Chapter 1

Introduction to Optimization

Chapter Contents

Chapter 1
Introduction to Optimization

Overview

This chapter describes how to use SAS/OR software to solve a wide variety of optimization problems. The basic optimization problem is that of minimizing or maximizing an objective function subject to constraints imposed on the variables of that function. The objective function and constraints can be linear or nonlinear; the constraints can be bound constraints, equality or inequality constraints, or integer constraints.

Traditionally, optimization problems are divided into linear programming (LP; all functions are linear) and nonlinear programming (NLP). Variations of LP problems are assignment problems, network flow problems, and transportation problems. Nonlinear regression (fitting a nonlinear model to a set of data and the subsequent statistical analysis of the results) is a special NLP problem. Since these applications are so common, SAS/OR software has separate procedures or facilities within procedures for solving each type of these problems. Model data are supplied in a form suited for the particular type of problem. Another benefit is that an optimization algorithm can be specialized for the particular type of problem, reducing solution times. Optimizers can exploit some structure in problems such as embedded networks, special ordered sets, least squares, and quadratic objective functions.

SAS/OR software has seven procedures used for optimization:

- **PROC ASSIGN** for solving assignment problems
- **PROC INTPOINT** for network programming problems with side constraints, and linear programming problems solved by an interior point algorithm
- **PROC LP** for solving linear and mixed integer programming problems
- **PROC NETFLOW** for solving network programming problems with side constraints
- **PROC NLP** for solving nonlinear programming problems
- **PROC QP** for solving quadratic programming problems
- **PROC TRANS** for solving transportation problems

SAS/OR procedures use syntax that is similar to other SAS procedures. In particular, all SAS retrieval, data management, reporting, and analysis can be used with SAS/OR software. Each optimizer is designed to integrate with the SAS System to simplify model building, maintenance, solution, and report writing.

Data for models are supplied to SAS/OR procedures in SAS data sets. These data sets can be saved and easily changed and the problem can be solved. Because the models are in SAS data sets, problem data that can represent pieces of a larger model can be concatenated and merged. The SAS/OR procedures output SAS data sets containing the solutions. These can then be used to produce customized reports. This structure allows decision support systems to be constructed using SAS/OR procedures and other tools in the SAS System as building blocks.

The following list suggests application areas where decision support systems have been used. In practice, models often contain elements of several applications listed here.

- **Product-Mix problems** find the mix of products that generates the largest return when there are several products that compete for limited resources.
- **Blending problems** find the mix of ingredients to be used in a product so that it meets minimum standards at minimum cost.
- **Time-Staged problems** are models whose structure repeats as a function of time. Production and inventory models are classic examples of time-staged problems. In each period, production plus inventory minus current demand equals inventory carried to the next period.
- **Scheduling problems** assign people to times, places, or tasks so as to optimize people's preferences while satisfying the demands of the schedule.
- **Multiple objective problems** have multiple conflicting objectives. Typically, the objectives are prioritized and the problems are solved sequentially in a priority order.
- **Capital budgeting and project selection problems** ask for the project or set of projects that will yield the greatest return.
- **Location problems** seek the set of locations that meets the distribution needs at minimum cost.
- **Cutting stock problems** find the partition of raw material that minimizes waste.

Data Flow

The LP, NETFLOW, INTPOINT, NLP, QP, TRANS, and ASSIGN procedures take a model that has been saved in one or more SAS data sets, solve it, and save the solution in other SAS data sets. Most of the procedures define a SAS macro variable that contains a character string indicating whether or not the procedure terminated successfully and the status of the optimizer (for example, whether the optimum was found). This information is useful when the procedure is one of the steps in a larger program.

PROC LP

The LP procedure solves linear and mixed integer programs. It can perform several types of post-optimality analysis, including range analysis, sensitivity analysis, and parametric programming. The procedure can also be used interactively.

PROC LP requires a problem data set that contains the model. In addition, a primal and active data set can be used for warm starting a problem that has been partially solved previously.

Figure 1.1 illustrates all the input and output data sets that are possible with PROC LP. It also shows the macro variable _ORLP_ that PROC LP defines.

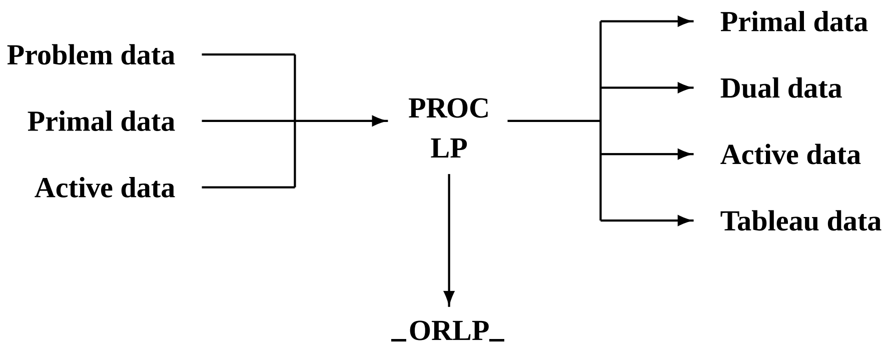


Figure 1.1. Data Flow in PROC LP

The problem data describing the model can be in one of two formats: a sparse or a dense format. The dense format represents the model as a rectangular matrix. The sparse format represents only the nonzero elements of a rectangular matrix. The sparse and dense input formats are described in more detail later in this chapter.

PROC NETFLOW

The NETFLOW procedure solves network flow problems with linear side constraints using either the network simplex algorithm or the interior point algorithm. In addition, it can solve linear programming (LP) problems using the interior point algorithm.

Networks and the Network Simplex Algorithm

PROC NETFLOW's network simplex algorithm solves pure network flow problems and network flow problems with linear side constraints. The procedure accepts the network specification in a format that is particularly suited to networks. Although network problems could be solved by PROC LP, the NETFLOW procedure generally solves network flow problems more efficiently than PROC LP.

Network flow problems, such as finding the minimum cost flow in a network, require model representation in a format that is simpler than PROC LP. The network is represented in two data sets: a node data set that names the nodes in the network and gives supply and demand information at them, and an arc data set that defines the arcs in the network using the node names and gives arc costs and capacities. In addition, a side-constraint data set is included that gives any side constraints that apply to the flow through the network. Examples of these are found later in this chapter.

The NETFLOW procedure saves solutions in four data sets. Two of these store solutions for the pure network model, ignoring the restrictions imposed by the side constraints. The remaining two data sets contain the solutions to the network flow problem when the side constraints apply.

Figure 1.2 illustrates the input and output data sets that are possible with PROC NETFLOW when using the network simplex method. It also shows the macro variable _ORNETFL that PROC NETFLOW defines.

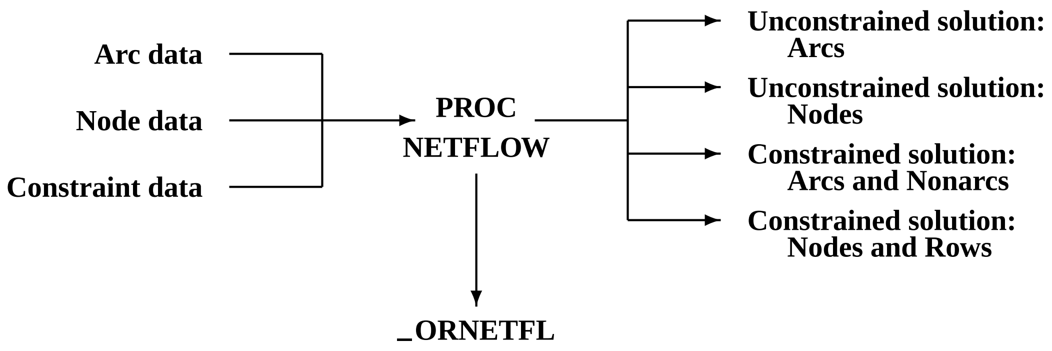

Arc data
Node data
Constraint data
PROC NETFLOW
Unconstrained solution: Arcs
Unconstrained solution: Nodes
Constrained solution: Arcs and Nonarcs
Constrained solution: Nodes and Rows
_ORNETFL

Figure 1.2. Data Flow in PROC NETFLOW: Simplex Algorithm

The constraint data can be specified in either the sparse or dense input formats. This is the same format that is used by PROC LP; therefore, any model-building techniques that apply to models for PROC LP also apply to network flow models having side constraints.

