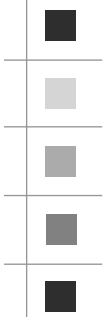
SAS Publishing

TECHNICAL REPORT

SAS/OR® Software: Changes and Enhancements, Release 8.2



The Power to Know™

58301

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/OR® Software: Changes and Enhancements, Release 8.2*, Cary, NC: SAS Institute Inc., 2001

ISBN 1-58025-851-4

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, January 2001

Table of Contents

Chapter 1
The BOM Procedure

Chapter Table of Contents

Chapter 1
The BOM Procedure

Overview

The new BOM procedure performs bill-of-material processing. It composes a series of single-level bills of material into a multilevel, tree-structured bill of material; determines the level of each part in the bill; and represents the multilevel bill of material structure in the form of an indented bill of material. PROC BOM can also output a summarized bill of material.

A *bill of material* (BOM) is a list of all parts, ingredients, or materials needed to make one production run of a product. The bill of material may also be called the *formula*, *recipe*, or *ingredients list* in certain process industries (Cox and Blackstone 1998). The way in which the bill of material data are organized and presented is called the *structure* of the bill of material or the structure of the product.

A variety of display formats are available for bills of material. The simplest format is the *single-level BOM*. It consists of a list of all components that are directly used in a parent item. A *multilevel bill of material* provides a display of all components that are directly or indirectly used in a parent item. When an item is a subcomponent, blend, intermediate, etc., then all of its components, including purchased parts and raw materials, are also exhibited. A multilevel structure can be illustrated by a tree with several levels. An *indented bill of material* is a form of multilevel BOM. It exhibits the final product as level 0 and all its components as level 1. The level numbers increase as you look down the tree structure. If an item is used in more than one parent within a given product structure, it appears more than once, under every subassembly in which it is used. The *summarized bill of material* is another form of a multilevel bill of material. It lists all the parts and their quantities required in a given product structure. Unlike the indented bill of material, it does not list the levels of parts and does not illustrate the part-component relationships. Moreover, the summarized bill of material lists each item only once for the total quantity required. Refer to Cox and Blackstone (1998) for further details.

The indented bill of material data generated by PROC BOM are organized in such a manner that they can be easily retrieved and manipulated to generate reports, and can also be used by other SAS/OR procedures to perform additional analysis. The summarized bill of material data are quite useful in gross requirements planning and other applications.

Getting Started

The example of the ABC Lamp Company product structure example from Fogarty, Blackstone, and Hoffmann (1991) illustrates the basic features of PROC BOM. The

single-level bill of material data are depicted in Figure 1.1. These consists of a list of each part in the product structure, the components that are directly used in the part, and the quantity of each component needed to make one unit of that particular part. The data set also includes a short description and the unit of measure for each part. The SAS DATA step code to create and display the single-level BOM data set is as follows:

```
   /* single-level BOM data   */
data SlBOM0;
   input Part      $8.
         Desc      $24.
         Unit      $8.
         Component $8.
         QtyPer    8.0
         ;
datalines;
LA01    Lamp LA                 Each    B100          1
                                        S100          1
                                        A100          1
B100    Base assembly           Each    1100          1
                                        1200          1
                                        1300          1
                                        1400          4
S100    Black shade             Each                  .
A100    Socket assembly         Each    1500          1
                                        1600          1
                                        1700          1
1100    Finished shaft          Each    2100         26
1200    7-Diameter steel plate  Each                  .
1300    Hub                     Each                  .
1400    1/4-20 Screw            Each                  .
1500    Steel holder            Each    1400          2
1600    One-way socket          Each                  .
1700    Wiring assembly         Each    2200         12
                                        2300          1
2100    3/8 Steel tubing        Inches                .
2200    16-Gauge lamp cord      Feet                  .
2300    Standard plug terminal  Each                  .
;

  /* Display the input data set */
proc print data=SlBOM0 noobs;
   title 'ABC Lamp Company';
   title3 'Single-Level Bill of Material';
run;
```

ABC Lamp Company

Single-Level Bill of Material

Part	Desc	Unit	Component	Qty Per
LA01	Lamp LA	Each	B100	1
			S100	1
			A100	1
B100	Base assembly	Each	1100	1
			1200	1
			1300	1
			1400	4
S100	Black shade	Each		.
A100	Socket assembly	Each	1500	1
			1600	1
			1700	1
1100	Finished shaft	Each	2100	26
1200	7-Diameter steel plate	Each		.
1300	Hub	Each		.
1400	1/4-20 Screw	Each		.
1500	Steel holder	Each	1400	2
1600	One-way socket	Each		.
1700	Wiring assembly	Each	2200	12
			2300	1
2100	3/8 Steel tubing	Inches		.
2200	16-Gauge lamp cord	Feet		.
2300	Standard plug terminal	Each		.

Figure 1.1. Single-Level Bill of Material

The following code invokes PROC BOM to produce the indented bill of material and the summarized bill of material. It also uses PROC NETDRAW to draw a *family tree* diagram for illustrating the multilevel product structure. For further details about PROC NETDRAW, refer to "The NETDRAW Procedure" chapter in the *SAS/OR User's Guide: Project Management*.

```
  /* Create the indented BOM and the summarized BOM */
proc bom data=SlBOM0 out=IndBOM0 summaryout=SumBOM0;
   structure / part=Part
               component=Component
               quantity=QtyPer
               id=(Desc Unit);
run;

  /* Draw a tree diagram for illustrating the product structure */
  /* Each record denotes a node in the tree */
data IndBOM0a(drop=Part_ID);
   set IndBOM0;
   Paren_ID=Part_ID;
run;

  /* Extract the Parent - Part information */
data IndBOM0b;
   set IndBOM0(keep=Paren_ID Part_ID);
run;
```

```
  /* Prepare the data set for running NETDRAW */
data TreBOM0;
   set IndBOM0a IndBOM0b;
run;

  /* Specify graphics options */
goptions hpos=32 vpos=80 border;
pattern1 v=s c=white;

title h=5 j=c 'Multilevel Bill of Material';
footnote h=2 j=l
   'Node shows ID Number, Part Number, and Quantity Required';

  /* Invoke PROC NETDRAW to display BOM tree */
proc netdraw data=TreBOM0( where=(Paren_ID NE .) );
   actnet / act=Paren_ID succ=Part_ID id=(Paren_ID _Part_ QtyPer)
            ctext=black font=swiss htext=3 cars=black
            ybetween=3 xbetween=8 centerid
            tree pcompress rotatetext rotate
            arrowhead=0 rectilinear nodefid nolabel;
run;
```

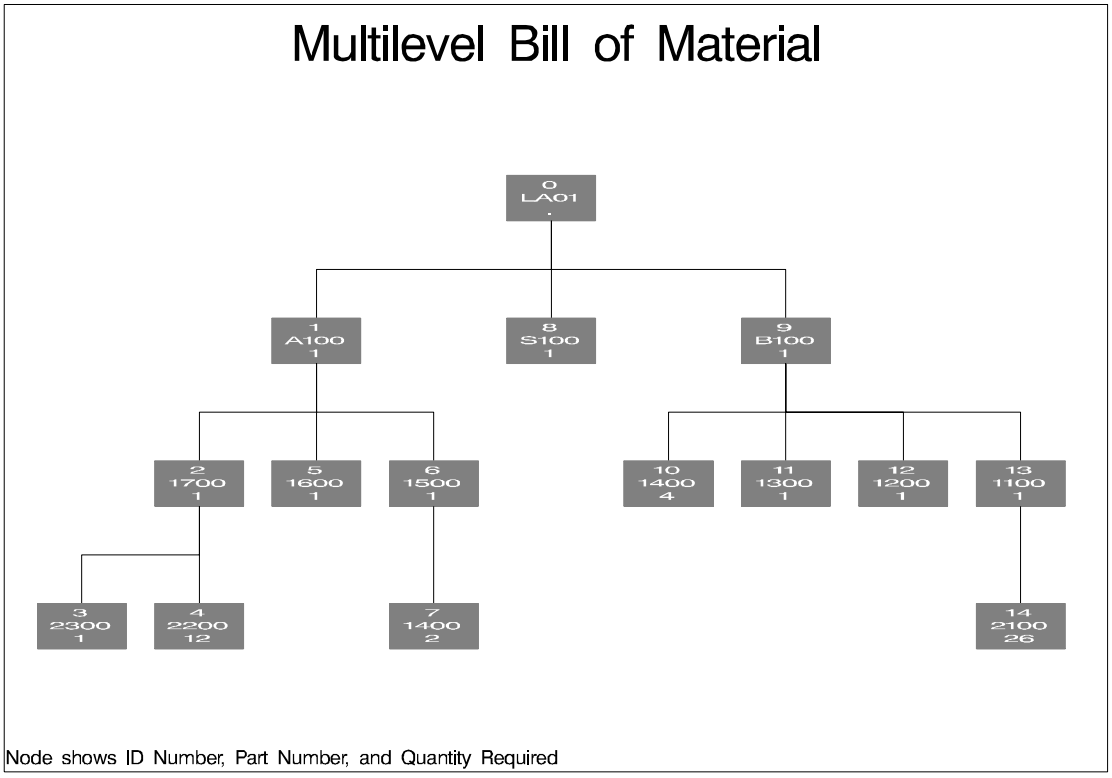


Figure 1.2. Multilevel Product Structure

Figure 1.2 displays the tree structure of the ABC Lamp Company. Corresponding to this tree diagram is the list of indented bill of material for the company as in Figure 1.3. The following code displays the indented BOM:

```
   /* Display the indented BOM data */
proc print data=IndBOM0 noobs;
   var _Level_ _Part_ Part_ID Desc QtyPer Qty_Prod
       Unit _Parent_ Paren_ID _Prod_;
   title 'ABC Lamp Company';
   title3 'Indented Bill of Material, Part LA01';
run;
```

ABC Lamp Company

Indented Bill of Material, Part LA01

Level	_Part_	Part_ID	Desc	Qty_Prod	QtyPer	Unit	_Parent_	Paren_ID	_Prod_
0	LA01	0	Lamp LA	.	1	Each		.	LA01
1	A100	1	Socket assembly	1	1	Each	LA01	0	LA01
2	1700	2	Wiring assembly	1	1	Each	A100	1	LA01
3	2300	3	Standard plug terminal	1	1	Each	1700	2	LA01
3	2200	4	16-Gauge lamp cord	12	12	Feet	1700	2	LA01
2	1600	5	One-way socket	1	1	Each	A100	1	LA01
2	1500	6	Steel holder	1	1	Each	A100	1	LA01
3	1400	7	1/4-20 Screw	2	2	Each	1500	6	LA01
1	S100	8	Black shade	1	1	Each	LA01	0	LA01
1	B100	9	Base assembly	1	1	Each	LA01	0	LA01
2	1400	10	1/4-20 Screw	4	4	Each	B100	9	LA01
2	1300	11	Hub	1	1	Each	B100	9	LA01
2	1200	12	7-Diameter steel plate	1	1	Each	B100	9	LA01
2	1100	13	Finished shaft	1	1	Each	B100	9	LA01
3	2100	14	3/8 Steel tubing	26	26	Inches	1100	13	LA01

Figure 1.3. Indented Bill of Material

Each *record* in the indented BOM data set is associated with a *node* in the tree structure, and the node or record is uniquely identified by a *sequence number* that is assigned to it by the procedure. The Part_ID variable denotes this sequence number of the node or record. The Part_ID information is needed in case some items are required at more than one place in the BOM. For example, the item `'1400'` (`1/4-20 Screw`) is required at `'B100'` (`Base assembly`) and `'1500'` (`Steel holder`). In this situation the part number is not sufficient to identify each node in the tree structure. Thus, the two different nodes are uniquely identified by the values `'7'` and `'10'` for the variable Part_ID.

The _Level_ variable denotes the level number of each node. The final product, `'LA01'`, is at level 0, and the level numbers increase as you look down the tree. The Part, QtyPer, and ID variables (Desc and Unit) in the single-level BOM data set, SlBOM0, are also included in the indented BOM data set. Note that the name of the Part variable is changed to _Part_ in the indented BOM data set.

