.

Output 5.6.3 Optimal Solution with Excess Demand

```
                              Generalized Networks

  Obs _from_ _to_ _cost_ _capac_ _lo_ _mult_ _supply_ _demand_ _flow_ _fcost_

   1    s1    d1     1       .      0    0.5       5        5       5       5
   2    s1    d2     8       .      0    0.5       5       10       0       0
   3    s2    d1     4       .      0    1.0      20        5       0       0
   4    s2    d2     2       .      0    1.0      20       10       5      10
   5    s2    d3     1       .      0    1.0      20       20      15      15
   6    s3    d2     5       .      0    0.5      10       10       0       0
   7    s3    d3     4       .      0    0.5      10       20      10      40
```

The log is displayed in Output 5.6.4.

Output 5.6.4 OPTMODEL Log

```
NOTE: There were 6 observations read from the data set WORK.GNODES.
NOTE: There were 7 observations read from the data set WORK.GARCS1.
NOTE: The problem has 7 variables (0 free, 0 fixed).
NOTE: The problem has 6 linear constraints (0 LE, 3 EQ, 3 GE, 0 range).
NOTE: The problem has 14 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The OPTMODEL presolver is disabled for linear problems.
NOTE: The OPTLP presolver value AUTOMATIC is applied.
NOTE: The OPTLP presolver removed 2 variables and 2 constraints.
NOTE: The OPTLP presolver removed 4 constraint coefficients.
NOTE: The presolved problem has 5 variables, 4 constraints, and 10 constraint
      coefficients.
NOTE: The DUAL SIMPLEX solver is called.
                        Objective
      Phase Iteration   Value
        2           1      65.000000
        2           2      70.000000
NOTE: Optimal.
NOTE: Objective = 70.
NOTE: The data set WORK.GNETOUT1 has 7 observations and 10 variables.
```

Example 5.7: Migration to OPTMODEL: Maximum Flow

The following example shows how to use PROC OPTMODEL to solve the example “Maximum Flow Problem” in Chapter 6, “The NETFLOW Procedure” (*SAS/OR User’s Guide: Mathematical Programming Legacy Procedures*). The input data set is the same as in that example.

```
title 'Maximum Flow Problem';

data arcs;
   input _from_ $ _to_ $ _cost_ _capac_;
   datalines;
S a . .
S b . .
a c 1 7
b c 2 9
a d 3 5
b d 4 8
c e 5 15
d f 6 20
e g 7 11
f g 8 6
e h 9 12
f h 10 4
g T . .
h T . .
;
```

The following PROC OPTMODEL statements read the data sets, build the linear programming model, solve the model, and output the optimal solution to a SAS data set called GOUT3:

```
proc optmodel;
   str source = 'S';
   str sink = 'T';

   set <str> NODES;
   num _supdem_ {i in NODES} = (if i in {source, sink} then . else 0);

   set <str,str> ARCS;
   num _lo_ {ARCS} init 0;
   num _capac_ {ARCS} init .;
   num _cost_ {ARCS} init 0;
   read data arcs nomiss into ARCS=[_from_ _to_] _cost_ _capac_;
   NODES = (union {<i,j> in ARCS} {i,j});

   var Flow {<i,j> in ARCS} >= _lo_[i,j];
   for {<i,j> in ARCS: _capac_[i,j] ne .} Flow[i,j].ub = _capac_[i,j];
   max obj = sum {<i,j> in ARCS: j = sink} Flow[i,j];
   con balance {i in NODES diff {source, sink}}:
      sum {<(i),j> in ARCS} Flow[i,j]
      - sum {<j,(i)> in ARCS} Flow[j,i] = _supdem_[i];

   solve;

   num _supply_ {<i,j> in ARCS} =
      (if _supdem_[i] ne 0 then _supdem_[i] else .);
   num _demand_ {<i,j> in ARCS} =
      (if _supdem_[j] ne 0 then -_supdem_[j] else .);
   num _fcost_ {<i,j> in ARCS} = _cost_[i,j] * Flow[i,j].sol;
```

```
      create data gout3 from [_from_ _to_]
         _cost_ _capac_ _lo_ _supply_ _demand_ _flow_=Flow _fcost_;
   quit;
```

To solve a maximum flow problem, you solve a network flow problem that has a zero supply or demand at all nodes other than the source and sink nodes, as specified in the declaration of the _SUPDEM_ numeric parameter and the balance constraint. The objective declaration uses the logical condition J = SINK to maximize the flow into the sink node. The output data set is displayed in Output 5.7.1.