Linear and Network Programs Solved by the Interior Point Algorithm

The data required by PROC NETFLOW for a linear program resembles the data for nonarc variables and constraints for constrained network problems. It is similar to the data required by PROC LP.

The LP representation requires a data set that defines the variables in the LP using variable names, and gives objective function coefficients and upper and lower bounds. In addition, a constraint data set can be included that specifies any constraints.

Figure 1.3 illustrates the input and output data sets that are possible with PROC NETFLOW for solving linear programs using the interior point algorithm. It also shows the macro variable _ORNETFL that PROC NETFLOW defines.

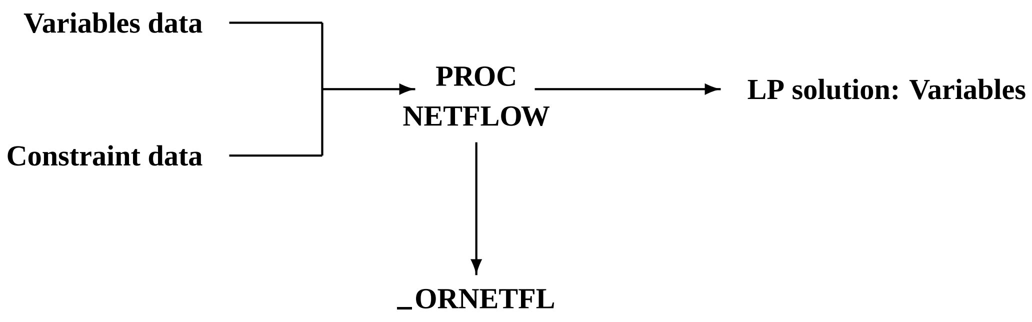


Figure 1.3. Data Flow in PROC NETFLOW: LP Problems

When solving a constrained network problem, you can specify the INTPOINT option to indicate that the interior point algorithm is to be used. The input data is the same whether the simplex or interior point method is used. The interior point method is often faster when problems have many side constraints.

Figure 1.4 illustrates the input and output data sets that are possible with PROC NETFLOW for solving network problems using the interior point algorithm. It also shows the macro variable _ORNETFL that PROC NETFLOW defines.

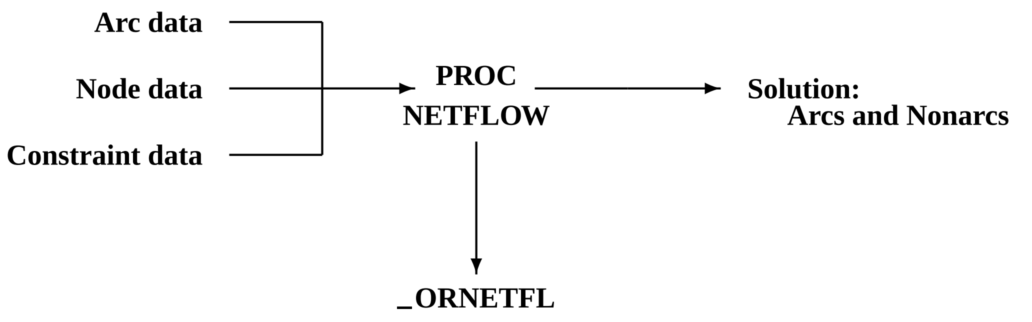


Figure 1.4. Data Flow in PROC NETFLOW: Interior Point Algorithm

The constraint data can be specified in either the sparse or dense input format. This is the same format that is used by PROC LP; therefore, any model-building techniques that apply to models for PROC LP also apply to LP models solved by PROC NETFLOW.

PROC INTPOINT

The INTPOINT procedure solves the Network Program with Side Constraints (NPSC) problem and the more general Linear Programming (LP) problem using the interior point algorithm.

The data required by PROC INTPOINT is similar to the data required by PROC NETFLOW when solving network flow models using the interior point algorithm.

Figure 1.5 illustrates the input and output data sets that are possible with PROC INTPOINT.

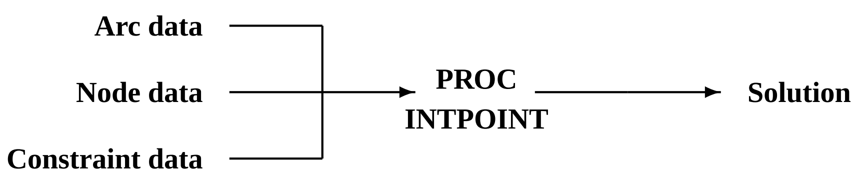


Figure 1.5. Data Flow in PROC INTPOINT

The constraint data can be specified in either the sparse or dense input format. This is the same format that is used by PROC LP and PROC NETFLOW; therefore, any model-building techniques that apply to models for PROC LP or PROC NETFLOW also apply to PROC INTPOINT.

PROC NLP

The NLP procedure (**N**on**L**inear **P**rogramming) offers a set of optimization techniques for minimizing or maximizing a continuous nonlinear function subject to linear and nonlinear, equality and inequality, and lower and upper bound constraints. Problems of this type are found in many settings ranging from optimal control to maximum likelihood estimation.

Nonlinear programs can be input into the procedure in various ways. The objective, constraint, and derivative functions are specified using the programming statements of PROC NLP. In addition, information in SAS data sets can be used to define the structure of objectives and constraints, and to specify constants used in objectives, constraints, and derivatives.

PROC NLP uses data sets to input various pieces of information:

- The DATA= data set enables you to specify data shared by all functions involved in a least squares problem.
- The INQUAD= data set contains the arrays appearing in a quadratic programming problem.
- The INEST= data set specifies initial values for the decision variables, the values of constants that are referred to in the program statements, and simple boundary and general linear constraints.
- The MODEL= data set specifies a model (functions, constraints, derivatives) saved at a previous execution of the NLP procedure.

PROC NLP uses data sets to output various results:

- The OUTEST= data set saves the values of the decision variables, the derivatives, the solution, and the covariance matrix at the solution.

- The OUT= output data set contains variables generated in the program statements defining the objective function, as well as selected variables of the DATA= input data set, if available.
- The OUTMODEL= data set saves the programming statements. It can be used to input a model in the MODEL= input data set.

Figure 1.6 illustrates all the input and output data sets that are possible with PROC NLP.

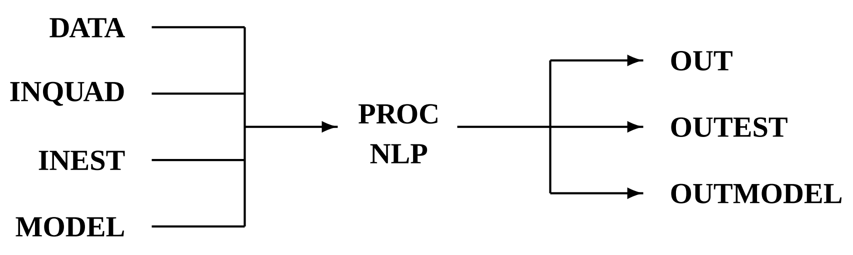


Figure 1.6. Data Flow in PROC NLP

As an alternative to supplying data in SAS data sets, some or all data for the model can be specified using SAS programming statements. These are similar to those used in the SAS DATA step.

PROC QP (Experimental)

The experimental QP procedure solves Quadratic Programming (QP) problems and Quadratic Network Problems with Side Constraints (QNPSC).

The data required by PROC QP is similar to the data required by PROC INTPOINT, with the addition of a Hessian matrix that must be specified in a data set. Figure 1.7 illustrates the input and output data sets that are possible with PROC QP.

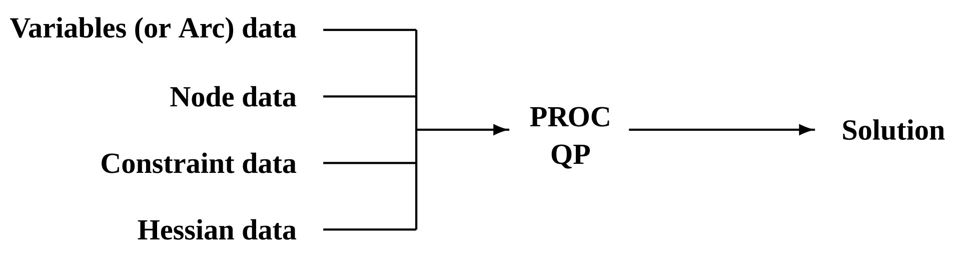


Figure 1.7. Data Flow in PROC QP

The constraint data can be specified in either the sparse or dense input format. This is the same format that is used by PROC LP, PROC NETFLOW, and PROC INTPOINT; therefore, any model-building techniques that apply to these models also apply to PROC QP.