There are two variables in the indented BOM data set identifying parent information: the _Parent_ variable denotes the *part number* for the parent item (next assembly)

of the part identified by the _Part_ variable, while the Paren_ID variable specifies the *sequence number* for the parent node of the current node (record) identified by the Part_ID variable. For example, the indented BOM data set, as shown in Figure 1.3, reveals that the part '1400' has two parent items: part '1500' and part 'B100'. Meanwhile, the parent node of node '7' is node '6', and the parent node of node '10' is node '9'.

Finally, the _Prod_ variable denotes the part number of the final product in the production structure. The Qty_Prod variable contains the required quantity of the item identified by the _Part_ variable in order to make one unit of the final product identified by the _Prod_ variable. Note that in this particular example, the values of the Qty_Prod variable are identical to the values of the QtyPer variable. This is because the values of the QtyPer variable for all parent items are 1.

Figure 1.2 and Figure 1.3 show that the Indented BOM data set lists all of the parts of the tree structure in *depth-first* order (Aho, Hopcroft, and Ullman 1983). Unlike the Single-level BOM data set, the Indented BOM data set provides the "part-parent" information rather than the "part-component" relationship. The part-component information can still be easily retrieved. For example, the following code creates a list of components that are directly used in the part 'LA01':

```
  /* Display the components that are directly used */
  /* in item LA01                                  */
proc print data=IndBOM0(where=(_Parent_='LA01')
                        rename=(_Part_=Component))
           noobs;
   var Component Desc QtyPer Unit _Prod_;
   title 'ABC Lamp Company';
   title3 'Single-Level Bill of Material Retrieval, Part LA01';
run;
```

ABC Lamp Company

Single-Level Bill of Material Retrieval, Part LA01

Component	Desc	Qty Per	Unit	_Prod_
A100	Socket assembly	1	Each	LA01
S100	Black shade	1	Each	LA01
B100	Base assembly	1	Each	LA01

Figure 1.4. Components That Are Directly Used in Part LA01

You can also create *single-level where-used* reports from the Indented BOM data set. For example, the single-level where-used data set Used0b, displayed in Figure 1.5, lists all parent items that directly use the part '1400'. The SAS code that creates the data set Used0b from the Indented BOM data set IndBOM0 is as follows:

```
  /* Create the where-used data set */
data Used0a(keep=_Parent_ Paren_ID QtyPer Unit _Prod_);
   set IndBOM0(where=(_Part_='1400'));
run;

  /* Get the part description from the IndBOM0 data set */
proc sql;
   create table Used0b as
      select Used0a._Parent_, IndBOM0.Desc,
             Used0a.QtyPer, Used0a.Unit, Used0a._Prod_
         from Used0a left join IndBOM0
            on Used0a.Paren_ID=IndBOM0.Part_ID;
quit;

  /* Display the where-used data set */
proc print data=Used0b noobs;
   var _Parent_ Desc QtyPer Unit _Prod_;
   title 'ABC Lamp Company';
   title3 'Single-Level Where-used Report, Part 1400';
run;
```

```
                    ABC Lamp Company

        Single-Level Where-used Report, Part 1400

                                  Qty
    _Parent_        Desc          Per    Unit    _Prod_

      1500      Steel holder      2     Each     LA01
      B100      Base assembly     4     Each     LA01
```

Figure 1.5. Parents in Which the Part 1400 Is Directly Used

The following SAS code sorts and displays the Summarized BOM data set that is produced by the BOM procedure:

```
   /* Sort and display the summarized BOM data */
proc sort data=SumBOM0;
   by _Part_;
run;

proc print data=SumBOM0 noobs;
   title 'ABC Lamp Company';
   title3 'Summarized Bill of Material, Part LA01';
run;
```

ABC Lamp Company

Summarized Bill of Material, Part LA01

Part	Low_Code	Gros_Req	On_Hand	Net_Req	Desc	Unit
1100	2	1	0	1	Finished shaft	Each
1200	2	1	0	1	7-Diameter steel plate	Each
1300	2	1	0	1	Hub	Each
1400	3	6	0	6	1/4-20 Screw	Each
1500	2	1	0	1	Steel holder	Each
1600	2	1	0	1	One-way socket	Each
1700	2	1	0	1	Wiring assembly	Each
2100	3	26	0	26	3/8 Steel tubing	Inches
2200	3	12	0	12	16-Gauge lamp cord	Feet
2300	3	1	0	1	Standard plug terminal	Each
A100	1	1	0	1	Socket assembly	Each
B100	1	1	0	1	Base assembly	Each
LA01	0	1	0	1	Lamp LA	Each
S100	1	1	0	1	Black shade	Each

Figure 1.6. Summarized Bill of Material

The Summarized BOM data set, as displayed in Figure 1.6, lists all the parts and their quantity required in the given ABC Lamp Company product structure. Since the Single-level BOM data set, SlBOM0, does not contain any gross requirement information, PROC BOM assumes that the final product, `'LA01'`, is the only master schedule item. The *master schedule items* (MSI) are items selected to be planned by the master scheduler. These items are deemed critical in their impact on lower-level components or resources. Therefore, the master scheduler, not the computer, maintains the plan for these items. The BOM procedure also assumes that the gross requirement of `'LA01'` is 1 unit and that there are no items currently on hand. The net requirement of the part `'LA01'` is the gross requirement (1) minus the quantity on hand (0). The gross requirements of the other items are then calculated sequentially, level by level, as follows:

'B100' Base assembly (1 per lamp)

Gross requirement	$1 \times 1 = 1$
Quantity on hand	-0
Net requirement	1

'1100' Finished shaft (1 per base assembly)

Gross requirement	$1 \times 1 = 1$
Quantity on hand	-0
Net requirement	1

'2100' 3/8 Steel tubing (26 inches per shaft)

Gross requirement	$1 \times 26 = 26$
Quantity on hand	-0
Net requirement	26

If a part (like item `'1400'`) is required at more than one assembly, the gross requirement is the total needed in all assemblies where it is required.

The Net_Req variable denotes the net requirement, and the Low_Code variable denotes the *low-level code* of the part. The low-level code is a number that indicates

the lowest level in any bill of material at which the item appears. The low-level codes are necessary to make sure that the net requirement for a given item is not calculated until all the gross requirements have been calculated down to that level.

Syntax

The following statements are available in PROC BOM.

PROC BOM *options* **;**
STRUCTURE */ options* **;**

Functional Summary

The following table outlines the options available for the BOM procedure.

Table 1.1. PROC BOM Options

Task	Statement	Option
Data Set Specification		
Single-level BOM data set	BOM	DATA=
Indented BOM data set	BOM	OUT=
Summarized BOM data set	BOM	SUMMARYOUT=
Output Control Options		
ID variables	STRUCTURE	ID=
Part Information Specification		
lead time variable	STRUCTURE	LEADTIME=
quantity on hand variable	STRUCTURE	QTYONHAND=
gross requirement variable	STRUCTURE	REQUIREMENT=
Problem Size Options		
number of distinct part names	BOM	NNAMES=
do not use utility data set	BOM	NOUTIL
number of items	BOM	NPARTS=
Product Structure Specification		
component variables	STRUCTURE	COMPONENT=
end items	STRUCTURE	ENDITEM=
part variable	STRUCTURE	PART=
quantity of component needed to make one part item	STRUCTURE	QUANTITY=

PROC BOM Statement

PROC BOM *options* **;**

The PROC BOM statement invokes the procedure. You can specify the following options in the PROC BOM statement.

DATA=*SAS-data-set*
names the input SAS data set. This data set (also referred to as the Single-level BOM data set) contains all of the single-level bill of material information. See the "Single-Level Bills of Material Data Set" section on page 15 for information about the variables that can be included in this data set. If you do not specify the DATA= option, PROC BOM uses the most recently created SAS data set as the input data set.

NNAMES=*tsize*
specifies the size of the symbolic table used to look up the part names/numbers for which memory is initially allocated in core by the procedure. This can help PROC BOM to make better use of available memory. See the "Computer Resource Requirements" section on page 21 for details. If the number of part names/numbers exceeds *tsize*, the procedure uses a utility data set for storing the tree used for the table lookup. The default value for *tsize* is set to $nobs \times (ncomp+1)$, where *nobs* is the number of observations in the Single-level BOM data set and *ncomp* is the number of Component variables.

NOUTIL
specifies that if the number of items in the problem exceeds the number of items for which memory is allocated in core, the procedure will try to expend the amount of allocated memory so that the whole problem can fit in. If you do not specify this option, the procedure resorts to the use of utility data sets and swaps between core memory and utility data sets as necessary if the number of items in the problem is larger than the number of items for which memory is allocated in core. On the other hand, if you specified this option and the problem is too large to fit in core memory, PROC BOM will stop with an error message.

NPARTS=*nparts*
specifies the number of items for which memory is initially allocated in core by the procedure. This can help PROC BOM to make better use of available memory. See the "Computer Resource Requirements" section on page 21 for details. If the number of items exceeds *nparts*, the procedure uses a utility data set for storing the item information array. The default value for *nparts* is set to $nobs \times (ncomp+1)$, where *nobs* is the number of observations in the Single-level BOM data set and *ncomp* is the number of Component variables.

OUT=*SAS-data-set*

specifies a name for the output SAS data set that contains all of the indented bill of material information. This data set is also referred to as the Indented BOM data set. See the "Indented Bills of Material Data Set" section on page 17 for information about the variables that are included in this data set. If the OUT= option is omitted, the SAS System creates a data set and names it according to the DATA*n* naming convention.

SUMMARYOUT=*SAS-data-set*

specifies a name for the output SAS data set that contains all of the summarized bill of material information. This data set is also referred to as the Summarized BOM data set. See the "Summarized Bills of Material Data Set" section on page 18 for information about the variables that are included in this data set. If the SUMMARYOUT= option is omitted, PROC BOM does not produce the summarized bill of material.

STRUCTURE Statement

STRUCTURE / *options* **;**

The STRUCTURE statement lists the variables in the input data set (the Single-level BOM data set). You should specify the Part variable and at least one Component variable. All other variables are optional. In this statement, you can also explicitly specify the parts that are to be used as the level-0 items.

COMPONENT=(*variables*)
COMP=(*variables*)

specifies variables in the input data set that name all of the components directly used in the item identified by the Part variable. These variables are also referred to as Component variables. The Component variables must be of the same type, format, and length as the Part variable. At least one Component variable must be specified. **NOTE:** This variable is case-sensitive when it is in character format.

ENDITEM=(*enditems*)

specifies the parts that are to be used as the highest level or level-0 items in the indented bills of material. In other words, this option can be used to construct indented bills of material for subassemblies. The values for *enditems* must be either numbers (if the Part variable is numeric) or quoted strings (if the Part variable is character). If you do not specify this option, the procedure uses all final products as end items and builds an indented BOM for each of these items.

ID=(*variables*)

identifies all variables not specified in the COMPONENT=, LEADTIME=, PART=, QTYONHAND=, QUANTITY=, or REQUIREMENT= options that are to be included in the Indented BOM and the Summarized BOM output data sets. These variables are also referred to as ID variables. The ID= option is useful for carrying any relevant information about each part (identified by the Part variable) from the Single-level BOM data set to the output data sets. The ID variables cannot be named as Gros_Req, _Level_, Low_Code, Net_Req, On_Hand, Qty_Per, Qty_Prod, _Parent_, Paren_ID, _Part_, Part_ID, _Prod_, or Tot_Lead.

LEADTIME=*variable*
DUR=variable
identifies the variable in the input data set that contains the lead time information for the part identified by the Part variable. This variable is also referred to as the LeadTime variable. The *lead time* of a part is the length of time between recognition of the need for an order of this part and the receipt of the parts. The variable specified must be numeric. If you do not specify this option, PROC BOM assumes the lead time for each part to be 0. The information in the LeadTime variable is carried to the Indented BOM output data set. Moreover, the procedure creates a new variable, Tot_Lead, in the Indented BOM data set if this option is specified. See the "Indented Bills of Material Data Set" section on page 17 for more information about the Tot_Lead variable.

PART=*variable*
ITEM=variable
specifies the variable in the input data set that names the parent item. This variable is also referred to as the Part variable. The Part variable can be either character or numeric. You must specify the PART= option in the STRUCTURE statement. All the information specified by the ID variables corresponds to the part identified by the Part variable. All the information in the Part variable, except the name of the variable, is carried to the Indented BOM and the Summarized BOM output data sets. The name of this variable is changed to _Part_ in both output data sets.