Output 5.7.1 Optimal Solution

Maximum Flow Problem

Obs	_from_	_to_	_cost_	_capac_	_lo_	_supply_	_demand_	_flow_	_fcost_
1	S	a	0	.	0	.	.	12	0
2	S	b	0	.	0	.	.	13	0
3	a	c	1	7	0	.	.	7	7
4	b	c	2	9	0	.	.	8	16
5	a	d	3	5	0	.	.	5	15
6	b	d	4	8	0	.	.	5	20
7	c	e	5	15	0	.	.	15	75
8	d	f	6	20	0	.	.	10	60
9	e	g	7	11	0	.	.	3	21
10	f	g	8	6	0	.	.	6	48
11	e	h	9	12	0	.	.	12	108
12	f	h	10	4	0	.	.	4	40
13	g	T	0	.	0	.	.	9	0
14	h	T	0	.	0	.	.	16	0

The log is displayed in Output 5.7.2.

Output 5.7.2 OPTMODEL Log

```
NOTE: There were 14 observations read from the data set WORK.ARCS.
NOTE: The problem has 14 variables (0 free, 0 fixed).
NOTE: The problem has 8 linear constraints (0 LE, 8 EQ, 0 GE, 0 range).
NOTE: The problem has 24 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The OPTMODEL presolver is disabled for linear problems.
NOTE: The OPTLP presolver value AUTOMATIC is applied.
NOTE: The OPTLP presolver removed 10 variables and 6 constraints.
NOTE: The OPTLP presolver removed 20 constraint coefficients.
NOTE: The presolved problem has 4 variables, 2 constraints, and 4 constraint
      coefficients.
NOTE: The DUAL SIMPLEX solver is called.
                         Objective
      Phase Iteration    Value
        2           1      25.000000
        2           2      25.000000
NOTE: Optimal.
NOTE: Objective = 25.
NOTE: The data set WORK.GOUT3 has 14 observations and 9 variables.
```

Example 5.8: Migration to OPTMODEL: Production, Inventory, Distribution

The following example shows how to use PROC OPTMODEL to solve the example "Production, Inventory, Distribution Problem" in Chapter 6, "The NETFLOW Procedure" (*SAS/OR User's Guide: Mathematical Programming Legacy Procedures*). The input data sets are the same as in that example.

```
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   .
;
```

The following PROC OPTMODEL statements read the data sets, build the linear programming model, solve the model, and output the optimal solution to SAS data sets called ARC1 and NODE2:

```
proc optmodel;
   set <str> NODES;
   num _supdem_ {NODES} init 0;
   read data node0 into NODES=[_node_] _supdem_;

   set <str,str> ARCS;
   num _lo_ {ARCS} init 0;
   num _capac_ {ARCS} init .;
   num _cost_ {ARCS};
   num diagonal {ARCS};
   num factory {ARCS};
```

```
   str key_id {ARCS};
   str mth_made {ARCS};
   str _name_ {ARCS};
   read data arc0 nomiss into ARCS=[_tail_ _head_] _lo_ _capac_ _cost_
      diagonal factory key_id mth_made _name_;
   NODES = NODES union (union {<i,j> in ARCS} {i,j});

   var Flow {<i,j> in ARCS} >= _lo_[i,j];
   for {<i,j> in ARCS: _capac_[i,j] ne .} Flow[i,j].ub = _capac_[i,j];
   min obj = sum {<i,j> in ARCS} _cost_[i,j] * Flow[i,j];
   con balance {i in NODES}: sum {<(i),j> in ARCS} Flow[i,j]
      - sum {<j,(i)> in ARCS} Flow[j,i] = _supdem_[i];

   num infinity = min {r in {}} r;
   num excess = sum {i in NODES} _supdem_[i];
   if (excess > 0) then do;
      /* change equality constraint to le constraint for supply nodes */
      for {i in NODES: _supdem_[i] > 0} balance[i].lb = -infinity;
   end;
   else if (excess < 0) then do;
      /* change equality constraint to ge constraint for demand nodes */
      for {i in NODES: _supdem_[i] < 0} balance[i].ub = infinity;
   end;

   solve;

   num _supply_ {<i,j> in ARCS} =
      (if _supdem_[i] ne 0 then _supdem_[i] else .);
   num _demand_ {<i,j> in ARCS} =
      (if _supdem_[j] ne 0 then -_supdem_[j] else .);
   num _fcost_ {<i,j> in ARCS} = _cost_[i,j] * Flow[i,j].sol;

   create data arc1 from [_tail_ _head_]
      _cost_ _capac_ _lo_ _name_ _supply_ _demand_ _flow_=Flow _fcost_
      _rcost_ =
         (if Flow[_tail_,_head_].rc ne 0 then Flow[_tail_,_head_].rc else .)
      _status_ = Flow.status diagonal factory key_id mth_made;
   create data node2 from [_node_]
      _supdem_ = (if _supdem_[_node_] ne 0 then _supdem_[_node_] else .)
      _dual_ = balance.dual;
quit;
```