PROC TRANS

Transportation networks are a special type of network, called *bipartite* networks, that have only supply and demand nodes and arcs directed from supply nodes to demand nodes. For these networks, data can be given most efficiently in a rectangular or matrix form. The TRANS procedure takes cost, capacity, and lower bound data in this form. The observations in these data sets correspond to supply nodes, and the variables correspond to demand nodes. The solution is saved in a single output data set.

Figure 1.8 illustrates the input and output data sets that are possible with PROC TRANS. It also shows the macro variable _ORTRANS that PROC TRANS defines.

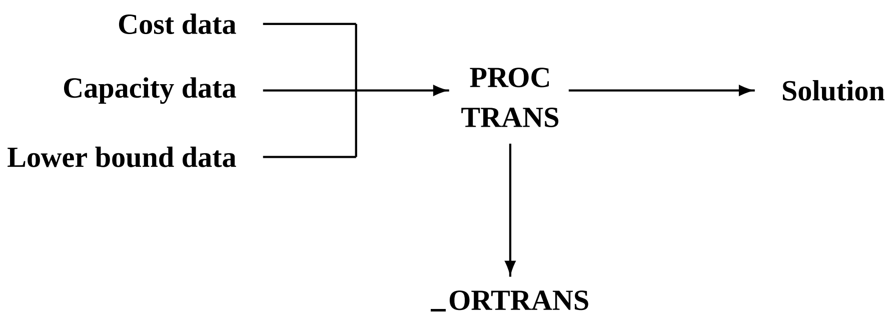


Figure 1.8. Data Flow in PROC TRANS

PROC ASSIGN

The assignment problem is a special type of transportation problem, one having supply and demand values of one unit. As with the transportation problem, the cost data for this type of problem are saved in a SAS data set in rectangular form. The ASSIGN procedure saves the solution in a SAS data set.

Figure 1.9 illustrates the input and output data sets that are possible with PROC ASSIGN. It also shows the macro variable _ORASSIG that PROC ASSIGN defines.

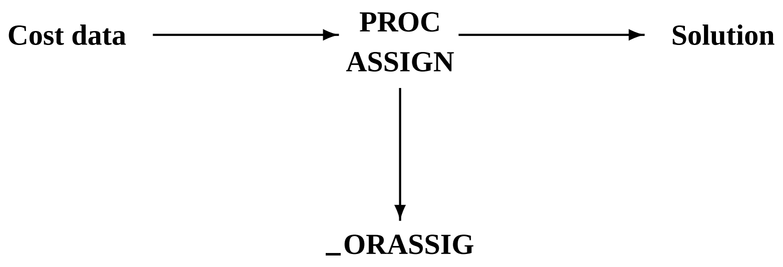


Figure 1.9. Data Flow in PROC ASSIGN

Model Formats: PROC LP and PROC NETFLOW

Model generation and maintenance are often difficult and expensive aspects of applying mathematical programming techniques. The flexible input formats for the optimization procedures in SAS/OR software simplify this task.

A small product mix problem serves as a starting point for a discussion of different types of model formats supported in SAS/OR software.

A candy manufacturer makes two products: chocolates and toffee. What combination of chocolates and toffee should be produced in a day in order to maximize the company's profit? Chocolates contribute $0.25 per pound to profit, and toffee contributes $0.75 per pound. The decision variables are *chocolates* and *toffee*.

Four processes are used to manufacture the candy:

1. Process 1 combines and cooks the basic ingredients for both chocolates and toffee.
2. Process 2 adds colors and flavors to the toffee, then cools and shapes the confection.
3. Process 3 chops and mixes nuts and raisins, adds them to the chocolates, then cools and cuts the bars.
4. Process 4 is packaging: chocolates are placed in individual paper shells; toffee are wrapped in cellophane packages.

During the day, there are 7.5 hours (27,000 seconds) available for each process.

Firm time standards have been established for each process. For Process 1, mixing and cooking take 15 seconds for each pound of chocolate, and 40 seconds for each pound of toffee. Process 2 takes 56.25 seconds per pound of toffee. For Process 3, each pound of chocolate requires 18.75 seconds of processing. In packaging, a pound of chocolates can be wrapped in 12 seconds, whereas 50 seconds are required for a pound of toffee. These data are summarized below:

```
                    Available       Required per Pound
                      Time         chocolates     toffee
  Process             (sec)          (sec)         (sec)
1 Cooking             27,000          15            40
2 Color/Flavor        27,000                       56.25
3 Condiments          27,000         18.75
4 Packaging           27,000          12            50
```

The objective is to

Maximize: 0.25(*chocolates*) + 0.75(*toffee*)

which is the company's total profit.

The production of the candy is limited by the time available for each process. The limits placed on production by Process 1 are expressed by the following inequality.

Process 1: $15(\textit{chocolates}) + 40(\textit{toffee}) \leq 27{,}000$

Process 1 can handle any combination of chocolates and toffee that satisfies this inequality.

The limits on production by other processes generate constraints described by the following inequalities.

Process 2: $56.25(\textit{toffee}) \leq 27{,}000$

Process 3: $18.75(\textit{chocolates}) \leq 27{,}000$

Process 4: $12(\textit{chocolates}) + 50(\textit{toffee}) \leq 27{,}000$

This linear program illustrates the type of problem known as a product mix example. The mix of products that maximizes the objective without violating the constraints is the solution. Two formats — dense or sparse — can be used to represent this model.

Dense Format

The following DATA step creates a SAS data set for this product mix problem. Notice that the values of CHOCO and TOFFEE in the data set are the coefficients of those variables in the equations corresponding to the objective function and constraints. The variable _id_ contains a character string that names the rows in the data set. The variable _type_ is a character variable that contains keywords that describes the type of each row in the problem data set. The variable _rhs_ contains the right-hand-side values.

```
data factory;
   input _id_ $ CHOCO TOFFEE _type_ $ _rhs_;
   datalines;
object      0.25    0.75   MAX     .
process1   15.00   40.00   LE   27000
process2    0.00   56.25   LE   27000
process3   18.75    0.00   LE   27000
process4   12.00   50.00   LE   27000
;
```

To solve this problem using the interior point algorithm of PROC NETFLOW, specify

```
proc netflow arcdata=factory condata=factory;
```

However, this example will be solved by the LP procedure. Because the special variables _id_, _type_, and _rhs_ are used in the problem data set, there is no need to identify them to the LP procedure. Therefore, the following statement is all that is needed to solve this problem.

```
proc lp;
```

The output from the LP procedure is in four sections.

Problem Summary

The first section of the output, the Problem Summary, describes the problem by identifying the objective function (defined by the first observation in the data set used as input), the right-hand-side variable, the type variable, and the density of the problem. The problem density describes the relative number of elements in the problem matrix that are nonzero. The fewer zeros in the matrix, the higher the problem density. The Problem Summary describes the problem, giving the number and type of variables in the model and the number and type of constraints. The types of variables in the problem are also identified. Variables are either structural or logical. Structural variables are identified in the VAR statement when the dense format is used. They are the unknowns in the equations defining the objective function and constraints. By default, PROC LP assumes that structural variables have the additional constraint that they must be nonnegative. Upper and lower bounds to structural variables can be defined.

```
                              The LP Procedure

                              Problem Summary

                     Objective Function     Max object
                     Rhs Variable                _rhs_
                     Type Variable              _type_
                     Problem Density (%)         41.67

                     Variables                  Number

                     Non-negative                    2
                     Slack                           4

                     Total                           6

                     Constraints                Number

                     LE                              4
                     Objective                       1

                     Total                           5
```

Figure 1.10. Problem Summary

The Problem Summary shows, for example, that there are two nonnegative decision variables, namely CHOCO and TOFFEE. It also shows that there are four constraints of type LE.

After the procedure displays this information, it solves the problem and displays the Solution Summary.

Solution Summary

The Solution Summary (shown in Figure 1.11) gives information about the solution that was found, including whether the optimizer terminated successfully, having found the optimum.