NOTE: This variable is case-sensitive when it is in character format.

QTYONHAND=*variable*
INVENTORY=*variable*
identifies the variable in the input data set that contains the quantity currently on hand for the part identified by the Part variable. This variable is also referred to as the QtyOnHand variable. The QtyOnHand variable must be numeric. If you do not specify the QtyOnHand variable, the procedure assumes that you do not have any items on hand. In addition, the default name, On_Hand, is used for this variable. The information in the QtyOnHand variable is carried to the Summarized BOM output data set if the SUMMARYOUT= option is specified.

QUANTITY=(*variables*)
QTYPER=(*variables*)
identifies variables in the input data set that contain the quantity of items identified by the Component variables required to make 1 unit of the item identified by the Part variable. These variables are also referred to as the Quantity or QtyPer variables. The variables specified must be numeric and the number of Quantity variables must be equal to the number of Component variables. If you do not specify these variables, the procedure assumes that you need 1 unit of each listed component identified by a Component variable to make 1 unit of the parent item identified by the Part variable. If there is exactly one Quantity variable, the information in this variable is carried to the Indented BOM output data set. Otherwise, the output data set uses the default name Qty_Per.

REQUIREMENT=*variable*
GROSSREQ=*variable*
identifies the variable in the input data set that contains the gross requirement of the part identified by the Part variable. This variable is also referred to as the Requirement variable and must be numeric. If you do not specify this option, the procedure assumes that the gross requirement for all parts are missing except for the final products. All final products are assumed to have gross requirements of 1 unit. In addition, the default name, Gros_Req, is used for this variable. The information in this variable is carried to the Summarized BOM output data set if the SUMMARYOUT= option is specified.

Details

Single-Level Bills of Material Data Set

The BOM procedure uses the Single-level BOM data set as input data with key variable names being used to identify the appropriate information. Table 1.2 lists all of the variables in this input data set, with their type and their interpretation by the BOM procedure. It also lists the options in the STRUCTURE statement that are used to identify those variables.

Table 1.2. Single-Level BOM Data Set and Associated Variables

Variable	Type	Option	Interpretation
Component	same as Part	COMPONENT=	Component name or number
ID	character or numeric	ID=	Additional information about the part
LeadTime	numeric	LEADTIME=	Lead time of the part
Part	character or numeric	PART=	Part name or number
QtyOnHand	numeric	QTYONHAND=	Quantity of the part that is currently on hand
Quantity	numeric	QUANTITY=	Quantity of the component required per unit of part
Requirement	numeric	REQUIREMENT=	Gross requirement of the part

The value of the Requirement variable should be missing for most parts, except master schedule items. A *master schedule item* (MSI) is a part that is selected to be planned by the master scheduler. In general, final products are master schedule items. Some companies like to select a few items that are deemed critical in their impact on lower level components or resources as master schedule items. The requirements of the master schedule items are also called *independent demands.* In the Single-level BOM data set, if the value of the Requirement variable is greater than or equal to

0, then the procedure treats the item identified by the Part variable as an MSI and assumes that its gross requirement is planned by the master scheduler. The gross requirements of other parts (also known as the *dependent demands*) are determined by the procedure as the value of the net requirements of their parent items times the quantity required to make one unit of the parent items. This process is called the *dependent demand process* (Clement, Coldrick, and Sari 1992).

In general, the information in the Part and the LeadTime variables and all ID variables are stored in the company's *part master data file*. The part master data file gives various details of fairly constant data associated with the individual parts. It might contain the following kinds of data (Clement, Coldrick, and Sari 1992):

- part number or name
- part description
- engineering drawing number
- revision level
- commodity code
- cost
- product code
- lead time
- order quantity
- item type or status
- latest engineering change dates
- lot size
- planner/buyer codes
- unit of measure
- make or buy

The information for the Part, Component, and Quantity variables can usually be found in the *product structure data*. The product structure data describe the material content of a product at each stocking level in the manufacturing process. The quantity of each part that is currently on hand is contained in the company's inventory data, and the gross requirements of master schedule items come from the *master production schedule* (MPS). The Single-level BOM data set contains information from all these typical data sets. To obtain the Single-level BOM data set, you can combine the part master data file, inventory data file, and master production schedule, based on the common variable Part, to create a new data set. Then you can either append the product structure data (part-component relationship information) to the end of the new data set (as in Example 1.1 on page 21) or merge them together (as in Example 1.2 on page 27).

Indented Bills of Material Data Set

The Indented BOM data set produced by PROC BOM contains all the information in the Single-level BOM data set, plus a few more variables, to describe the structure of the products. The Part, LeadTime, Quantity, and all ID variables in the Single-level BOM data set are carried to this output data set. The name of the Part variable is changed to _Part_. Moreover, if there is more than one Quantity variable in the Single-level BOM data set, the procedure uses the name Qty_Per as the Quantity variable in the Indented BOM data set. Like the Single-level BOM data set, the information contained in those variables is associated with the item identified by the _Part_ variable. A few new variables are also added to the Indented BOM data set.

The _Level_ variable contains the indenture level of the item identified by the _Part_ variable. The top-most parts (the final products) have level 0, and all components that are directly used by these parts have level 1. All subsequent components of those items have the level number increased by 1. This process continues until there are no subsequent components. The _Parent_ variable contains the part name or number of the parent item. The _Prod_ variable contains the part name or number of the final product in the product structure. If the input data set contains the lead time information for each part, PROC BOM measures the total lead time accumulated from the final product (identified by the _Prod_ variable) to the part (identified by the _Part_ variable) and puts it in the Tot_Lead variable. The Qty_Prod variable denotes the quantity of the part (identified by the _Part_ variable) required to make one unit of the product identified by the _Prod_ variable.

Two more variables are added to the output data set: variable Part_ID and variable Paren_ID. Note that if an item is used in more than one parent item, it appears more than once, under every subassembly in which it is used. In order to distinguish those parts that have multiple appearances, PROC BOM assigns a unique number to each part in the multilevel tree structure. The Part_ID variable contains this identification number of the part (identified by the _Part_ variable), and the Paren_ID variable contains the identification number of the parent item. These two variables are useful when you use the Indented BOM data set as the input data set for other SAS/OR procedures, such as PROC NETDRAW and PROC CPM, which require unique identification of each node.

Table 1.3 lists all of the variables in the Indented BOM data set. It also lists the type and a brief description of these variables.

Table 1.3. Indented BOM Data Set and Associated Variables

Variable	Type	Interpretation
ID	character or numeric	Additional information about the part
LeadTime	numeric	Lead time of the part
Level	numeric	Level number of the part
Parent	same as _Part_	Parent item of the part
Paren_ID	numeric	Identification number of the parent

Table 1.3. (continued)

Variable	Type	Interpretation
Part	character or numeric	Part name or number
Part_ID	numeric	Identification number of the part
Prod	same as _Part_	Name or number of the final product
Qty_Prod	numeric	Quantity required to produce one unit of final product
Quantity	numeric	Quantity per assembly
Tot_Lead	numeric	Total lead time

The records in the Indented BOM data set are organized so that the final product item is always listed first. In addition, each item is listed directly after its parent and before any right siblings of the parent. For example, from the Indented BOM data as shown in Figure 1.3 on page 7, you can easily see that the final product `'LA01'` is the first record of the Indented BOM data set. Moreover, the part `'1700'` is listed directly after its parent, `'A100'`, and before the parts `'S100'` and `'B100'` (the right siblings of the part `'A100'`).

The single-level (part-component) and multilevel relationships of the product structure are clearly shown in the Indented BOM data. For example, the three components that go into the parent `'LA01'` are easily determined by the level 1 identifiers. It is also easy to see that the part `'B100'` (`Base assembly`) requires four components at level 2.

Summarized Bills of Material Data Set

If the SUMMARYOUT= option in the PROC BOM statement is specified, the BOM procedure creates a Summarized BOM data set. The summarized bill of material data set lists all the parts and their quantities required and currently on hand. Unlike the indented bill of material data set, it does not list the part-parent relationships and the levels of manufacture, and it lists a part only once for the total quantity used.

The Part, QtyOnHand, and all ID variables in the Single-level BOM data set are carried to this output data set. The name of the Part variable is changed to _Part_. If the item identified by the _Part_ variable is a master schedule item, the value of the Requirement variable (the independent demand) is also carried to this data set. The value of the Requirement variable (dependent demand) for those items that are not master schedule items is determined by the net requirement of the parent item times the quantity needed per parent item.

The new variable, Net_Req, denotes the net requirement of the item identified by the _Part_ variable. The value of the Net_Req variable is determined as

$$\text{Net\_Req} = \text{Requirement} - \text{QtyOnHand}$$

The Low_Code variable denotes the low-level code of the item identified by the _Part_ variable, that is, the lowest level in any bill of material at which the item appears. The low-level codes are necessary to make sure that the net requirement of any item is not calculated until all the gross requirements have been calculated down to that level.

Table 1.4 lists all of the variables in the Summarized BOM data set. It also lists the type and a brief description of these variables.

Table 1.4. Summarized BOM Data Set and Associated Variables

Variable	Type	Interpretation
ID	character or numeric	Additional information about the part
Low_Code	numeric	Low-level code of the part
Net_Req	numeric	Net requirement of the part
Part	character or numeric	Part name or number
QtyOnHand	numeric	Quantity of the part that is currently on hand
Requirement	numeric	Gross requirement of the part

Missing Values in the Input Data Set

The following table summarizes the treatment of missing values for variables in the Single-level BOM input data set.

Table 1.5. Treatment of Missing Values in the BOM Procedure

Variable	Value Used / Assumption Made / Action Taken
Component	value ignored
ID	missing, if Part is not missing; otherwise ignored
LeadTime	0, if Part is not missing; otherwise ignored
Part	input error: procedure stops with error message, if this is the first record; otherwise, treat this record as a continuation of the previous record

Table 1.5. (continued)

Variable	Value Used / Assumption Made / Action Taken
QtyOnHand	0, if Part is not missing; otherwise ignored
Quantity	1, if corresponding Component variable is not missing; otherwise ignored
Requirement	ignored if Part is missing; 1, if Part is not missing and the part identified by the Part variable is a final product; otherwise, the value is determined by the procedure

Note that a missing value is allowed for the Part variable only when the information for an item is specified in more than one record (see Figure 1.1 on page 5 as an example). If the Part variable is missing, the values for the ID, LeadTime, QtyOnHand, and Requirement variables are ignored by the procedure. If the Part variable is not missing and the item identified by the Part variable is not a master schedule item (that is, it is not a final product and the value of the corresponding Requirement variable is missing), the gross requirement of this part is determined based on the dependent demand process. See the "Summarized Bills of Material Data Set" section on page 18 for details about the dependent demand process.

Macro Variable _ORBOM_

The BOM procedure defines a macro variable named _ORBOM_. This variable contains a character string that indicates the status of the procedure. It is set at procedure termination. The form of the _ORBOM_ character string is STATUS= REASON=, where STATUS= is either SUCCESSFUL or ERROR_EXIT, and REASON= (if PROC BOM terminated unsuccessfully) can be one of the following:

CYCLE

BADDATA_ERROR

MEMORY_ERROR

IO_ERROR

SEMANTIC_ERROR

SYNTAX_ERROR

BOM_BUG

UNKNOWN_ERROR

This information can be used when PROC BOM is one step in a larger program that needs to determine whether the procedure terminated successfully or not. Because _ORBOM_ is a standard SAS macro variable, it can be used in the ways that all macro variables can be used.

Computer Resource Requirements

There is no inherent limit on the size of the product structure that can be handled with the BOM procedure. The number of parts and components are constrained only by the amount of memory available. Naturally, there needs to be a sufficient amount of core memory available in order to invoke and initialize the SAS system. As far as possible, the procedure attempts to store all the data in core memory.

However, if the problem is too large to fit in core memory, the procedure resorts to the use of utility data sets and swaps between core memory and utility data sets as necessary, unless the NOUTIL option is specified. The procedure uses the NPARTS= and NNAMES= options to determine approximate problem size. If these options are not specified, the procedure estimates default values on the basis of the number of observations in the Single-level BOM data set. See the "Syntax" section on page 11 for default specifications.

The storage requirement for the data area and the time required by the procedure are proportional to the number of items in the product structure.