The PROC OPTMODEL statements use both single-dimensional (NODES) and multiple-dimensional (ARCS) index sets, which are populated from the corresponding data set variables in the READ DATA statements. The _SUPDEM_, _LO_, and _CAPAC_ parameters are given initial values, and the NOMISS option in the READ DATA statement tells PROC OPTMODEL to read only the nonmissing values from the input data set. The balance constraint is initially declared as an equality, but depending on the total supply or demand, the sense of this constraint is changed to "$\leq$" or "$\geq$" by relaxing the constraint's lower or upper bound, respectively. The ARC1 output data set contains most of the same information as in the NETFLOW example, including reduced cost, basis status, and dual values. The _ANUMB_ and _TNUMB_ values do not apply here.

The PROC PRINT statements are similar to the PROC NETFLOW example:

```
options ls=80 ps=54;
proc print data=arc1 heading=h width=min;
   var _tail_ _head_ _cost_ _capac_ _lo_ _name_
   _supply_ _demand_ _flow_ _fcost_;
   sum _fcost_;
run;
proc print data=arc1 heading=h width=min;
   var _rcost_ _status_ diagonal factory key_id mth_made;
run;
proc print data=node2;
run;
```

The output data sets are displayed in Output 5.8.1.

Output 5.8.1 Output Data Sets

Minimum Cost Flow Problem
Production Planning/Inventory/Distribution

Obs	_tail_	_head_	_cost_	_capac_	_lo_	_name_
1	fact1_1	f1_mar_1	127.90	500	50	prod f1 19 mar
2	fact1_1	f1_apr_1	78.60	600	50	prod f1 19 apl
3	fact1_1	f1_may_1	95.10	400	50	
4	f1_mar_1	f1_apr_1	15.00	50	0	
5	f1_apr_1	f1_may_1	12.00	50	0	
6	f1_apr_1	f1_mar_1	28.00	20	0	back f1 19 apl
7	f1_may_1	f1_apr_1	28.00	20	0	back f1 19 may
8	f1_mar_1	f2_mar_1	11.00	.	0	
9	f1_apr_1	f2_apr_1	11.00	.	0	
10	f1_may_1	f2_may_1	16.00	.	0	
11	f1_mar_1	shop1_1	-327.65	250	0	
12	f1_apr_1	shop1_1	-300.00	250	0	
13	f1_may_1	shop1_1	-285.00	250	0	
14	f1_mar_1	shop2_1	-362.74	250	0	
15	f1_apr_1	shop2_1	-300.00	250	0	
16	f1_may_1	shop2_1	-245.00	250	0	
17	fact2_1	f2_mar_1	88.00	450	35	prod f2 19 mar
18	fact2_1	f2_apr_1	62.40	480	35	prod f2 19 apl
19	fact2_1	f2_may_1	133.80	250	35	
20	f2_mar_1	f2_apr_1	18.00	30	0	
21	f2_apr_1	f2_may_1	20.00	30	0	
22	f2_apr_1	f2_mar_1	17.00	15	0	back f2 19 apl
23	f2_may_1	f2_apr_1	25.00	15	0	back f2 19 may