When PROC LP solves a problem, an iterative process is used. First, the procedure finds a feasible solution that satisfies the constraints. The second phase finds the optimal solution from the set of feasible solutions. The Solution Summary lists the number of iterations in each of these phases, the number of variables in the initial feasible solution, the time the procedure used to solve the problem, and the number of matrix inversions necessary.

```
                                The LP Procedure

                                Solution Summary

                             Terminated Successfully

                Objective Value                              475

                Phase 1 Iterations                             0
                Phase 2 Iterations                             3
                Phase 3 Iterations                             0
                Integer Iterations                             0
                Integer Solutions                              0
                Initial Basic Feasible Variables               6
                Time Used (seconds)                            0
                Number of Inversions                           3

                Epsilon                                     1E-8
                Infinity                            1.797693E308
                Maximum Phase 1 Iterations                   100
                Maximum Phase 2 Iterations                   100
                Maximum Phase 3 Iterations              99999999
                Maximum Integer Iterations                   100
                Time Limit (seconds)                         120
```

Figure 1.11. Solution Summary

After performing three Phase 2 iterations, the procedure terminated successfully with optimal objective value of 475.

Variable Summary

The next section of the output is the Variable Summary, as shown in Figure 1.12. For each variable, the Variable Summary gives the value, objective function coefficient, status in the solution, and reduced cost.

```
                                The LP Procedure

                                Variable Summary

                 Variable                                         Reduced
             Col Name     Status Type        Price  Activity         Cost

               1 CHOCO    BASIC  NON-NEG      0.25      1000            0
               2 TOFFEE   BASIC  NON-NEG      0.75       300            0
               3 process1        SLACK           0         0    -0.012963
               4 process2 BASIC  SLACK           0     10125            0
               5 process3 BASIC  SLACK           0      8250            0
               6 process4        SLACK           0         0     -0.00463
```

Figure 1.12. Variable Summary

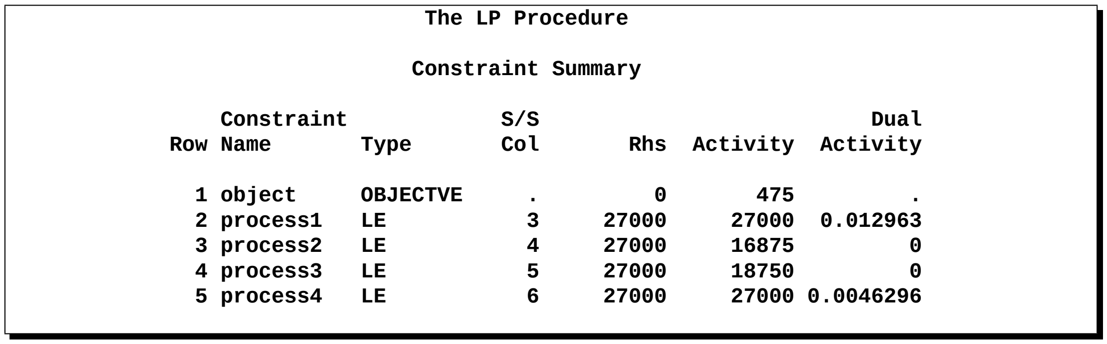
The Variable Summary contains details about each variable in the solution. The Activity variable shows that optimum profitability is achieved when 1000 pounds of chocolate and 300 pounds of toffee are produced. The variables process1, process2, process3, and process4 correspond to the four slack variables in the Process 1, Process 2, Process 3, and Process 4 constraints, respectively. Producing 1000 pounds of chocolate and 300 pounds of toffee a day leaves 10,125 seconds of slack time in Process 2 (where colors and flavors are added to the toffee), and 8,250 seconds of slack time in Process 3 (where nuts and raisins are mixed and added to the chocolate).

Constraint Summary

The last section of the output is the Constraint Summary, as shown in Figure 1.13. The Constraint Summary gives the value of the objective function, the value of each constraint, and the dual activities.

The Activity variable gives the value of the right-hand side of each equation when the problem is solved using the information given in the Variable Summary.

The Dual Activity variable reveals that each second in Process 1 (mixing-cooking) is worth approximately $.013, and each second in Process 4 (Packaging) is worth approximately $.005. These figures (called *shadow prices*) can be used to decide whether the total available time for Process 1 and Process 4 should be increased. If a second can be added to the total production time in Process 1 for less than $.013, it would be profitable to do so. The dual activities for Process 2 and Process 3 are zero, since adding time to those processes does not increase profits. Keep in mind that the dual activity gives the marginal improvement to the objective, and that adding time to Process 1 changes the original problem and solution.



```
                          The LP Procedure

                         Constraint Summary

           Constraint            S/S                         Dual
       Row Name        Type      Col       Rhs  Activity  Activity

         1 object      OBJECTVE    .         0       475         .
         2 process1    LE          3     27000     27000  0.012963
         3 process2    LE          4     27000     16875         0
         4 process3    LE          5     27000     18750         0
         5 process4    LE          6     27000     27000 0.0046296
```

Figure 1.13. Constraint Summary

For a complete description of the output from PROC LP, see Chapter 4, “The LP Procedure.”

Sparse Format

Typically, mathematical programming models are sparse. That is, few of the coefficients in the constraint matrix are nonzero. The dense problem format shown in the previous section is an inefficient way to represent sparse models. The LP procedure also accepts data in a sparse input format. Only the nonzero coefficients must be specified. It is consistent with the standard MPS sparse format, and much more flexible; models using the MPS format can be easily converted to the LP format.

Although the factory example of the last section is not sparse, an example of the sparse input format for that problem is illustrated here. The sparse data set has four variables: a row type identifying variable (_type_), a row name variable (_row_), a column name variable (_col_), and a coefficient variable (_coef_).

```
data factory;
   format _type_ $8. _row_ $10. _col_ $10.;
   input _type_ $_row_ $ _col_ $ _coef_ ;
   datalines;
max      object     .            .
.        object     chocolate    .25
.        object     toffee       .75
le       process1   .            .
.        process1   chocolate    15
.        process1   toffee       40
.        process1   _RHS_        27000
le       process2   .            .
.        process2   toffee       56.25
.        process2   _RHS_        27000
le       process3   .            .
.        process3   chocolate    18.75
.        process3   _RHS_        27000
le       process4   .            .
.        process4   chocolate    12
.        process4   toffee       50
.        process4   _RHS_        27000
;
```

To solve this problem using the interior point algorithm of PROC NETFLOW, specify

```
proc netflow sparsecondata arcdata=factory condata=factory;
```

However, this example will be solved by the LP procedure.

Notice that the _type_ variable contains keywords as for the dense format, the _row_ variable contains the row names in the model, the _col_ variable contains the column names in the model, and the _coef_ variable contains the coefficients for that particular row and column. Since the row and column names are the values of variables in a SAS data set, they are not limited to eight characters. This feature, as well as the order independence of the format, simplifies matrix generation.

The SPARSEDATA option in the PROC LP statement tells the LP procedure that the model in the problem data set is in the sparse format. This example also illustrates how the solution of the linear program is saved in two output data sets: the primal data set and the dual data set.

```
proc lp
   data=factory sparsedata
   primalout=primal dualout=dual;
   run;
```

The primal data set (shown in Figure 1.14) contains the information that is displayed in the Variable Summary, plus additional information about the bounds on the variables.

```
     Obs    _OBJ_ID_    _RHS_ID_    _VAR_               _TYPE_     _STATUS_

      1      object      _RHS_      chocolate           NON-NEG    _BASIC_
      2      object      _RHS_      toffee              NON-NEG    _BASIC_
      3      object      _RHS_      process1            SLACK
      4      object      _RHS_      process2            SLACK      _BASIC_
      5      object      _RHS_      process3            SLACK      _BASIC_
      6      object      _RHS_      process4            SLACK
      7      object      _RHS_      PHASE_1_OBJECTIV    OBJECT     _DEGEN_
      8      object      _RHS_      object              OBJECT     _BASIC_

     Obs    _LBOUND_    _VALUE_     _UBOUND_     _PRICE_     _R_COST_

      1         0         1000     1.7977E308      0.25      0.000000
      2         0          300     1.7977E308      0.75      0.000000
      3         0            0     1.7977E308      0.00     -0.012963
      4         0        10125     1.7977E308      0.00      0.000000
      5         0         8250     1.7977E308      0.00      0.000000
      6         0            0     1.7977E308      0.00     -0.004630
      7         0            0              0      0.00      0.000000
      8         0          475     1.7977E308      0.00      0.000000
```

Figure 1.14. Primal Data Set

The dual data set (shown in Figure 1.15) contains the information that is displayed in the Constraint Summary, plus additional information about bounds on the rows.

```
Obs _OBJ_ID_ _RHS_ID_ _ROW_ID_ _TYPE_ _RHS_      _L_RHS_ _VALUE_ _U_RHS_   _DUAL_

 1   object   _RHS_   object   OBJECT   475         475     475     475    .
 2   object   _RHS_   process1 LE     27000 -1.7977E308  27000   27000  0.012963
 3   object   _RHS_   process2 LE     27000 -1.7977E308  16875   27000  0.000000
 4   object   _RHS_   process3 LE     27000 -1.7977E308  18750   27000  0.000000
 5   object   _RHS_   process4 LE     27000 -1.7977E308  27000   27000  0.004630
```

Figure 1.15. Dual Data Set

Network Format

Network flow problems can be described by specifying the nodes in the network and their supplies and demands, and the arcs in the network and their costs, capacities, and lower flow bounds. Consider the simple transshipment problem in Figure 1.16 as an illustration.