Examples

This section contains examples that illustrate the features of the BOM procedure. Most of the examples use the data from the ABC Lamp Company product structure described in the "Getting Started" section beginning on page 3.

Example 1.1. Bill of Material with Lead Time Information

As explained in the section "Single-Level Bills of Material Data Set" beginning on page 15, most companies store the relatively constant information for parts in a part master file. In this example, you start with two different data files, Pmaster1 and ParComp1. The Pmaster1 data set, as shown in Output 1.1.1, is a simplified version of the ABC Lamp Company's part master file. Note that the data set contains the description, the lead time, and the unit of measure information for each part. The data set ParComp1 (as shown in Output 1.1.2) lists the part-component relationship of each part along with the quantity of the component item needed to make one unit of that particular part.

The SAS DATA step code to accomplish this is as follows:

```
   /* Part Master data */
data PMaster1;
   input Part      $8.
         Desc      $24.
         Unit      $8.
         LeadTime  8.0
         ;

datalines;
1100    Finished shaft          Each          2
1200    6-Diameter steel plate  Each          3
```

```
1300    Hub                    Each          2
1400    1/4-20 Screw           Each          1
1500    Steel holder           Each          2
1600    One-way socket         Each          2
1700    Wiring assembly        Each          1
2100    3/8 Steel tubing       Inches        3
2200    16-Gauge lamp cord     Feet          2
2300    Standard plug terminal Each          1
A100    Socket assembly        Each          1
B100    Base assembly          Each          1
LA01    Lamp LA                Each          2
S100    Black shade            Each          2
;


   /* Part-Component relationship data   */
data ParComp1;
   input Part      $8.
         Component $8.
         QtyPer    8.0 ;
datalines;
LA01    B100           1
        S100           1
        A100           1
B100    1100           1
        1200           1
        1300           1
        1400           4
A100    1500           1
        1600           1
        1700           1
1100    2100          26
1500    1400           2
1700    2200          12
        2300           1
;


  /* Display the part master data */
proc print data=PMaster1 noobs;
   title 'ABC Lamp Company';
   title3 'Part Master Data';
run;


  /* Display the part-component relationship data */
proc print data=ParComp1 noobs;
   title 'ABC Lamp Company';
   title3 'Part-Component Relationship Data';
run;
```

Output 1.1.1. Part Master Data Set

ABC Lamp Company

Part Master Data

Part	Desc	Unit	Lead Time
1100	Finished shaft	Each	2
1200	6-Diameter steel plate	Each	3
1300	Hub	Each	2
1400	1/4-20 Screw	Each	1
1500	Steel holder	Each	2
1600	One-way socket	Each	2
1700	Wiring assembly	Each	1
2100	3/8 Steel tubing	Inches	3
2200	16-Gauge lamp cord	Feet	2
2300	Standard plug terminal	Each	1
A100	Socket assembly	Each	1
B100	Base assembly	Each	1
LA01	Lamp LA	Each	2
S100	Black shade	Each	2

Output 1.1.2. Part-Component Relationship Data Set

ABC Lamp Company

Part-Component Relationship Data

Part	Component	Qty Per
LA01	B100	1
	S100	1
	A100	1
B100	1100	1
	1200	1
	1300	1
	1400	4
A100	1500	1
	1600	1
	1700	1
1100	2100	26
1500	1400	2
1700	2200	12
	2300	1

The SlBOM1 data set is created by appending the part-component data set, ParComp1, to the end of the data set PMaster1 to form an input data set for the procedure:

```
  /* Append the Part-Component information to */
  /* the part master data                     */
data SlBOM1;
   set PMaster1 ParComp1;
run;
```

The following code produces the Indented BOM data set and displays it in Output 1.1.3. A new variable, Tot_Lead, has been added to the output data set. This variable denotes the total lead time accumulated from the product 'LA01' to the item identified by the _Part_ variable:

```
  /* Create the indented BOM with lead time */
proc bom data=SlBOM1 out=IndBOM1;
   structure / part=Part
               leadtime=LeadTime
               component=Component
               quantity=QtyPer
               id=(Desc Unit);
run;

  /* Display the indented BOM data */
proc print data=IndBOM1 noobs;
   var _Level_ _Part_ Desc QtyPer Qty_Prod
       Unit LeadTime Tot_Lead _Parent_ _Prod_;
   title 'ABC Lamp Company';
   title3 'Indented Bill of Material, Part LA01';
run;
```

Output 1.1.3. Indented Bill of Material with Lead Time

ABC Lamp Company

Indented Bill of Material, Part LA01

Level	_Part_	Desc	Qty	Qty_Prod	Unit	LeadTime	Tot_Lead	_Parent_	_Prod_
0	LA01	Lamp LA	.	1	Each	2	2		LA01
1	A100	Socket assembly	1	1	Each	1	3	LA01	LA01
2	1700	Wiring assembly	1	1	Each	1	4	A100	LA01
3	2300	Standard plug terminal	1	1	Each	1	5	1700	LA01
3	2200	16-Gauge lamp cord	12	12	Feet	2	6	1700	LA01
2	1600	One-way socket	1	1	Each	2	5	A100	LA01
2	1500	Steel holder	1	1	Each	2	5	A100	LA01
3	1400	1/4-20 Screw	2	2	Each	1	6	1500	LA01
1	S100	Black shade	1	1	Each	2	4	LA01	LA01
1	B100	Base assembly	1	1	Each	1	3	LA01	LA01
2	1400	1/4-20 Screw	4	4	Each	1	4	B100	LA01
2	1300	Hub	1	1	Each	2	5	B100	LA01
2	1200	6-Diameter steel plate	1	1	Each	3	6	B100	LA01
2	1100	Finished shaft	1	1	Each	2	5	B100	LA01
3	2100	3/8 Steel tubing	26	26	Inches	3	8	1100	LA01

The following code uses the Indented BOM data set produced in the previous invocation of PROC BOM to produce a data set that can be used by the NETDRAW procedure:

```
  /* Prepare the data set for running NETDRAW */
data IndBOM1a(drop=Part_ID);
   set IndBOM1;
   Paren_ID=Part_ID;
run;

data IndBOM1b;
   set IndBOM1(keep=Paren_ID Part_ID);
run;

data TreBOM1;
   set IndBOM1a IndBOM1b;
   LTnQP = put(LeadTime, f3.)||"  "||put(QtyPer, f3.);
run;
```

PROC NETDRAW is invoked with the NODISPLAY option to create a data set (Layout1) that contains all the layout information for the tree structure. The OUT= option specifies the name of the layout data set:

```
  /* Specify graphics options */
title h=1 j=c 'Multilevel Bill of Material with Lead-time Offset';
footnote h=0.5 j=l
         'Node shows Part Number, Lead-time, and Quantity Required';
pattern1 v=e c=gray;

  /* get the layout information for the BOM tree */
proc netdraw data=TreBOM1( where=(Paren_ID NE .) )
             out=Layout1 nodisplay;
   actnet / act=Paren_ID succ=Part_ID id=(_Part_ LTnQP Tot_Lead)
            ybetween=3 xbetween=15 tree
            rectilinear nodefid nolabel;
run;
```

In the next invocation, PROC NETDRAW uses a modified layout of the nodes to produce a diagram where the nodes are aligned according to the total lead time:

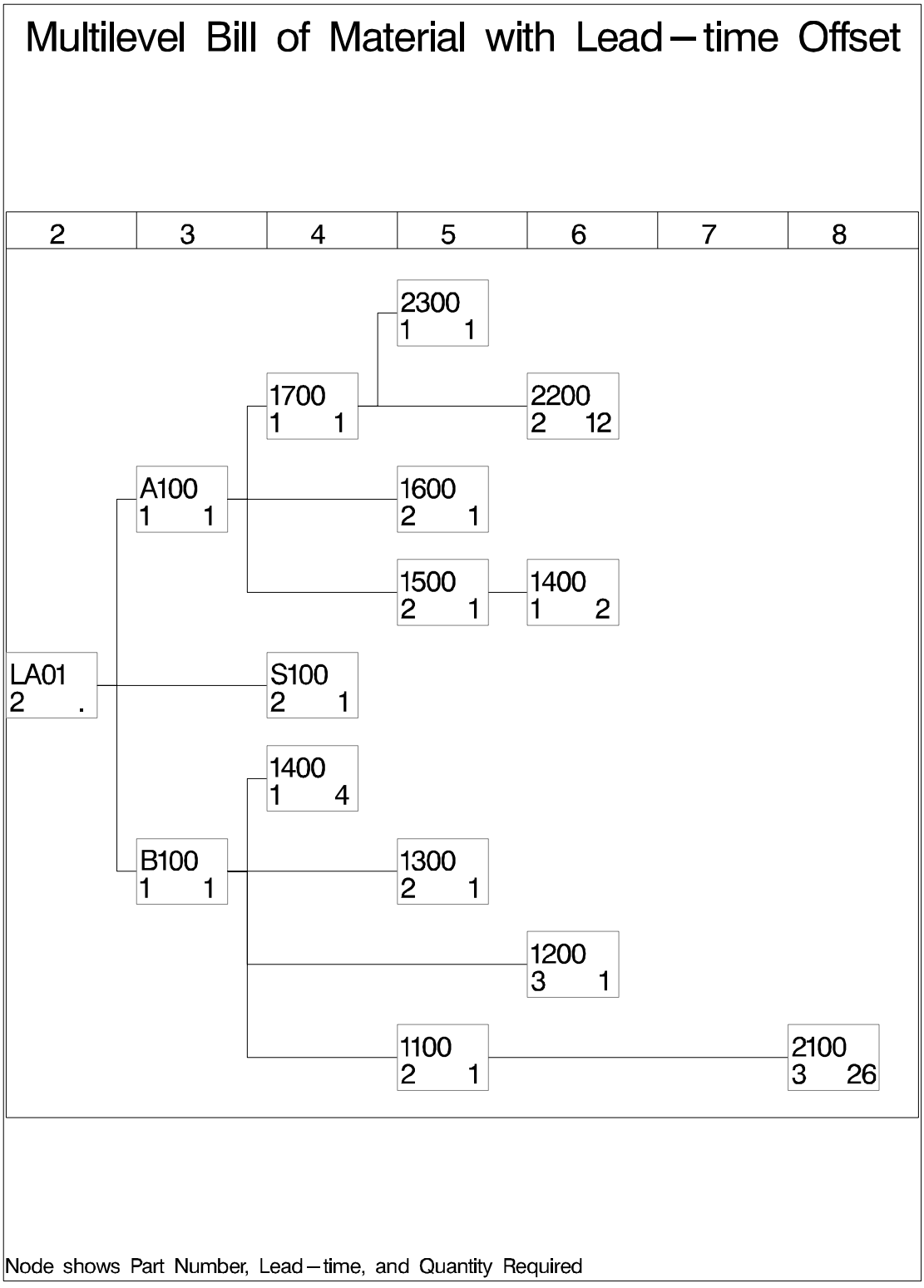
```
  /* Lead time offset the X coordinate of each node */
data TreBOM1a;
   set Layout1;
   if _seq_ = 0;
   drop _seq_;
   _x_ = Tot_Lead;
run;

  /* display the BOM tree with lead-time offset */
proc netdraw data=TreBOM1a;
   actnet / id=(_Part_ LTnQP)
            font=swiss ctext=black htext=3 carcs=black
            align=Tot_Lead frame pcompress
            xbetween=15 ybetween=3
            arrowhead=0 rectilinear nodefid nolabel;
run;
```

Refer to "The NETDRAW Procedure" chapter in the *SAS/OR User's Guide: Project Management* for details.

The resulting tree diagram is shown in Output 1.1.4.

Output 1.1.4. BOM Tree Structure with Lead-time Offset



Example 1.2. Planning Bill of Material

The previous examples dealt with the bill of material from the perspective of the manufacturing process. In this example, let's turn to another type of BOM that is often useful in planning and handling engineering charges. It is referred to as a *planning BOM*, a *pseudo BOM*, *super BOM*, or a *phantom BOM*.