Obs	_supply_	_demand_	_flow_	_fcost_
1	1000	.	345	44125.50
2	1000	.	600	47160.00
3	1000	.	50	4755.00
4	.	.	0	0.00
5	.	.	50	600.00
6	.	.	20	560.00
7	.	.	0	0.00
8	.	.	0	0.00
9	.	.	30	330.00
10	.	.	100	1600.00
11	.	900	155	-50785.75
12	.	900	250	-75000.00
13	.	900	0	0.00
14	.	900	250	-90685.00
15	.	900	250	-75000.00
16	.	900	0	0.00
17	850	.	290	25520.00
18	850	.	480	29952.00
19	850	.	35	4683.00
20	.	.	0	0.00
21	.	.	15	300.00
22	.	.	0	0.00
23	.	.	0	0.00

Output 5.8.1 *continued*

Minimum Cost Flow Problem
Production Planning/Inventory/Distribution

Obs	_tail_	_head_	_cost_	_capac_	_lo_	_name_
24	f2_mar_1	f1_mar_1	10.00	40	0	
25	f2_apr_1	f1_apr_1	11.00	40	0	
26	f2_may_1	f1_may_1	13.00	40	0	
27	f2_mar_1	shop1_1	-297.40	250	0	
28	f2_apr_1	shop1_1	-290.00	250	0	
29	f2_may_1	shop1_1	-292.00	250	0	
30	f2_mar_1	shop2_1	-272.70	250	0	
31	f2_apr_1	shop2_1	-312.00	250	0	
32	f2_may_1	shop2_1	-299.00	250	0	
33	fact1_2	f1_mar_2	217.90	400	40	prod f1 25 mar
34	fact1_2	f1_apr_2	174.50	550	50	prod f1 25 apl
35	fact1_2	f1_may_2	133.30	350	40	
36	f1_mar_2	f1_apr_2	20.00	40	0	
37	f1_apr_2	f1_may_2	18.00	40	0	
38	f1_apr_2	f1_mar_2	32.00	30	0	back f1 25 apl
39	f1_may_2	f1_apr_2	41.00	15	0	back f1 25 may
40	f1_mar_2	f2_mar_2	23.00	.	0	
41	f1_apr_2	f2_apr_2	23.00	.	0	
42	f1_may_2	f2_may_2	26.00	.	0	
43	f1_mar_2	shop1_2	-559.76	.	0	
44	f1_apr_2	shop1_2	-524.28	.	0	
45	f1_may_2	shop1_2	-475.02	.	0	
46	f1_mar_2	shop2_2	-623.89	.	0	

Obs	_supply_	_demand_	_flow_	_fcost_
24	.	.	40	400.00
25	.	.	0	0.00
26	.	.	0	0.00
27	.	900	250	-74350.00
28	.	900	245	-71050.00
29	.	900	0	0.00
30	.	900	0	0.00
31	.	900	250	-78000.00
32	.	900	150	-44850.00
33	1000	.	400	87160.00
34	1000	.	550	95975.00
35	1000	.	40	5332.00
36	.	.	0	0.00
37	.	.	0	0.00
38	.	.	30	960.00
39	.	.	15	615.00
40	.	.	0	0.00
41	.	.	0	0.00
42	.	.	0	0.00
43	.	900	0	0.00
44	.	900	0	0.00
45	.	900	25	-11875.50
46	.	1450	455	-283869.95