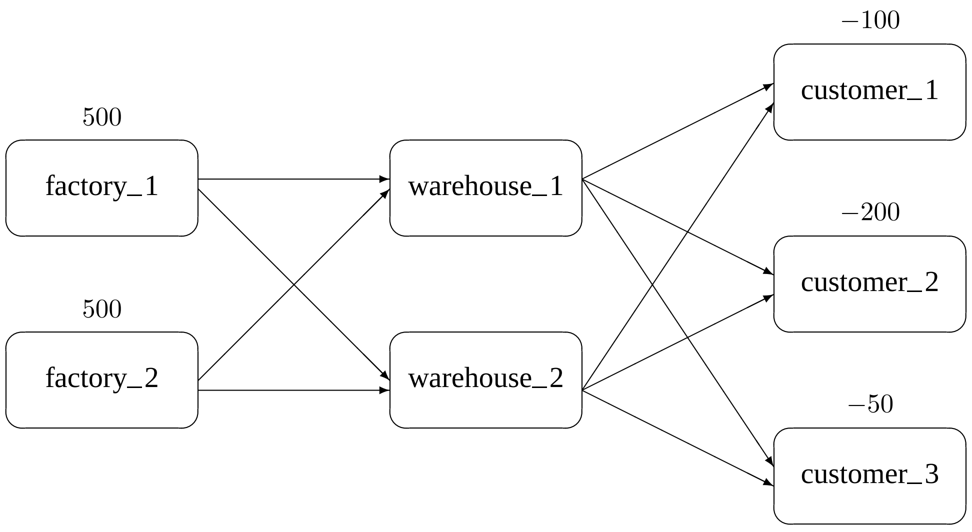


Figure 1.16. Transshipment Problem

Suppose the candy manufacturing company has two factories, two warehouses, and three customers for chocolate. The two factories each have a production capacity of 500 pounds per day. The three customers have demands of 100, 200, and 50 pounds per day, respectively.

The following data set describes the supplies (positive values for the supdem variable) and the demands (negative values for the supdem variable) for each of the customers and factories.

```
data nodes;
   format node $10. ;
   input node $ supdem;
   datalines;
customer_1   -100
customer_2   -200
customer_3    -50
factory_1     500
factory_2     500
;
```

Suppose that there are two warehouses that are used to store the chocolate before shipment to the customers, and that there are different costs for shipping between each factory, warehouse, and customer. What is the minimum cost routing for supplying the customers?

Arcs are described in another data set. Each observation defines a new arc in the network and gives data about the arc. For example, there is an arc between the node factory_1 and the node warehouse_1. Each unit of flow on that arc costs 10.

Although this example does not include it, lower and upper bounds on the flow across that arc can be listed here.

```
data network;
   format from $12. to $12.;
   input from $ to $ cost ;
   datalines;
factory_1     warehouse_1  10
factory_2     warehouse_1   5
factory_1     warehouse_2   7
factory_2     warehouse_2   9
warehouse_1   customer_1    3
warehouse_1   customer_2    4
warehouse_1   customer_3    4
warehouse_2   customer_1    5
warehouse_2   customer_2    5
warehouse_2   customer_3    6
;
```

You can use PROC NETFLOW to find the minimum cost routing. This procedure takes the model as defined in the network and nodes data sets and finds the minimum cost flow.

```
proc netflow arcout=arc_sav
             arcdata=network nodedata=nodes;
   node node;      /* node data set information */
   supdem supdem;
   tail from;      /* arc data set information */
   head to;
   cost cost;
   run;

proc print;
   var from to cost _capac_ _lo_ _supply_ _demand_
       _flow_ _fcost_ _rcost_;
   sum _fcost_;
   run;
```

PROC NETFLOW produces the following messages on the SAS log:

```
NOTE: Number of nodes= 7 .
NOTE: Number of supply nodes= 2 .
NOTE: Number of demand nodes= 3 .
NOTE: Total supply= 1000 , total demand= 350 .
NOTE: Number of arcs= 10 .
NOTE: Number of iterations performed (neglecting
      any constraints)= 7 .
NOTE: Of these, 2 were degenerate.
NOTE: Optimum (neglecting any constraints) found.
NOTE: Minimal total cost= 3050 .
NOTE: The data set WORK.ARC_SAV has 10 observations
      and 13 variables.
```

The solution (Figure 1.17) saved in the arc_sav data set shows the optimal amount of chocolate to send across each arc (the amount to ship from each factory to each warehouse and from each warehouse to each customer) in the network per day.

```
                                                          _    _
                                                _         S    D           _  _
                                                C         U    E     _     F  R
                                                A         P    M     F     C  C
    f                                    c      P     _   P    A     L     O  O
 O  r                                    o      A     L   L    N     O     S  S
 b  o                 t                  s      C     O   Y    D     W     T  T
 s  m                 o                  t      _     _   _    _     _     _  _

 1  warehouse_1       customer_1         3  99999999  0    .  100  100   300  .
 2  warehouse_2       customer_1         5  99999999  0    .  100    0     0  4
 3  warehouse_1       customer_2         4  99999999  0    .  200  200   800  .
 4  warehouse_2       customer_2         5  99999999  0    .  200    0     0  3
 5  warehouse_1       customer_3         4  99999999  0    .   50   50   200  .
 6  warehouse_2       customer_3         6  99999999  0    .   50    0     0  4
 7  factory_1         warehouse_1       10  99999999  0  500    .    0     0  5
 8  factory_2         warehouse_1        5  99999999  0  500    .  350  1750  .
 9  factory_1         warehouse_2        7  99999999  0  500    .    0     0  .
10  factory_2         warehouse_2        9  99999999  0  500    .    0     0  2
                                                                        ====
                                                                        3050
```

Figure 1.17. ARCOUT Data Set

Notice which arcs have positive flow (_FLOW_ is greater than 0). These arcs indicate the amount of chocolate that should be sent from factory_2 to warehouse_1 and from there on to the three customers. The model indicates no production at factory_1 and no use of warehouse_2.

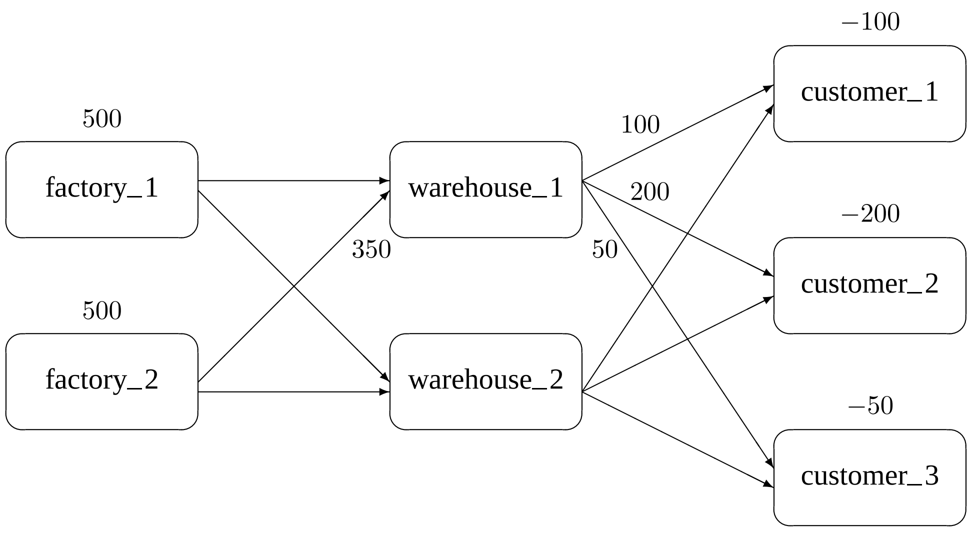


Figure 1.18. Optimal Solution for the Transshipment Problem

Model Formats: PROC ASSIGN and PROC TRANS

The transportation and assignment models are described in rectangular data sets. Suppose that instead of sending chocolate from factories to warehouses and then to the customers, chocolate is sent directly from the factories to the customers.