Let's assume that the ABC Lamp Company sells lamps with 8 different shades (either 14 inches or 15 inches, in the color black, white, cream, or yellow), 3 alternate base plates (6 inches, 7 inches, or 8 inches), and 2 types of sockets ('One-way' and 'Three-way'). Working with all different options, the company now has 48 ($= 8 \times 3 \times 2$) different final products. In other words, the ABC Lamp Company has 48 different end items to forecast and plan. This might not sound too complicated; however, this example is only a simplified case. In the real world, some manufacturing companies making automobiles, computers, or machine tools can easily make over $5,000$ different final products. So many final products are obviously impossible to forecast accurately and plan with any hope of validity. However, a reasonable forecast for total lamp sales could be made, for example, $10,000$ per week. Some parts like the `'1100 Finished shaft'` are common to all configurations. Although these parts are produced independently of one another, they can be grouped as common parts on the BOM for administrative purposes. The new part that is created to group all common parts is never stocked and hence it is called a *phantom part*. Phantom items are generally assigned 0 lead times and lot-for-lot order quantity. Besides the common parts phantom item, other phantom items are created for planning different options. These items are also referred to as *model items*. This process is known as the *modularized bill of material construction*.

The following SAS code creates three data sets: The Phantom2 data set contains the generic part `'LAXX'` and all phantom items. The data set Option2 lists all options of each model item that are not already in the part master data set PMaster1 (described in Example 1.1 on page 21). Finally, the ParComp2 data set contains the re-grouped part-component relationship information:

```
   /* Generic and phantom part data */
data Phantom2;
   input Part      $8.
         Desc      $24.
         Req       8.0
         Unit      $8.
         LeadTime  4.0
         ;
datalines;
LAXX    Lamp LA                   10000 Each      3
4000    Common parts                  . Each      0
A10X    Socket assembly options       . Each      0
B10X    Base assembly options         . Each      0
S10X    Shade options                 . Each      0
;
```

```
  /* Additional options and alternative parts */
data Option2;
   input Part      $8.
         Desc      $24.
         Req       8.0
         Unit      $8.
         LeadTime  4.0
         ;
datalines;
A101    Three-way socket assem.        . Each      1
B101    7in Base assembly              . Each      1
B102    8in Base assembly              . Each      1
S101    14in White shade               . Each      2
S102    14in Cream shade               . Each      2
S103    14in Yellow shade              . Each      2
S104    15in Black shade               . Each      2
S105    15in White shade               . Each      2
S106    15in Cream shade               . Each      2
S107    15in Yellow shade              . Each      2
1201    7-Diameter steel plate         . Each      3
1202    8-Diameter steel plate         . Each      3
1601    Three-way socket               . Each      2
;

  /* part-component relationship data */
data ParComp2;
   input Part      $8.
         Component $8.
         QtyPer    8.2
         ;
datalines;
LAXX    4000        1.00
LAXX    B10X        1.00
LAXX    S10X        1.00
LAXX    A10X        1.00
4000    1100        1.00
4000    1300        1.00
4000    1400        4.00
4000    1500        1.00
4000    1700        1.00
B10X    B100        0.32
B10X    B101        0.41
B10X    B102        0.33
S10X    S100        0.07
S10X    S101        0.18
S10X    S102        0.24
S10X    S103        0.10
S10X    S104        0.06
S10X    S105        0.14
S10X    S106        0.22
S10X    S107        0.10
A10X    A100        0.11
A10X    A101        0.92
B100    1200        1.00
```

```
B101    1201        1.00
B102    1202        1.00
A100    1600        1.00
A101    1601        1.00
1100    2100       26.00
1500    1400        2.00
1700    2200       12.00
1700    2300        1.00
;
```

The part identified as `'4000'` is the phantom item for common parts `'1100'`, `'1300'`, `'1400'`, `'1500'`, and `'1700'`. Each available option is structured as a model item with a quantity per parent (identified as the QtyPer variable) that represents its forecast popularity or option percentage. Note that the total percentage of each option in this example is more than 100 percent (for example, the total percentage of the `'A10X: Socket assembly options'` is $0.11 + 0.92 = 1.03$). This extra percentage is used to cover the uncertainty of the exact percentage split. Using this procedure to cover possible high side demand for each option is called *option overplanning* (Fogarty, Blackstone, and Hoffmann 1991).

The new part master data set PMaster2 combines the generic and phantom part data, the additional options data, and the old part master file shown in Output 1.1.1 on page 23. The modularized single-level BOM data set, SlBOM2, combines the data set PMaster2 with the part-component data set ParComp2 by using PROC SQL. The SAS code to accomplish this is as follows:

```
  /* Append the old part master data to the new   */
  /* phantom item and additional option data sets */
data PMaster2;
   set Phantom2
       Option2
       PMaster1(where=(Part NE 'LA01'));
run;

proc sort data=PMaster2;
   by Part;
run;


proc sort data=ParComp2;
   by Part;
run;

  /* Merge the Part-Component information to the new */
  /* part master data to create an input data set    */
proc sql;
   create table SlBOM2 as
      select PMaster2.*,
                 ParComp2.Component, ParComp2.QtyPer
         from PMaster2 left join ParComp2
            on PMaster2.Part=ParComp2.Part;
quit;
```

Output 1.2.1. Modularized Single-Level Bills of Material

ABC Lamp Company

Modularized Single-Level Bills of Material

Part	Desc	Req	Unit	Lead Time	Component	Qty Per
1100	Finished shaft	.	Each	2	2100	26.00
1200	6-Diameter steel plate	.	Each	3		.
1201	7-Diameter steel plate	.	Each	3		.
1202	8-Diameter steel plate	.	Each	3		.
1300	Hub	.	Each	2		.
1400	1/4-20 Screw	.	Each	1		.
1500	Steel holder	.	Each	2	1400	2.00
1600	One-way socket	.	Each	2		.
1601	Three-way socket	.	Each	2		.
1700	Wiring assembly	.	Each	1	2200	12.00
1700	Wiring assembly	.	Each	1	2300	1.00
2100	3/8 Steel tubing	.	Inches	3		.
2200	16-Gauge lamp cord	.	Feet	2		.
2300	Standard plug terminal	.	Each	1		.
4000	Common parts	.	Each	0	1100	1.00
4000	Common parts	.	Each	0	1300	1.00
4000	Common parts	.	Each	0	1400	4.00
4000	Common parts	.	Each	0	1500	1.00
4000	Common parts	.	Each	0	1700	1.00
A100	Socket assembly	.	Each	1	1600	1.00
A101	Three-way socket assem.	.	Each	1	1601	1.00
A10X	Socket assembly options	.	Each	0	A100	0.11
A10X	Socket assembly options	.	Each	0	A101	0.92
B100	Base assembly	.	Each	1	1200	1.00
B101	7in Base assembly	.	Each	1	1201	1.00
B102	8in Base assembly	.	Each	1	1202	1.00
B10X	Base assembly options	.	Each	0	B100	0.32
B10X	Base assembly options	.	Each	0	B101	0.41
B10X	Base assembly options	.	Each	0	B102	0.33
LAXX	Lamp LA	10000	Each	3	4000	1.00
LAXX	Lamp LA	10000	Each	3	B10X	1.00
LAXX	Lamp LA	10000	Each	3	S10X	1.00
LAXX	Lamp LA	10000	Each	3	A10X	1.00
S100	Black shade	.	Each	2		.
S101	14in White shade	.	Each	2		.
S102	14in Cream shade	.	Each	2		.
S103	14in Yellow shade	.	Each	2		.
S104	15in Black shade	.	Each	2		.
S105	15in White shade	.	Each	2		.
S106	15in Cream shade	.	Each	2		.
S107	15in Yellow shade	.	Each	2		.
S10X	Shade options	.	Each	0	S100	0.07
S10X	Shade options	.	Each	0	S101	0.18
S10X	Shade options	.	Each	0	S102	0.24
S10X	Shade options	.	Each	0	S103	0.10
S10X	Shade options	.	Each	0	S104	0.06
S10X	Shade options	.	Each	0	S105	0.14
S10X	Shade options	.	Each	0	S106	0.22
S10X	Shade options	.	Each	0	S107	0.10

The following SAS code sorts and displays the modularized single-level BOM data set as shown in Output 1.2.1:

```
proc sort data=SlBOM2;
   by Part;
run;
```

```
  /* Display the single-level BOM data */
proc print data=SlBOM2 noobs;
   title 'ABC Lamp Company';
   title3 'Modularized Single-Level Bills of Material';
run;
```

The following code invokes PROC BOM to generate the planning BOM and the summarized BOM from the modularized single-level BOM:

```
  /* Generate the indented BOM and summarized BOM data sets */
proc bom data=SlBOM2 out=IndBOM2 summaryout=SumBOM2;
   structure / part=Part
               requirement=Req
               leadtime=LeadTime
               component=Component
               quantity=QtyPer
               id=(Desc Unit);
run;


   /* Display the indented BOM data */
proc print data=IndBOM2 noobs;
   var _Level_ _Part_ Desc QtyPer Qty_Prod
       Unit LeadTime Tot_Lead _Parent_ _Prod_;
   title 'ABC Lamp Company';
   title3 'Indented Bill of Material, Part LAXX';
run;


proc sort data=SumBOM2;
   by _Part_;
run;

  /* Display the summarized BOM data */
proc print data=SumBOM2 noobs;
   title 'ABC Lamp Company';
   title3 'Summarized Bill of Material, Part LAXX';
run;
```

The indented bill of material is shown in Output 1.2.2. Output 1.2.3 lists the quantity of each part that is needed to build $10,000$ lamps.

Output 1.2.2. Planning Bill of Material with Option Overplanning

ABC Lamp Company

Indented Bill of Material, Part LAXX

Level	Part	Desc	Qty Per	Qty Prod	Unit	Lead Time	Total Lead	Parent	Prod
0	LAXX	Lamp LA	.	1.00	Each	3	3		LAXX
1	A10X	Socket assembly options	1.00	1.00	Each	0	3	LAXX	LAXX
2	A101	Three-way socket assem.	0.92	0.92	Each	1	4	A10X	LAXX
3	1601	Three-way socket	1.00	0.92	Each	2	6	A101	LAXX
2	A100	Socket assembly	0.11	0.11	Each	1	4	A10X	LAXX
3	1600	One-way socket	1.00	0.11	Each	2	6	A100	LAXX
1	S10X	Shade options	1.00	1.00	Each	0	3	LAXX	LAXX
2	S107	15in Yellow shade	0.10	0.10	Each	2	5	S10X	LAXX
2	S106	15in Cream shade	0.22	0.22	Each	2	5	S10X	LAXX
2	S105	15in White shade	0.14	0.14	Each	2	5	S10X	LAXX
2	S104	15in Black shade	0.06	0.06	Each	2	5	S10X	LAXX
2	S103	14in Yellow shade	0.10	0.10	Each	2	5	S10X	LAXX
2	S102	14in Cream shade	0.24	0.24	Each	2	5	S10X	LAXX
2	S101	14in White shade	0.18	0.18	Each	2	5	S10X	LAXX
2	S100	Black shade	0.07	0.07	Each	2	5	S10X	LAXX
1	B10X	Base assembly options	1.00	1.00	Each	0	3	LAXX	LAXX
2	B102	8in Base assembly	0.33	0.33	Each	1	4	B10X	LAXX
3	1202	8-Diameter steel plate	1.00	0.33	Each	3	7	B102	LAXX
2	B101	7in Base assembly	0.41	0.41	Each	1	4	B10X	LAXX
3	1201	7-Diameter steel plate	1.00	0.41	Each	3	7	B101	LAXX
2	B100	Base assembly	0.32	0.32	Each	1	4	B10X	LAXX
3	1200	6-Diameter steel plate	1.00	0.32	Each	3	7	B100	LAXX
1	4000	Common parts	1.00	1.00	Each	0	3	LAXX	LAXX
2	1700	Wiring assembly	1.00	1.00	Each	1	4	4000	LAXX
3	2300	Standard plug terminal	1.00	1.00	Each	1	5	1700	LAXX
3	2200	16-Gauge lamp cord	12.00	12.00	Feet	2	6	1700	LAXX
2	1500	Steel holder	1.00	1.00	Each	2	5	4000	LAXX
3	1400	1/4-20 Screw	2.00	2.00	Each	1	6	1500	LAXX
2	1400	1/4-20 Screw	4.00	4.00	Each	1	4	4000	LAXX
2	1300	Hub	1.00	1.00	Each	2	5	4000	LAXX
2	1100	Finished shaft	1.00	1.00	Each	2	5	4000	LAXX
3	2100	3/8 Steel tubing	26.00	26.00	Inches	3	8	1100	LAXX