Output 5.8.1 *continued*

```
                    Minimum Cost Flow Problem
            Production Planning/Inventory/Distribution

 Obs     _tail_      _head_       _cost_    _capac_    _lo_       _name_

  47    f1_apr_2    shop2_2      -549.68        .         0
  48    f1_may_2    shop2_2      -460.00        .         0
  49    fact2_2     f2_mar_2      182.00      650        35     prod f2 25 mar
  50    fact2_2     f2_apr_2      196.70      680        35     prod f2 25 apl
  51    fact2_2     f2_may_2      201.40      550        35
  52    f2_mar_2    f2_apr_2       28.00       50         0
  53    f2_apr_2    f2_may_2       38.00       50         0
  54    f2_apr_2    f2_mar_2       31.00       15         0     back f2 25 apl
  55    f2_may_2    f2_apr_2       54.00       15         0     back f2 25 may
  56    f2_mar_2    f1_mar_2       20.00       25         0
  57    f2_apr_2    f1_apr_2       21.00       25         0
  58    f2_may_2    f1_may_2       43.00       25         0
  59    f2_mar_2    shop1_2      -567.83      500         0
  60    f2_apr_2    shop1_2      -542.19      500         0
  61    f2_may_2    shop1_2      -461.56      500         0
  62    f2_mar_2    shop2_2      -542.83      500         0
  63    f2_apr_2    shop2_2      -559.19      500         0
  64    f2_may_2    shop2_2      -489.06      500         0


 Obs    _supply_    _demand_     _flow_        _fcost_

  47        .         1450         535      -294078.80
  48        .         1450           0            0.00
  49     1500            .         645       117390.00
  50     1500            .         680       133756.00
  51     1500            .          35         7049.00
  52        .            .           0            0.00
  53        .            .           0            0.00
  54        .            .           0            0.00
  55        .            .          15          810.00
  56        .            .          25          500.00
  57        .            .           0            0.00
  58        .            .           0            0.00
  59        .          900         500      -283915.00
  60        .          900         375      -203321.25
  61        .          900           0            0.00
  62        .         1450         120       -65139.60
  63        .         1450         320      -178940.80
  64        .         1450          20        -9781.20
                                           ===========
                                           -1281110.35
```

Output 5.8.1 *continued*

Minimum Cost Flow Problem
Production Planning/Inventory/Distribution

Obs	_rcost_	_status_	diagonal	factory	key_id	mth_made
1	.	B	19	1	production	March
2	-0.65	U	19	1	production	April
3	0.85	L	19	1	production	May
4	63.65	L	19	1	storage	March
5	-3.00	U	19	1	storage	April
6	-20.65	U	19	1	backorder	April
7	43.00	L	19	1	backorder	May
8	50.90	L	19	.	f1_to_2	March
9	.	B	19	.	f1_to_2	April
10	.	B	19	.	f1_to_2	May
11	.	B	19	1	sales	March
12	-21.00	U	19	1	sales	April
13	9.00	L	19	1	sales	May
14	-46.09	U	19	1	sales	March
15	-32.00	U	19	1	sales	April
16	38.00	L	19	1	sales	May
17	.	B	19	2	production	March
18	-27.85	U	19	2	production	April
19	23.55	L	19	2	production	May
20	15.75	L	19	2	storage	March
21	.	B	19	2	storage	April
22	19.25	L	19	2	backorder	April
23	45.00	L	19	2	backorder	May
24	-29.90	U	19	.	f2_to_1	March
25	22.00	L	19	.	f2_to_1	April
26	29.00	L	19	.	f2_to_1	May
27	-9.65	U	19	2	sales	March
28	.	B	19	2	sales	April
29	18.00	L	19	2	sales	May
30	4.05	L	19	2	sales	March
31	-33.00	U	19	2	sales	April
32	.	B	19	2	sales	May
33	-45.16	U	25	1	production	March
34	-14.35	U	25	1	production	April
35	2.11	L	25	1	production	May
36	94.21	L	25	1	storage	March
37	75.66	L	25	1	storage	April
38	-42.21	U	25	1	backorder	April
39	-16.66	U	25	1	backorder	May
40	104.06	L	25	.	f1_to_2	March
41	13.49	L	25	.	f1_to_2	April
42	28.96	L	25	.	f1_to_2	May
43	47.13	L	25	1	sales	March
44	8.40	L	25	1	sales	April
45	.	B	25	1	sales	May
46	.	B	25	1	sales	March
47	.	B	25	1	sales	April
48	32.02	L	25	1	sales	May
49	.	B	25	2	production	March