Finding the minimum cost routing could be done using the NETFLOW procedure. However, since the network represents a transportation problem, the data for the problem can be represented more simply.

```
data transprt;
   input source $ supply cust_1 cust_2 cust_3 ;
   datalines;
demand        .   100    200     50
factory1   500    10      9       7
factory2   500     9     10       8
;
```

This data set shows the source names as the values for the source variable, the supply at each source node as the values for the supply variable, and the unit shipping cost for source to sink as the values for the sink variables cust_1 to cust_3. Notice that the first record contains the demands at each of the sink nodes.

The TRANS procedure finds the minimum cost routing. It solves the problem and saves the solution in an output data set.

```
proc trans
   nothrunet data=transprt out=transout;
   supply supply;
   id source;

proc print; run;
```

The optimum solution total (3050) is reported on the SAS log. The entire solution, saved in the output data set transout and shown in Figure 1.19, shows the amount of chocolate to ship from each factory to each customer per day.

The transout data set contains the variables listed in the transprt data set, and a new variable called _DUAL_. The _DUAL_ variable contains the marginal costs of increasing the supply at each origin point. The last observation in the transout data set has the marginal costs of increasing the demand at each destination point. These variables are called dual variables.

```
Obs    source     supply    cust_1    cust_2    cust_3    _DUAL_

 1     _DEMAND_      .        100       200       50         .
 2     factory1     500         0       200       50         0
 3     factory2     500       100         0        0         0
 4     _DUAL_        .          9         9        7         .
```

Figure 1.19. TRANSOUT Data Set

Model Building

It is often desirable to keep the data separate from the structure of the model. This is useful for large models with numerous identifiable components. The data are best organized in rectangular tables that can be easily examined and modified. Then, before the problem is solved, the model is built using the stored data. This process of model building is known as *matrix generation*. In conjunction with the sparse format, the SAS DATA step provides a good matrix generation language.

For example, consider the candy manufacturing example introduced previously. Suppose that, for the user interface, it is more convenient to organize the data so that each record describes the information related to each product (namely, the contribution to the objective function and the unit amount needed for each process). A DATA step for saving the data might look like this:

```
data manfg;
   format product $12.;
   input product $ object process1 - process4 ;
   datalines;
chocolate        .25    15  0.00 18.75    12
toffee           .75    40 56.25  0.00    50
licorice        1.00    29 30.00 20.00    20
jelly_beans      .85    10  0.00 30.00    10
_RHS_            .   27000 27000 27000 27000
;
```

Notice that there is a special record at the end having product _RHS_. This record gives the amounts of time available for each of the processes. This information could have been stored in another data set. The next example illustrates a model where the data are stored in separate data sets.

Building the model involves adding the data to the structure. There are as many ways to do this as there are programmers and problems. The following DATA step shows one way to use the candy data to build a sparse format model to solve the product mix problem.

```
data model;
   array process object process1-process4;
   format _type_ $8. _row_ $12. _col_ $12. ;
   keep _type_ _row_ _col_ _coef_;

   set manfg;         /* read the manufacturing data */

   /* build the object function */

   if _n_=1 then do;
      _type_='max'; _row_='object'; _col_=' '; _coef_=.;
      output;
   end;

   /* build the constraints */
```

```
      do over process;
         if _i_>1 then do;
            _type_='le'; _row_='process'||put(_i_-1,1.);
         end;
         else             _row_='object';
         _col_=product; _coef_=process;
         output;
      end;
   run;
```

The sparse format data set is shown in Figure 1.20.

```
      Obs    _type_     _row_              _col_                 _coef_

        1    max        object                                      .
        2    max        object             chocolate              0.25
        3    le         process1           chocolate             15.00
        4    le         process2           chocolate              0.00
        5    le         process3           chocolate             18.75
        6    le         process4           chocolate             12.00
        7               object             toffee                 0.75
        8    le         process1           toffee                40.00
        9    le         process2           toffee                56.25
       10    le         process3           toffee                 0.00
       11    le         process4           toffee                50.00
       12               object             licorice               1.00
       13    le         process1           licorice              29.00
       14    le         process2           licorice              30.00
       15    le         process3           licorice              20.00
       16    le         process4           licorice              20.00
       17               object             jelly_beans            0.85
       18    le         process1           jelly_beans           10.00
       19    le         process2           jelly_beans            0.00
       20    le         process3           jelly_beans           30.00
       21    le         process4           jelly_beans           10.00
       22               object             _RHS_                    .
       23    le         process1           _RHS_              27000.00
       24    le         process2           _RHS_              27000.00
       25    le         process3           _RHS_              27000.00
       26    le         process4           _RHS_              27000.00
```

Figure 1.20. Sparse Data Format

The model data set looks a little different from the sparse representation of the candy model shown earlier. It not only includes additional products (licorice and jelly_beans), but it also defines the model in a different order. Since the sparse format is robust, the model can be generated in ways that are convenient for the DATA step program.

If the problem had more products, you could increase the size of the manfg data set to include the new product data. Also, if the problem had more than four processes, you could add the new process variables to the manfg data set and increase the size of the process array in the model data set. With these two simple changes and additional data, a product mix problem having hundreds of processes and products can be solved.

Matrix Generation

It is desirable to keep data in separate tables, then automate model building and reporting. This example illustrates a problem that has elements of a product mix problem and a blending problem. Suppose four kinds of ties are made; all silk, all polyester, a 50-50 polyester-cotton blend, and a 70-30 cotton-polyester blend.

The data includes cost and supplies of raw material, selling price, minimum contract sales, maximum demand of the finished products, and the proportions of raw materials that go into each product. The product mix that maximizes profit is to be found.

The data are saved in three SAS data sets. The program that follows demonstrates one way for these data to be saved. Alternatively, the full-screen editor PROC FSEDIT can be used to store and edit these data.

```
data material;
   format descpt $20.;
   input descpt $ cost supply;
   datalines;
silk_material             .21   25.8
polyester_material        .6    22.0
cotton_material           .9    13.6
;

data tie;
   format descpt $20.;
   input descpt $ price contract demand;
   datalines;
all_silk                  6.70    6.0    7.00
all_polyester             3.55   10.0   14.00
poly_cotton_blend         4.31   13.0   16.00
cotton_poly_blend         4.81    6.0    8.50
;

data manfg;
   format descpt $20.;
   input descpt $ silk poly cotton;
   datalines;
all_silk                   100     0      0
all_polyester                0   100      0
poly_cotton_blend            0    50     50
cotton_poly_blend            0    30     70
;
```

The following program takes the raw data from the three data sets and builds a linear program model in the data set called model. Although it is designed for the three-resource, four-product problem described here, it can be easily extended to include more resources and products. The model-building DATA step remains essentially the same; all that changes are the dimensions of loops and arrays. Of course, the data tables must increase to accommodate the new data.

```
data model;
   array  raw_mat {3} $ 20 ;
   array  raw_comp {3} silk poly cotton;
   length _type_ $ 8  _col_ $ 20  _row_ $ 20 _coef_ 8 ;
   keep   _type_      _col_       _row_      _coef_ ;

   /* define the objective, lower, and upper bound rows */

   _row_='profit'; _type_='max';     output;
   _row_='lower';  _type_='lowerbd'; output;
   _row_='upper';  _type_='upperbd'; output;
   _type_=' ';

   /* the object and upper rows for the raw materials */

   do i=1 to 3;
      set material;
      raw_mat[i]=descpt;  _col_=descpt;
      _row_='profit';    _coef_=-cost;  output;
      _row_='upper';     _coef_=supply; output;
   end;

   /* the object, upper, and lower rows for the products */

   do i=1 to 4;
      set tie;
      _col_=descpt;
      _row_='profit'; _coef_=price;    output;
      _row_='lower';  _coef_=contract; output;
      _row_='upper';  _coef_=demand;   output;
   end;

   /* the coefficient matrix for manufacturing */

   _type_='eq';
   do i=1 to 4;          /* loop for each raw material */
      set manfg;
      do j=1 to 3;       /* loop for each product      */

         _col_=descpt;   /* % of material in product   */
         _row_  = raw_mat[j];
         _coef_ = raw_comp[j]/100;
         output;

         _col_  = raw_mat[j];  _coef_ = -1;
         output;

         /* the right-hand-side */

         if i=1 then do;
            _col_='_RHS_';
            _coef_=0;
            output;
         end;
         end;
         _type_='  ';
      end;
      stop;
   run;
```