Output 1.2.3. Summarized Part Requirement

ABC Lamp Company

Summarized Bill of Material, Part LAXX

Part	Low_Code	Req	On_Hand	Net_Req	Desc	Unit
1100	2	10000	0	10000	Finished shaft	Each
1200	3	3200	0	3200	6-Diameter steel plate	Each
1201	3	4100	0	4100	7-Diameter steel plate	Each
1202	3	3300	0	3300	8-Diameter steel plate	Each
1300	2	10000	0	10000	Hub	Each
1400	3	60000	0	60000	1/4-20 Screw	Each
1500	2	10000	0	10000	Steel holder	Each
1600	3	1100	0	1100	One-way socket	Each
1601	3	9200	0	9200	Three-way socket	Each
1700	2	10000	0	10000	Wiring assembly	Each
2100	3	260000	0	260000	3/8 Steel tubing	Inches
2200	3	120000	0	120000	16-Gauge lamp cord	Feet
2300	3	10000	0	10000	Standard plug terminal	Each
4000	1	10000	0	10000	Common parts	Each
A100	2	1100	0	1100	Socket assembly	Each
A101	2	9200	0	9200	Three-way socket assem.	Each
A10X	1	10000	0	10000	Socket assembly options	Each
B100	2	3200	0	3200	Base assembly	Each
B101	2	4100	0	4100	7in Base assembly	Each
B102	2	3300	0	3300	8in Base assembly	Each
B10X	1	10000	0	10000	Base assembly options	Each
LAXX	0	10000	0	10000	Lamp LA	Each
S100	2	700	0	700	Black shade	Each
S101	2	1800	0	1800	14in White shade	Each
S102	2	2400	0	2400	14in Cream shade	Each
S103	2	1000	0	1000	14in Yellow shade	Each
S104	2	600	0	600	15in Black shade	Each
S105	2	1400	0	1400	15in White shade	Each
S106	2	2200	0	2200	15in Cream shade	Each
S107	2	1000	0	1000	15in Yellow shade	Each
S10X	1	10000	0	10000	Shade options	Each

Example 1.3. Bills of Material Verification

The previous example illustrated how some companies may have more than one type of bill of material. It is important that while the different types of BOM may differ in the way parts are grouped together, they must have the same component items (except those phantom parts) in the same quantities. This example demonstrates a simple method that compares the different types of BOM and detects any differences.

The transaction data set Trans3 contains the new values of the gross requirements of the lamp `'LAXX'` and the options. Note that the options `'B100'`, `'S100'`, and `'A100'` that are associated with the lamp `'LA01'` described in the "Getting Started" section on page 3, as well as the lamp `'LAXX'`, have gross requirements of 1. All other options have 0 gross requirements. Therefore, the item `'LAXX'` and all of the selected options are master schedule items. See the "Single-Level Bills of Material Data Set" section beginning on page 15 for a detailed description about master schedule items.

The following SAS DATA step code updates the value of the Req variable in the SlBOM2 data set (shown in Output 1.2.1 on page 30) with the value of the variable in the transaction data set Trans3:

```
  /* Gross requirement transaction data set */
data Trans3;
   input Part      $8.
         Req       8.0
         ;
datalines;
LAXX           1
B100           1
B101           0
B102           0
S100           1
S101           0
S102           0
S103           0
S104           0
S105           0
S106           0
S107           0
A100           1
A101           0
;

proc sort data=Trans3;
   by Part;
run;

  /* Update the gross requirement values of the */
  /* single-level BOM data set                  */
data SlBOM3(drop=OldReq);
   merge SlBOM2(rename=(Req=OldReq)) Trans3(in=in2);
   by Part;

   if not in2 then Req=OldReq;
run;
```

The following code invokes PROC BOM with the new input data set. The summarized bill of material is shown in Output 1.3.1:

```
   /* Generate the indented BOM and Summarized BOM */
proc bom data=SlBOM3 out=IndBOM3 summaryout=SumBOM3;
   structure / part=Part
               requirement=Req
               leadtime=LeadTime
               component=Component
               quantity=QtyPer
               id=(Desc Unit);
run;
```

```
   /* Sort and display the summarized BOM data */
proc sort data=SumBOM3;
   by _Part_;
run;

proc print data=SumBOM3(where=(Net_Req NE 0)) noobs;
   title 'ABC Lamp Company';
   title3 'Summarized Bill of Material, Part LA01';
run;
```

Output 1.3.1. Summarized Bills of Material

ABC Lamp Company

Summarized Bill of Material, Part LA01

Part	Low_Code	Req	On_Hand	Net_Req	Desc	Unit
1100	2	1	0	1	Finished shaft	Each
1200	3	1	0	1	6-Diameter steel plate	Each
1300	2	1	0	1	Hub	Each
1400	3	6	0	6	1/4-20 Screw	Each
1500	2	1	0	1	Steel holder	Each
1600	3	1	0	1	One-way socket	Each
1700	2	1	0	1	Wiring assembly	Each
2100	3	26	0	26	3/8 Steel tubing	Inches
2200	3	12	0	12	16-Gauge lamp cord	Feet
2300	3	1	0	1	Standard plug terminal	Each
4000	1	1	0	1	Common parts	Each
A100	2	1	0	1	Socket assembly	Each
A10X	1	1	0	1	Socket assembly options	Each
B100	2	1	0	1	Base assembly	Each
B10X	1	1	0	1	Base assembly options	Each
LAXX	0	1	0	1	Lamp LA	Each
S100	2	1	0	1	Black shade	Each
S10X	1	1	0	1	Shade options	Each

By comparing the summarized BOM in Output 1.3.1 with the one displayed in Figure 1.6 on page 10, you can verify that the two bills of material are in agreement except for the low-level code of each part and the phantom items, such as `'4000 Common parts'`, `'A10X Socket assembly options'`, `'B10X Base assembly options'`, and `'S10X Shade options'`.

References

Aho, A.V., Hopcroft, J.E., and Ullman, J.D. (1983), *Data Structures and Algorithms*, Reading, MA: Addison-Wesley.

Clement, J., Coldrick, A., and Sari, J. (1992), *Manufacturing Data Structures: Building Foundations for Excellence with Bills of Materials and Process Information*, Essex Junction, VT: Oliver Wright Limited Publications, Inc.

Cox, J.F., III, and Blackstone, J.H., Jr., eds. (1998), *APICS Dictionary*, Ninth Edition, Alexandria, VA: APICS.

Fogarty, D.W., Blackstone, J.H., and Hoffmann, T.R. (1991), *Production and Inventory Management*, Second Edition, Cincinnati, OH: South-Western Publishing Co.

Chapter 2
The CPM Procedure

Chapter Table of Contents

Chapter 2
The CPM Procedure

Overview

Enhancements have been added to the CPM procedure as follows:

- The SETFINISHMILESTONE option has been added to the CPM procedure statement to enable you to specify that milestones (zero duration activities) should have the same start and finish times as the finish time of their predecessor. In other words, this option enables milestones that mark the *end* of the preceding activity to coincide with its finish time.
- The FIXASTART option has been added to the ACTUAL statement to enable you to request that the actual start and finish times of an activity should be left unchanged even if they coincide with a non-working time. Thus, if the actual start time is specified to be sometime on Sunday, it is left unchanged even if Sunday is a non-working day in the activity's calendar.

Syntax

PROC CPM Statement

The following new option is available in the CPM procedure statement:

SETFINISHMILESTONE
specifies that milestones (zero duration activities) should have the same start and finish times as the finish time of their predecessor. In other words, this option enables milestones that mark the *end* of the preceding activity to coincide with its finish time. By default, if a milestone M is a successor to an activity that finishes at the end of the day (say 15Mar2000), the start and finish times for the milestone are specified as the beginning of the next day (16Mar2000). This corresponds to the definition of start times in the CPM procedure: *all* start times indicate the *beginning* of the date specified. For zero duration activities, the finish time is defined to be the same as the start time. The SETFINISHMILESTONE option specifies that the start and finish times for the milestone M should be specified as 15Mar2000, with the interpretation that the milestone's schedule corresponds to the *end* of the day. There may be exceptions to this definition if there are special alignment constraints on the milestone. For details, see the "Finish Milestones" section on page 40.

ACTUAL Statement

The following new option is available in the ACTUAL statement:

FIXASTART
specifies that the actual start time of an activity should not be overwritten if it is specified to be on a non-work day. By default, none of the start or finish times of an activity can occur during a non-work period corresponding to the activity's calendar. If the actual start time is specified on a non-work day, it is moved to the nearest work day. The FIXASTART option specifies that the actual start and finish times be left unchanged even if they coincide with a non-working time. Thus, if the actual start time is specified to be sometime on Sunday, it is left unchanged even if Sunday is a non-working day in the activity's calendar.

Details

Finish Milestones

By default, the start and finish times for the different schedules computed by PROC CPM denote the first and last *day* of work, respectively, when the values are formatted as SAS *date* values. All start times are assumed to denote the beginning of the day and all finish times are assumed to correspond to the end of the day. If the times are SAS *time* or *datetime* values, they denote the first and last *second* of work, respectively. However, for zero duration activities, *both* the start and the finish times correspond to the beginning of the date (or second) specified.

Thus, according to the preceding definitions, the CPM procedure assumes that all milestones are scheduled at the *beginning* of the day indicated by their start times. In other words, the milestones can be regarded as *start* milestones since they correspond to the *beginning* of the time period indicated by their scheduled times. However, in some situations, you may want to treat the milestones as *finish* milestones.

Consider the following example:

Activity 'A' has a 2-day duration and is followed by a milestone (zero duration) activity, 'B'. Suppose that activity 'A' starts on March 15, 2000. The default calculations by the CPM procedure will produce the following schedule for the two activities:

OBS	Activity	Duration	E_START	E_FINISH
1	A	2	15MAR2000	16MAR2000
2	B	0	17Mar2000	17MAR2000

The start and finish times of the milestone activity, 'B', are interpreted as the beginning of March 17, 2000. In some situations, you may want the milestones to start and finish on the same day as their predecessors. For instance, in this example, you may want the start and finish time of activity 'B' to be set to March 16, 2000, with the interpretation that the time corresponds to the *end* of the day. Such milestones will be referred to as *finish milestones*.

The SETFINISHMILESTONE option in the PROC CPM statement indicates that a milestone that is linked to its predecessor by a *Finish-to-Start* or a *Finish-to-Finish* precedence constraint should be treated as a *finish milestone*. In other words, such a milestone should have the start and finish time set to the *end* of the day that the predecessor activity finishes. There are some exceptions to this rule:

- There is an alignment constraint on activity 'B' that requires the milestone to start on a later day than the date dictated by the precedence constraint.
- Activity 'B' has an actual start or finish time specified that is inconsistent with the predecessor's finish date.

Note that the alignment constraint that affects the early schedule of the project may not have any impact on the late schedule. Thus, a milestone may be treated as a finish milestone for the late schedule even if it is not a finish milestone according to the early schedule. See Example 2.1 for an illustration of this situation. In addition, while computing the resource-constrained schedule, a start milestone (according to the early schedule) may in fact turn out to be a finish milestone according to the resource-constrained schedule.

Since the same milestone could be treated as either a start or a finish milestone depending on the presence or absence of an alignment constraint, or depending on the type of the schedule (early, late, resource-constrained, or actual), the CPM procedure adds extra variables to the Schedule data set corresponding to each type of schedule. These variables, EFINMILE, LFINMILE, SFINMILE, and AFINMILE, indicate for each milestone activity in the project whether the corresponding schedule times (early, late, resource-constrained, or actual) are to be interpreted as finish milestone times. These variables have a value of '1' if the milestone is treated as a finish milestone for the corresponding schedule; otherwise, the value is missing. In addition to providing an unambiguous interpretation for the schedule times of the milestones, these variables are useful in plotting the schedules correctly using the Gantt procedure. (See Example 2.1.)

Examples

Example 2.1. Use of the SETFINISHMILESTONE Option

A simple activity network is used to illustrate the use of the SETFINISHMILESTONE option in a couple of different scenarios.