Output 5.8.1 *continued*

Minimum Cost Flow Problem
Production Planning/Inventory/Distribution

Obs	_rcost_	_status_	diagonal	factory	key_id	mth_made
50	-1.66	U	25	2	production	April
51	73.17	L	25	2	production	May
52	11.64	L	25	2	storage	March
53	108.13	L	25	2	storage	April
54	47.36	L	25	2	backorder	April
55	-16.13	U	25	2	backorder	May
56	-61.06	U	25	.	f2_to_1	March
57	30.51	L	25	.	f2_to_1	April
58	40.04	L	25	.	f2_to_1	May
59	-42.00	U	25	2	sales	March
60	.	B	25	2	sales	April
61	10.50	L	25	2	sales	May
62	.	B	25	2	sales	March
63	.	B	25	2	sales	April
64	.	B	25	2	sales	May

Minimum Cost Flow Problem
Production Planning/Inventory/Distribution

Obs	_node_	_supdem_	_dual_
1	fact1_1	1000	0.00
2	fact2_1	850	0.00
3	fact1_2	1000	0.00
4	fact2_2	1500	0.00
5	shop1_1	-900	199.75
6	shop2_1	-900	188.75
7	shop1_2	-900	343.83
8	shop2_2	-1450	360.83
9	f1_mar_1	.	-127.90
10	f1_apr_1	.	-79.25
11	f1_may_1	.	-94.25
12	f2_mar_1	.	-88.00
13	f2_apr_1	.	-90.25
14	f2_may_1	.	-110.25
15	f1_mar_2	.	-263.06
16	f1_apr_2	.	-188.85
17	f1_may_2	.	-131.19
18	f2_mar_2	.	-182.00
19	f2_apr_2	.	-198.36
20	f2_may_2	.	-128.23

The log is displayed in Output 5.8.2.

Output 5.8.2 OPTMODEL Log

```
NOTE: There were 8 observations read from the data set WORK.NODE0.
NOTE: There were 64 observations read from the data set WORK.ARC0.
NOTE: The problem has 64 variables (0 free, 0 fixed).
NOTE: The problem has 20 linear constraints (4 LE, 16 EQ, 0 GE, 0 range).
NOTE: The problem has 128 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The OPTMODEL presolver is disabled for linear problems.
NOTE: The OPTLP presolver value AUTOMATIC is applied.
NOTE: The OPTLP presolver removed 0 variables and 0 constraints.
NOTE: The OPTLP presolver removed 0 constraint coefficients.
NOTE: The presolved problem has 64 variables, 20 constraints, and 128
      constraint coefficients.
NOTE: The DUAL SIMPLEX solver is called.
                       Objective
      Phase Iteration  Value
        2           1      -2952213
        2          30      -1281110
NOTE: Optimal.
NOTE: Objective = -1281110.35.
NOTE: The data set WORK.ARC1 has 64 observations and 16 variables.
NOTE: The data set WORK.NODE2 has 20 observations and 3 variables.
```

Example 5.9: Migration to OPTMODEL: Shortest Path

The following example shows how to use PROC OPTMODEL to solve the example "Shortest Path Problem" in Chapter 6, "The NETFLOW Procedure" (*SAS/OR User's Guide: Mathematical Programming Legacy Procedures*). The input data set is the same as in that example.

```
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
;
```

The following PROC OPTMODEL statements read the data sets, build the linear programming model, solve the model, and output the optimal solution to a SAS data set called SPATH:

```
proc optmodel;
   str sourcenode = 'Honolulu';
   str sinknode = 'Heathrow London';

   set <str> NODES;
   num _supdem_ {i in NODES} = (if i = sourcenode then 1
      else if i = sinknode then -1 else 0);

   set <str,str> ARCS;
   num _lo_ {ARCS} init 0;
   num _capac_ {ARCS} init .;
   num _cost_ {ARCS};
   read data aircost1 into ARCS=[ffrom tto] _cost_;
   NODES = (union {<i,j> in ARCS} {i,j});

   var Flow {<i,j> in ARCS} >= _lo_[i,j];
   min obj = sum {<i,j> in ARCS} _cost_[i,j] * Flow[i,j];
   con balance {i in NODES}: sum {<(i),j> in ARCS} Flow[i,j]
      - sum {<j,(i)> in ARCS} Flow[j,i] = _supdem_[i];
   solve;

   num _supply_ {<i,j> in ARCS} =
      (if _supdem_[i] ne 0 then _supdem_[i] else .);
   num _demand_ {<i,j> in ARCS} =
      (if _supdem_[j] ne 0 then -_supdem_[j] else .);
   num _fcost_ {<i,j> in ARCS} = _cost_[i,j] * Flow[i,j].sol;

   create data spath from [ffrom tto]
      _cost_ _capac_ _lo_ _supply_ _demand_ _flow_=Flow _fcost_
      _rcost_=(if Flow[ffrom,tto].rc ne 0 then Flow[ffrom,tto].rc else .)
      _status_=Flow.status;
quit;
```