The model is solved using PROC LP, which saves the solution in the PRIMALOUT data set named solution. PROC PRINT displays the solution, shown in Figure 1.21.

```
   proc lp sparsedata primalout=solution;

   proc print ;
      id _var_;
      var _lbound_--_r_cost_;
   run;
```

```
_VAR_                  _LBOUND_     _VALUE_      _UBOUND_    _PRICE_    _R_COST_

all_polyester             10        11.800          14.0       3.55       0.000
all_silk                   6         7.000           7.0       6.70       6.490
cotton_material            0        13.600          13.6      -0.90       4.170
cotton_poly_blend          6         8.500           8.5       4.81       0.196
polyester_material         0        22.000          22.0      -0.60       2.950
poly_cotton_blend         13        15.300          16.0       4.31       0.000
silk_material              0         7.000          25.8      -0.21       0.000
PHASE_1_OBJECTIVE          0         0.000           0.0       0.00       0.000
profit                     0       168.708    1.7977E308       0.00       0.000
```

Figure 1.21. Solution Data Set

The solution shows that 11.8 units of polyester ties, 7 units of silk ties, 8.5 units of the cotton-polyester blend, and 15.3 units of the polyester-cotton blend should be produced. It also shows the amounts of raw materials that go into this product mix to generate a total profit of 168.708.

Exploiting Model Structure

Another example helps to illustrate how the model can be simplified by exploiting the structure in the model when using the NETFLOW procedure.

Recall the chocolate transshipment problem discussed previously. The solution required no production at factory_1 and no storage at warehouse_2. Suppose this solution, although optimal, is unacceptable. An additional constraint requiring the production at the two factories to be balanced is required. Now, the production at the two factories can differ by, at most, 100 units. Such a constraint might look like

```
   -100 <= (factory_1_warehouse_1 + factory_1_warehouse_2 -
            factory_2_warehouse_1 - factory_2_warehouse_2) <= 100
```

The network and supply and demand information are saved in two data sets.

```
data network;
   format from $12. to $12.;
   input  from $ to $ cost ;
   datalines;
factory_1    warehouse_1  10
factory_2    warehouse_1   5
factory_1    warehouse_2   7
factory_2    warehouse_2   9
warehouse_1  customer_1   3
warehouse_1  customer_2   4
warehouse_1  customer_3   4
warehouse_2  customer_1   5
warehouse_2  customer_2   5
warehouse_2  customer_3   6
;

data nodes;
   format node $12. ;
   input node $  supdem;
   datalines;
customer_1   -100
customer_2   -200
customer_3    -50
factory_1     500
factory_2     500
;
```

The factory-balancing constraint is not a part of the network. It is represented in the sparse format in a data set for side constraints.

```
data side_con;
   format _type_ $8. _row_ $8. _col_ $21. ;
   input  _type_  _row_  _col_  _coef_ ;
   datalines;
eq       balance   .                          .
.        balance  factory_1_warehouse_1      1
.        balance  factory_1_warehouse_2      1
.        balance  factory_2_warehouse_1     -1
.        balance  factory_2_warehouse_2     -1
.        balance  diff                      -1
lo       lowerbd  diff                    -100
up       upperbd  diff                     100
;
```

This data set contains an equality constraint that sets the value of DIFF to be the amount that factory 1 production exceeds factory 2 production. It also contains implicit bounds on the DIFF variable. Note that the DIFF variable is a nonarc variable.

```
proc netflow
   conout=con_sav
   arcdata=network nodedata=nodes condata=side_con
   sparsecondata ;
   node node;
   supdem supdem;
   tail from;
   head to;
   cost cost;
   run;

proc print;
   var from to _name_ cost _capac_ _lo_ _supply_ _demand_
       _flow_ _fcost_ _rcost_;
   sum _fcost_;
   run;
```

The solution is saved in the con_sav data set (Figure 1.22).

```
                                                            _   _
                                                    _       S   D          _  _
                                      _             C       U   E     _    F  R
                                      N             A       P   M     F    C  C
   f                                  A    c        P    _  P   A     L    O  O
 O r                                  M    o        A    L  L   N     O    S  S
 b o                t                 E    s        C    O  Y   D     W    T  T
 s m                o                 _    t        _    _  _   _     _    _  _

 1 warehouse_1      customer_1             3 99999999    0    . 100  100  300  .
 2 warehouse_2      customer_1             5 99999999    0    . 100    0    0 1.0
 3 warehouse_1      customer_2             4 99999999    0    . 200   75  300  .
 4 warehouse_2      customer_2             5 99999999    0    . 200  125  625  .
 5 warehouse_1      customer_3             4 99999999    0    .  50   50  200  .
 6 warehouse_2      customer_3             6 99999999    0    .  50    0    0 1.0
 7 factory_1        warehouse_1           10 99999999    0  500    .    0    0 2.0
 8 factory_2        warehouse_1            5 99999999    0  500    .  225 1125  .
 9 factory_1        warehouse_2            7 99999999    0  500    .  125  875  .
10 factory_2        warehouse_2            9 99999999    0  500    .    0    0 5.0
11                                   diff  0      100 -100    .    . -100    0 1.5
                                                                        ====
                                                                        3425
```

Figure 1.22. CON_SAV Data Set

Notice that the solution now has production balanced across the factories; the production at factory 2 exceeds that at factory 1 by 100 units.

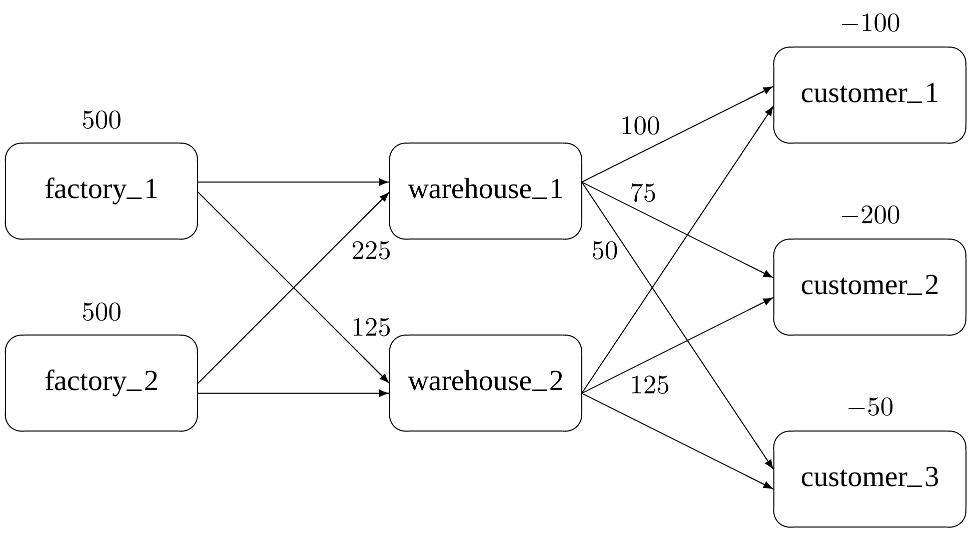


Figure 1.23. Constrained Optimum for the Transshipment Problem

Report Writing

The reporting of the solution is also an important aspect of modeling. Since the optimization procedures save the solution in one or more SAS data sets, report writing can be written using any of the tools in the SAS language.