The following DATA step reads the project network in AON format into a SAS data set named tasks. The data set (printed in Output 2.1.1) contains an Activity variable (act), a Successor variable (succ), a Lag variable (lag), and a Duration variable (dur). Note that there are several milestones linked to other activities through different types of precedence constraints. The data set also contains some alignment constraints as specified by the variables target and trgttype. Note that the treatment of the milestones will vary depending on the presence or absence of the alignment constraints. The data set also contains two variables that indicate the expected early

schedule dates for the milestones corresponding to two different invocations of PROC CPM: the variable notrgtmd corresponds to the non-aligned schedule and the variable miledate corresponds to an invocation with the ALIGNDATE statement.

```
data tasks;
   input act $ 1-7 succ $ 9-15 lag $ dur target date9.
         trgttype $ miledate date9. notrgtmd date9.;
   format target date7. miledate date7. notrgtmd date7.;
datalines;
Task 0  Mile 1  ss_0 1 24Jan00  SGE .        .
Mile 1  Task 2  .    0  .       .   24Jan00  24Jan00
Task 2  .       .    1  .       .   .        .
Task 3  Mile 4  .    1  .       .   .        .
Mile 4  .       .    0  .       .   24Jan00  24Jan00
Task 5  Mile 6  .    1  .       .   .        .
Mile 6  Mile 7  FS_1 0  .       .   24Jan00  24Jan00
Mile 7  .       .    0  .       .   25Jan00  25Jan00
Task 8  Mile 9  SS_3 1  .       .   .        .
Mile 9  Mile 10 .    0  .       .   27Jan00  27Jan00
Mile 10 .       .    0  .       .   27Jan00  27Jan00
Task 11 Mile 12 .    2  .       .   .        .
Mile 12 Mile 13 FS_1 0 26Jan00  SGE 26Jan00  25Jan00
Mile 13 .       .    0  .       .   27Jan00  26Jan00
;
```

Output 2.1.1. Input Data Set

Input Data Set

Obs	act	succ	lag	dur	target	trgttype	miledate	notrgtmd
1	Task 0	Mile 1	ss_0	1	24JAN00	SGE	.	.
2	Mile 1	Task 2		0	.		24JAN00	24JAN00
3	Task 2			1	.		.	.
4	Task 3	Mile 4		1	.		.	.
5	Mile 4			0	.		24JAN00	24JAN00
6	Task 5	Mile 6		1	.		.	.
7	Mile 6	Mile 7	FS_1	0	.		24JAN00	24JAN00
8	Mile 7			0	.		25JAN00	25JAN00
9	Task 8	Mile 9	SS_3	1	.		.	.
10	Mile 9	Mile 10		0	.		27JAN00	27JAN00
11	Mile 10			0	.		27JAN00	27JAN00
12	Task 11	Mile 12		2	.		.	.
13	Mile 12	Mile 13	FS_1	0	26JAN00	SGE	26JAN00	25JAN00
14	Mile 13			0	.		27JAN00	26JAN00

Output 2.1.2. Default Schedule

Default Schedule

Obs	activity	succ	dur	lag	notrgtmd	E_START	E_FINISH	L_START	L_FINISH	TOTFLOT	FREFLOT
1	Task 0	Mile 1	1	ss_0	.	24JAN00	24JAN00	26JAN00	26JAN00	2	0
2	Mile 1	Task 2	0		24JAN00	24JAN00	24JAN00	26JAN00	26JAN00	2	0
3	Task 2		1		.	24JAN00	24JAN00	26JAN00	26JAN00	2	2
4	Task 3	Mile 4	1		.	24JAN00	24JAN00	26JAN00	26JAN00	2	0
5	Mile 4		0		24JAN00	25JAN00	25JAN00	27JAN00	27JAN00	2	2
6	Task 5	Mile 6	1		.	24JAN00	24JAN00	25JAN00	25JAN00	1	0
7	Mile 6	Mile 7	0	FS_1	24JAN00	25JAN00	25JAN00	26JAN00	26JAN00	1	0
8	Mile 7		0		25JAN00	26JAN00	26JAN00	27JAN00	27JAN00	1	1
9	Task 8	Mile 9	1	SS_3	.	24JAN00	24JAN00	24JAN00	24JAN00	0	0
10	Mile 9	Mile 10	0		27JAN00	27JAN00	27JAN00	27JAN00	27JAN00	0	0
11	Mile 10		0		27JAN00	27JAN00	27JAN00	27JAN00	27JAN00	0	0
12	Task 11	Mile 12	2		.	24JAN00	25JAN00	24JAN00	25JAN00	0	0
13	Mile 12	Mile 13	0	FS_1	25JAN00	26JAN00	26JAN00	26JAN00	26JAN00	0	0
14	Mile 13		0		26JAN00	27JAN00	27JAN00	27JAN00	27JAN00	0	0

First, the CPM procedure is invoked with the default treatment of milestones. The resulting schedule is printed in Output 2.1.2. Note the dates for the milestones. Compare these dates with the values of the variable notrgtmd that contains the desired milestone schedule dates corresponding to the finish times of their predecessor.

```
/* Schedule the project */
proc cpm data=tasks out=out0
         collapse interval=day
         date='24jan00'd;
   activity act;
   successor succ /lag=(lag);
   duration dur;
   id lag notrgtmd;
   run;

title 'Default Schedule';
proc print; run;
```

Next, the CPM procedure is invoked with the option SETFINISHMILESTONE and the resulting schedule is printed in Output 2.1.3. The variables EFINMILE and LFINMILE indicate if the milestone is a finish milestone or not. For example, the milestone 'Mile 12' has E_FINISH = 25JAN00 and the value of EFINMILE is '1', indicating that the activity finishes at the end of the day on January 25, 2000. The milestone 'Mile 13' (with a finish-to-start lag of 1 day) finishes at the end of the day on January 26, 2000. In fact, as the late finish schedule indicates, the value of L_FINISH for 'Mile 13' (and the project finish time) is the end of the day on 26JAN00. Note that both the variables EFINMILE and LFINMILE have the same values for all the activities in this example.

```
proc cpm data=tasks out=out1
         collapse interval=day
         date='24jan00'd
         setfinishmilestone;
   activity act;
   successor succ /lag=(lag);
   duration dur;
   id lag notrgtmd;
   run;

title 'Schedule with option SETFINISHMILESTONE';
title2 'No Target Dates';
proc print;
  id act;
  var succ lag dur notrgtmd e_start e_finish
      l_start l_finish efinmile lfinmile;
  run;
```

Output 2.1.3. Schedule with SETFINISHMILESTONE Option

Schedule with option SETFINISHMILESTONE
No Target Dates

act	succ	lag	dur	notrgtmd	E_START	E_FINISH	L_START	L_FINISH	EFINMILE	LFINMILE
Task 0	Mile 1	ss_0	1	.	24JAN00	24JAN00	26JAN00	26JAN00	.	.
Mile 1	Task 2		0	24JAN00	24JAN00	24JAN00	26JAN00	26JAN00	.	.
Task 2			1	.	24JAN00	24JAN00	26JAN00	26JAN00	.	.
Task 3	Mile 4		1	.	24JAN00	24JAN00	26JAN00	26JAN00	.	.
Mile 4			0	24JAN00	24JAN00	24JAN00	26JAN00	26JAN00	1	1
Task 5	Mile 6		1	.	24JAN00	24JAN00	25JAN00	25JAN00	.	.
Mile 6	Mile 7	FS_1	0	24JAN00	24JAN00	24JAN00	25JAN00	25JAN00	1	1
Mile 7			0	25JAN00	25JAN00	25JAN00	26JAN00	26JAN00	1	1
Task 8	Mile 9	SS_3	1	.	24JAN00	24JAN00	24JAN00	24JAN00	.	.
Mile 9	Mile 10		0	27JAN00	27JAN00	27JAN00	27JAN00	27JAN00	.	.
Mile 10			0	27JAN00	27JAN00	27JAN00	27JAN00	27JAN00	.	.
Task 11	Mile 12		2	.	24JAN00	25JAN00	24JAN00	25JAN00	.	.
Mile 12	Mile 13	FS_1	0	25JAN00	25JAN00	25JAN00	25JAN00	25JAN00	1	1
Mile 13			0	26JAN00	26JAN00	26JAN00	26JAN00	26JAN00	1	1

The next invocation of CPM illustrates the effect of alignment constraints on the milestones. As explained in the "Finish Milestones" section on page 40, imposing an alignment constraint of type SGE on a milestone may change it from a finish milestone to a start milestone (default behavior) as far as the early schedule of the project is concerned. In the following program, the CPM procedure is invoked with the SETFINISHMILESTONE option and the ALIGNDATE and ALIGNTYPE statements. The resulting schedule is printed in Output 2.1.4. Note that the early schedule of the milestones now corresponds to the values in the variable miledate. Note also that the activities 'Mile 12' and 'Mile 13' are no longer finish milestones, as indicated by missing values for the variable EFINMILE.

```
proc cpm data=tasks out=out2
         collapse
         interval=day
         date='24jan00'd
         setfinishmilestone;
   activity act;
   successor succ /lag=(lag);
   duration dur;
   aligndate target;
   aligntype trgttype;
   id target trgttype lag miledate;
   run;

title 'Schedule with option SETFINISHMILESTONE';
title2 'Target Dates change Early Schedule for some Milestones';
proc print;
  id act;
  var succ lag target trgttype miledate e_start e_finish
      l_start l_finish efinmile lfinmile;
  run;
```

Output 2.1.4. Effect of Alignment Constraints

```
              Schedule with option SETFINISHMILESTONE
          Target Dates change Early Schedule for some Milestones

                              t       m             E             L E L
                              r       i       E     _       L     _ F F
                          t   g       l       _     F       _     F I I
                          a   t       e       S     I       S     I N N
          s               r   t       d       T     N       T     N M M
   a      u     l          g   y       a       A     I       A     I I I
   c      c     a          e   p       t       R     S       R     S L L
   t      c     g          t   e       e       T     H       T     H E E

Task 0  Mile 1  ss_0 24JAN00 SGE       . 24JAN00 24JAN00 26JAN00 26JAN00 . .
Mile 1  Task 2              .     24JAN00 24JAN00 24JAN00 26JAN00 26JAN00 . .
Task 2                      .           . 24JAN00 24JAN00 26JAN00 26JAN00 . .
Task 3  Mile 4              .           . 24JAN00 24JAN00 26JAN00 26JAN00 . .
Mile 4                      .     24JAN00 24JAN00 24JAN00 26JAN00 26JAN00 1 1
Task 5  Mile 6              .           . 24JAN00 24JAN00 25JAN00 25JAN00 . .
Mile 6  Mile 7  FS_1        .     24JAN00 24JAN00 24JAN00 25JAN00 25JAN00 1 1
Mile 7                      .     25JAN00 25JAN00 25JAN00 26JAN00 26JAN00 1 1
Task 8  Mile 9  SS_3        .           . 24JAN00 24JAN00 24JAN00 24JAN00 . .
Mile 9  Mile 10             .     27JAN00 27JAN00 27JAN00 27JAN00 27JAN00 . .
Mile 10                     .     27JAN00 27JAN00 27JAN00 27JAN00 27JAN00 . .
Task 11 Mile 12             .           . 24JAN00 25JAN00 24JAN00 25JAN00 . .
Mile 12 Mile 13 FS_1 26JAN00 SGE 26JAN00 26JAN00 26JAN00 25JAN00 25JAN00 . 1
Mile 13                     .     27JAN00 27JAN00 27JAN00 26JAN00 26JAN00 . 1
```

The interpretation of the start and finish times for a milestone depends on whether it is a start milestone or a finish milestone. By default, all milestones are start milestones and are assumed to be scheduled at the beginning of the date specified in the start or finish time variable. As such, PROC GANTT displays these milestones at the start of the corresponding days on the Gantt chart. However, if a milestone is a finish milestone then it may not be displayed correctly on the Gantt chart, depending on the scale of the display.