The statements use both single-dimensional (NODES) and multiple-dimensional (ARCS) index sets. The ARCS data set is populated from the ffrom and tto data set variables in the READ DATA statement. To solve a shortest path problem, you solve a minimum cost network flow problem that has a supply of one unit at the source node, a demand of one unit at the sink node, and zero supply or demand at all other nodes, as specified in the declaration of the _SUPDEM_ numeric parameter. The SPATH output data set contains most of the same information as in the PROC NETFLOW example, including reduced cost and basis status. The _ANUMB_ and _TNUMB_ values do not apply here.

The PROC PRINT statements are similar to the PROC NETFLOW example:

```
proc print data=spath;
   sum _fcost_;
run;
```

The output is displayed in Output 5.9.1.

Output 5.9.1 Output Data Set

Shortest Path Problem
How to get Hawaiian Pineapples to a London Restaurant

Obs	ffrom	tto	_cost_	_capac_	_lo_	_supply_	_demand_	_flow_	_fcost_	_rcost_	_status_
1	Honolulu	Chicago	105	.	0	1	.	0	0	.	B
2	Honolulu	San Francisco	75	.	0	1	.	0	0	.	B
3	Honolulu	Los Angeles	68	.	0	1	.	1	68	.	B
4	Chicago	Boston	45	.	0	.	.	0	0	61	L
5	Chicago	New York	56	.	0	.	.	0	0	49	L
6	San Francisco	Boston	71	.	0	.	.	0	0	57	L
7	San Francisco	New York	48	.	0	.	.	0	0	11	L
8	San Francisco	Atlanta	63	.	0	.	.	0	0	37	L
9	Los Angeles	New York	44	.	0	.	.	1	44	.	B
10	Los Angeles	Atlanta	57	.	0	.	.	0	0	24	L
11	Boston	Heathrow London	88	.	0	.	1	0	0	.	B
12	New York	Heathrow London	65	.	0	.	1	1	65	.	B
13	Atlanta	Heathrow London	76	.	0	.	1	0	0	.	B
									===		
									177		

The log is displayed in Output 5.9.2.

Output 5.9.2 OPTMODEL Log

```
NOTE: There were 13 observations read from the data set WORK.AIRCOST1.
NOTE: The problem has 13 variables (0 free, 0 fixed).
NOTE: The problem has 8 linear constraints (0 LE, 8 EQ, 0 GE, 0 range).
NOTE: The problem has 26 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The OPTMODEL presolver is disabled for linear problems.
NOTE: The OPTLP presolver value AUTOMATIC is applied.
NOTE: The OPTLP presolver removed all variables and constraints.
NOTE: Optimal.
NOTE: Objective = 177.
NOTE: The data set WORK.SPATH has 13 observations and 11 variables.
```

References

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993), *Network Flows*, Prentice-Hall, New Jersey.

Andersen, E. D. and Andersen, K. D. (1995), “Presolving in Linear Programming,” *Mathematical Programming*, 71(2), 221–245.

Dantzig, G. B. (1963), *Linear Programming and Extensions*, Princeton, NJ: Princeton University Press.

Forrest, J. J. and Goldfarb, D. (1992), “Steepest-Edge Simplex Algorithms for Linear Programming,” *Mathematical Programming*, 5, 1–28.

Gondzio, J. (1997), “Presolve Analysis of Linear Programs prior to Applying an Interior Point Method,” *INFORMS Journal on Computing*, 9 (1), 73–91.

Harris, P. M. J. (1973), “Pivot Selection Methods in the Devex LP Code,” *Mathematical Programming*, 57, 341–374.

Maros, I. (2003), *Computational Techniques of the Simplex Method*, Kluwer Academic.