The DATA Step

Use of the DATA step and PROC PRINT is the most general way to produce reports. For example, a table showing the revenue generated from the production and a table of the cost of material can be produced with the following program.

```
data product(keep= _var_ _value_ _price_ revenue)
     material(keep=_var_ _value_ _price_ cost);
   set solution;
   if _price_>0 then do;
      revenue=_price_*_value_; output product;
   end;
   else if _price_<0 then do;
      _price_=-_price_;
      cost = _price_*_value_; output material;
   end;
run;
```

```
/* display the product report */

proc print data=product;
   id _var_;
   var _value_ _price_ revenue ;
   sum revenue;
   title 'Revenue Generated from Tie Sales';
run;

/* display the materials report */

proc print data=material;
   id _var_;
   var _value_ _price_ cost;
   sum cost;
   title 'Cost of Raw Materials';
run;
```

This DATA step reads the solution data set saved by PROC LP and segregates the records based on whether they correspond to materials or products, namely whether the contribution to profit is positive or negative. Each of these is then displayed to produce Figure 1.24.

```
                    Revenue Generated from Tie Sales

            _VAR_                _VALUE_    _PRICE_    revenue

            all_polyester          11.8       3.55      41.890
            all_silk                7.0       6.70      46.900
            cotton_poly_blend       8.5       4.81      40.885
            poly_cotton_blend      15.3       4.31      65.943
                                                       =======
                                                       195.618
```

```
                         Cost of Raw Materials

                  _VAR_            _VALUE_    _PRICE_     cost

            cotton_material          13.6       0.90     12.24
            polyester_material       22.0       0.60     13.20
            silk_material             7.0       0.21      1.47
                                                         =====
                                                         26.91
```

Figure 1.24. Tie Problem: Revenues and Costs

Other Reporting Procedures

The GCHART procedure can be a useful tool for displaying the solution to mathematical programming models. The con_solv data set that contains the solution to the balanced transshipment problem can be effectively displayed using PROC GCHART. In Figure 1.25, the amount that is shipped from each factory and warehouse can be seen by submitting the following.

```
title;
proc gchart data=con_sav;
   hbar from / sumvar=_flow_;
run;
```

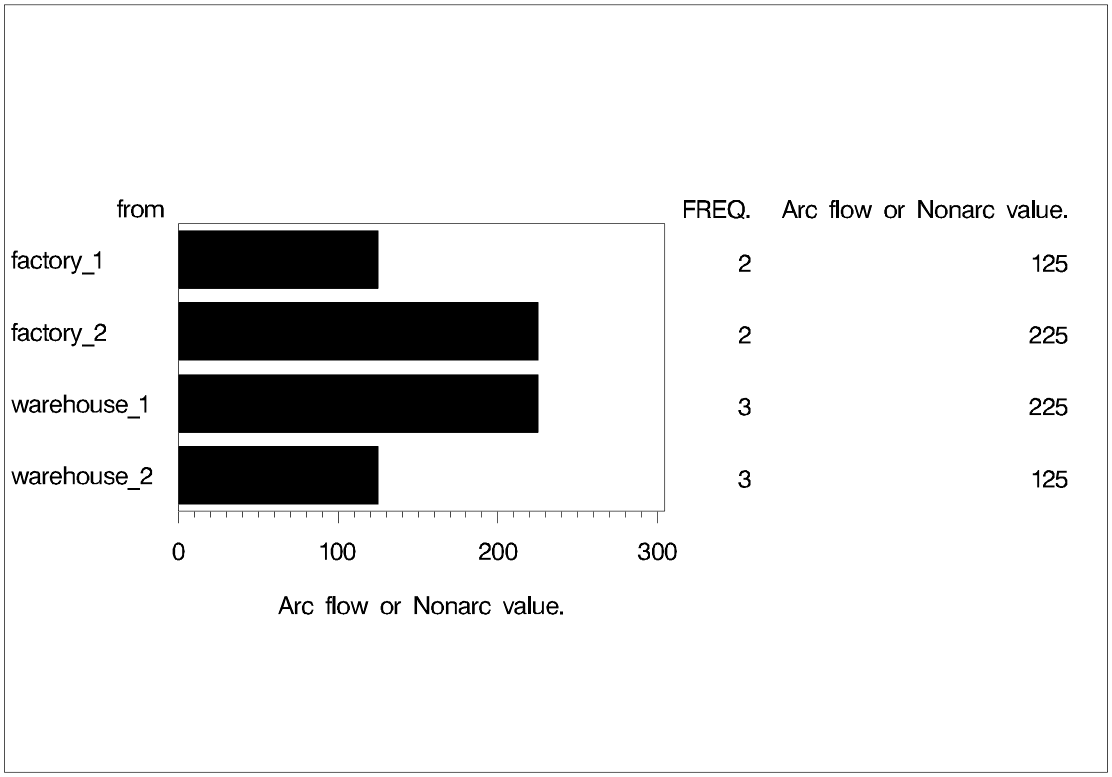


Figure 1.25. Tie Problem: Throughputs

The horizontal bar chart is just one way of displaying the solution to a mathematical program. The solution to the Tie Product Mix problem that was solved using PROC LP can also be illustrated using PROC GCHART. Here, a pie chart shows the relative contribution of each product to total revenues.

```
proc gchart data=product;
   pie _var_ / sumvar=revenue;
title 'Projected Tie Sales Revenue';
run;
```

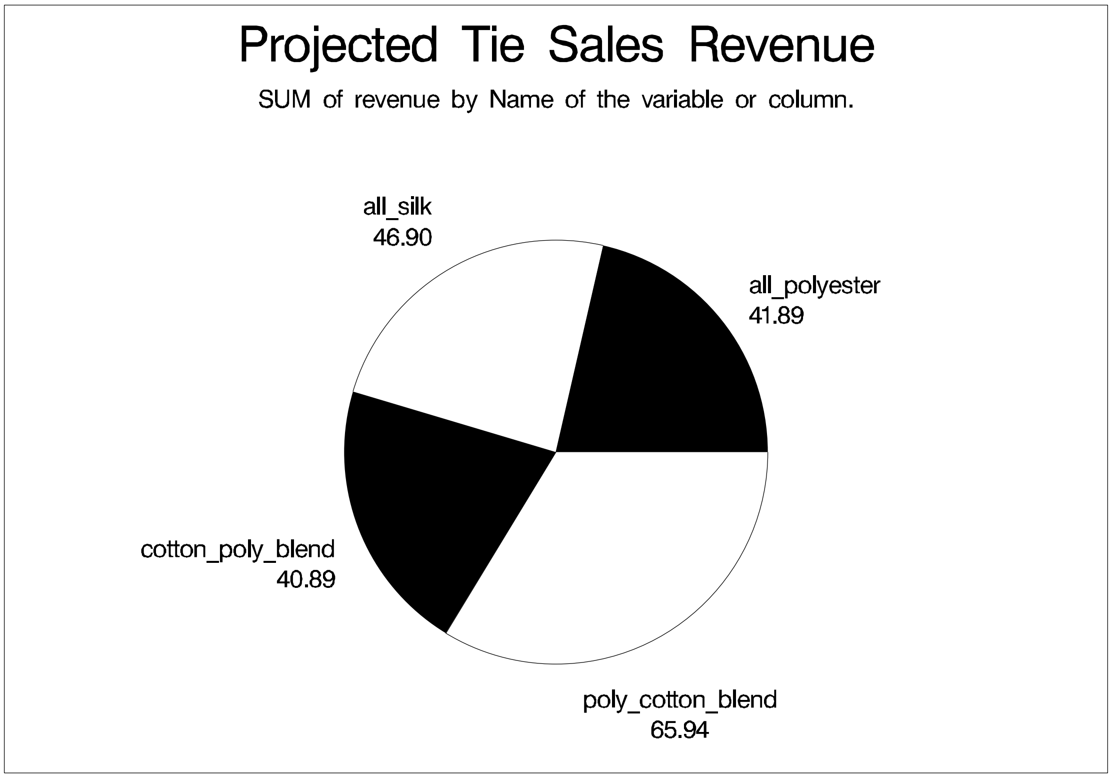


Figure 1.26. Tie Problem: Projected Tie Sales Revenue

The TABULATE procedure is another procedure that can help automate solution reporting. Several examples in Chapter 4, "The LP Procedure," illustrate its use.

Decision Support Systems

The close relationship between a SAS data set and the representation of the mathematical model makes it easy to build decision support systems.

The Full-Screen Interface

The ability to manipulate data using the full-screen tools in the SAS language further enhances the decision support capabilities. The several data set pieces that are components of a decision support model can be edited using the full-screen editing procedures FSEDIT and FSPRINT. The screen control language SCL directs data editing, model building, and solution reporting through its menuing capabilities.

The compatibility of each of these pieces in the SAS System makes construction of a full-screen decision support system based on mathematical programming an easy task.

Communicating with the Optimization Procedures

The optimization procedures communicate with any decision support system through the various problem and solution data sets. However, there is a need for the system to have a more intimate knowledge of the status of the optimization procedures. This is achieved through the use of macro variables defined by each of the optimization procedures.

References

Rosenbrock, H. H. (1960), “An Automatic Method for Finding the Greatest or Least Value of a Function,” *Computer Journal*, 3, 175–184.