In this example, PROC GANTT is used to display the schedule produced in Output 2.1.4. Recall that the schedule is saved in the data set out2. First, PROC GANTT is invoked without any modifications to the schedule data set. The resulting Gantt chart is displayed in Output 2.1.5. Note that the finish milestones (with values of EFINMILE = '1') are not plotted correctly. For example, 'Mile 6' is plotted at the *beginning* instead of the *end* of the schedule bar for the predecessor activity, 'Act 5'. To correct this problem, you can adjust the schedule variables for the finish milestones and plot the new values, as illustrated by the second invocation of PROC GANTT. The corrected Gantt chart is displayed in Output 2.1.6.

```
title h=1.5
    'Schedule with option SETFINISHMILESTONE and ALIGNDATE';
title2 'Gantt Chart of Early Schedule without adjustment';
proc gantt data=out2(drop=l_:);
   chart / compress act=act succ=succ lag=lag
           font=swiss scale=7
           cprec=cyan cmile=magenta
           caxis=black cframe=ligr;
           dur=dur nojobnum nolegend;
   id act succ lag e_start efinmile;
   run;

/* Save adjusted E_START and E_FINISH times for finish
   milestones */
data temp;
   set out2;
   format estart efinish date7.;
   estart = e_start;
   efinish = e_finish;
   if efinmile then do;
      estart=estart+1;
      efinish=efinish+1;
      end;
   run;

/* Plot the adjusted start and finish times for the
   early schedule */
title h=1.5
   'Schedule with option SETFINISHMILESTONE and ALIGNDATE';
title2 'Gantt Chart of Early Schedule after adjustment';
proc gantt data=temp(drop=l_:);
   chart / compress act=act succ=succ lag=lag
           font=swiss scale=7
           es=estart ef=efinish
```

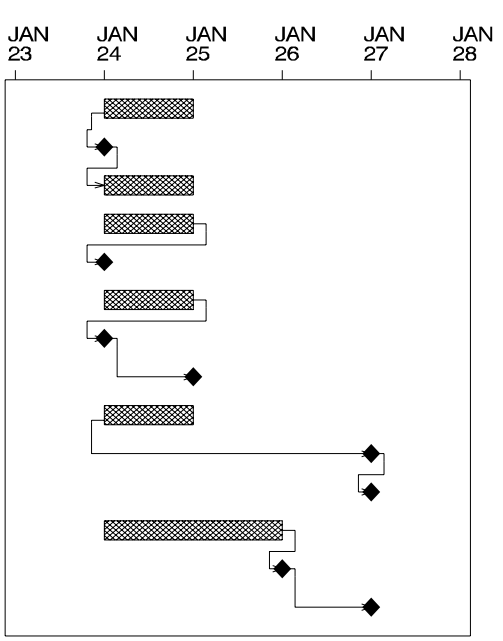
```
        cprec=cyan cmile=magenta
        caxis=black cframe=ligr;
        dur=dur nojobnum  nolegend;
id act succ lag e_start efinmile;
run;
```

Output 2.1.5. Gantt Chart of Unadjusted Schedule

Schedule with option SETFINISHMILESTONE and ALIGNDATE

Gantt Chart of Early Schedule without adjustment

act	succ	lag	Early Start	Early Finish Milestone
Task 0	Mile 1	ss_0	24JAN00	.
Mile 1	Task 2		24JAN00	.
Task 2			24JAN00	.
Task 3	Mile 4		24JAN00	.
Mile 4			24JAN00	1
Task 5	Mile 6		24JAN00	.
Mile 6	Mile 7	FS_1	24JAN00	1
Mile 7			25JAN00	1
Task 8	Mile 9	SS_3	24JAN00	.
Mile 9	Mile 10		27JAN00	.
Mile 10			27JAN00	.
Task 11	Mile 12		24JAN00	.
Mile 12	Mile 13	FS_1	26JAN00	.
Mile 13			27JAN00	.

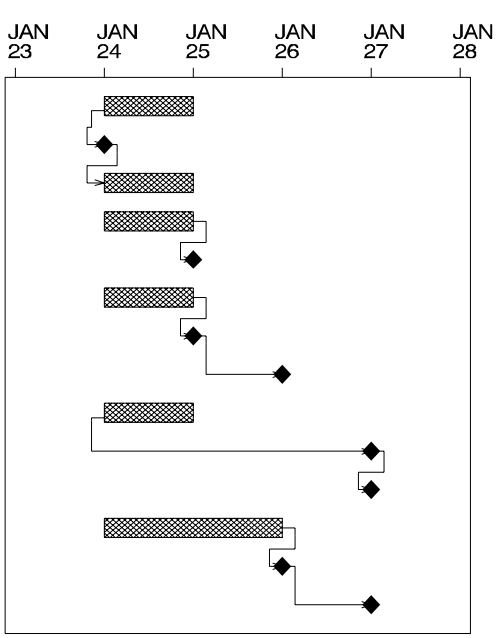


Output 2.1.6. Gantt Chart of Adjusted Schedule

Schedule with option SETFINISHMILESTONE and ALIGNDATE

Gantt Chart of Early Schedule after adjustment

act	succ	lag	Early Start	Early Finish Milestone
Task 0	Mile 1	ss_0	24JAN00	.
Mile 1	Task 2		24JAN00	.
Task 2			24JAN00	.
Task 3	Mile 4		24JAN00	.
Mile 4			24JAN00	1
Task 5	Mile 6		24JAN00	.
Mile 6	Mile 7	FS_1	24JAN00	1
Mile 7			25JAN00	1
Task 8	Mile 9	SS_3	24JAN00	.
Mile 9	Mile 10		27JAN00	.
Mile 10			27JAN00	.
Task 11	Mile 12		24JAN00	.
Mile 12	Mile 13	FS_1	26JAN00	.
Mile 13			27JAN00	.



Chapter 3
The GANTT Procedure

Chapter Table of Contents

Chapter 3
The GANTT Procedure

Overview

The following enhancements have been made to the Gantt procedure:

- The milestone color specifications have been made more flexible. The milestone colors are now determined using the same rules as for schedule bars, and consequently, they can also be set individually.
- The WEB= variable enables you to define an HTML reference for each activity. In the past the URL was only associated with the schedule bars for the activity. This association has now been extended to include milestones and ID variables.

Syntax

The following options have been enhanced in PROC GANTT:

CHART Statement

Graphics Options

CMILE=*color*

specifies the color to use for drawing the milestone symbol on the chart. If the CMILE= option is not specified, the default color of the milestones follows the rules for coloring the bars of the relevant schedule. For example, the milestone depicting a critical activity is drawn with the color of the fill pattern used for critical activities. For an activity with slack, the early start and late start milestone are drawn with the color of the fill pattern used for the duration and the slack time of a noncritical activity, respectively. You can also control the color at the activity level by using a PATTERN variable.

WEB=*variable*
HTML=*variable*

specifies the character variable in the schedule data set that identifies an HTML page for each activity. The procedure generates an HTML image map using this information for all the schedule bars, milestones, and ID variables corresponding to an activity.

Details

Web-Enabled Gantt Charts

The WEB= variable enables you to define an HTML reference for each activity. This HTML reference is currently associated with all the schedule bars, milestones, and ID variables that correspond to the activity. The WEB= variable is a character variable, and the values need to be of the form "HREF=htmlpage."

In addition, you can also store the coordinate and link information defined via the WEB= option in a SAS data set by specifying the IMAGEMAP= option in the PROC GANTT statement. By processing this SAS data set using a DATA step, you can generate customized HTML pages for your Gantt chart.

Chapter 4
The INTPOINT Procedure

Chapter Table of Contents

Chapter 4
The INTPOINT Procedure

Overview

The new INTPOINT procedure solves the Network Program with Side Constraints (NPSC) problem (defined in the "Mathematical Description of NPSC" section on page 56) and the more general Linear Programming (LP) problem (defined in the "Mathematical Description of LP" section on page 58).

NPSC and LP models can be used to describe a wide variety of real-world applications ranging from production, inventory, and distribution problems to financial applications.

Whether your problem is NPSC or LP, PROC INTPOINT uses the same optimization algorithm, the Interior Point algorithm. This algorithm is outlined in the section "The Interior Point Algorithm" on page 58.

While many of your problems may best be formulated as LP problems, there may be other instances when your problems are better formulated as NPSC problems. The "Network Models" section on page 66 describes typical models that have a network component and suggests reasons why NPSC may be preferable to LP. The "Getting Started: NPSC Problems" section on page 74 outlines how you supply data of any NPCS problem to the PROC INTPOINT and call the procedure. After it reads the NPSC data, PROC INTPOINT converts the problem into an equivalent LP problem, performs Interior Point optimization, then converts the solution it finds back into a form you can use as the optimum to the original NPSC model.

If your model is an LP problem, the way you supply the data to PROC INTPOINT and run the procedure is described in the "Getting Started: LP Problems" section on page 81.

The remainder of this chapter is organized as follows:

- The "Typical PROC INTPOINT Run" section on page 89 describes how to use this procedure.
- The "Functional Summary" section on page 93 lists the statements and options that can be used to control PROC INTPOINT.
- The "Syntax" section on page 91 describes all the statements and options of PROC INTPOINT.
- The "Details" section on page 125 contains fuller explanations, descriptions, and advice on the use and behavior of the procedure.
- PROC INTPOINT is demonstrated by solving several examples in the "Examples" section on page 155.
- The "References" section on page 181 concludes the chapter.

Mathematical Description of NPSC

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

Often all the details of a problem cannot be specified in a network model alone. In many of these cases, these details can be represented by the addition of side constraints to the model. Side constraints are linear functions of arc variables (variables containing flow through an arc) and nonarc variables (variables that are not part of the network). The data for a side constraint consist of coefficients of arcs and coefficients of nonarc variables, a constraint type (that is, $\leq$, =, or $\geq$) and a right-hand-side value (rhs). A nonarc variable has a name, an objective function coefficient analogous to an arc cost, an upper bound analogous to an arc capacity, and a lower bound analogous to an arc lower flow bound.

If a network component of NPSC is removed by merging arcs and nonarc variables into a single set of variables, and if the flow conservation constraints and side constraints are merged into a single set of constraints, the result is an LP problem. PROC INTPOINT will automatically transform an NPSC problem into an equivalent LP problem, perform the optimization, then transform the problem back into its original form. By doing this, PROC INTPOINT finds the flow through the network and the values of any nonarc variables that minimize the total cost of the solution. Flow conservation is met, flow through each arc is on or between the arc's lower flow bound and capacity, the value of each nonarc variable is on or between the nonarc's lower and upper bounds, and the side constraints are satisfied.

Note that, since many LPs have large embedded networks, PROC INTPOINT is an attractive alternative to the LP procedure in many cases. Rather than formulating all problems as LPs, network models remain conceptually easy since they are based on network diagrams that represent the problem pictorially. PROC INTPOINT accepts the network specification in a format that is particularly suited to networks. This not only simplifies problem description but also aids in the interpretation of the solution. The conversion to and from the equivalent LP is done "behind the scenes" by the procedure.

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

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

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

where

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

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

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

z is the g x 1 nonarc variable value vector

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

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

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

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

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

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

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

l is the a x 1 arc lower flow bound vector

u is the a x 1 arc capacity vector

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

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

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

The INTPOINT procedure can also be used to solve a network problem with side constraints but no nonarc variables, in which case Q, d, and z do not exist.

Mathematical Description of LP

PROC INTPOINT solves LP problems. These have a linear objective function and a collection of linear constraints. PROC INTPOINT finds the values of variables that minimize the total cost of the solution. The value of each variable is on or between the variable's lower and upper bounds, and the constraints are satisfied.

If an LP has g variables and k constraints, then the formal statement of the problem solved by PROC INTPOINT is

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

where

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

z is the g x 1 variable value vector

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

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

m is the g x 1 variable value lower bound vector

v is the g x 1 variable value upper bound vector

The Interior Point Algorithm

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

There are many variations of Interior Point algorithms. PROC INTPOINT uses the Primal-Dual with Predictor-Corrector algorithm. More information on this particular algorithm and related theory can be found in the texts by Roos, Terlaky, and Vial (1997), Wright (1996), and Ye (1996).

Interior Point Algorithmic Details

After preprocessing, the Linear Program to be solved is

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

subject to $Ax = b$

$x \geq 0$

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

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

$max\{b^T y\}$

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

$s \geq 0$

where y are dual variables, and s are dual constraint slacks

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

$Ax = b$

$A^T y + s = c$

$XSe = 0$

$x \geq 0$

$s \geq 0$

where $S = diag(s)$, (that is, $S_{i,j} = s_i$ if $i = j$, $S_{i,j} = 0$ otherwise)

$X = diag(x)$, and

$e_i = 1 \forall i$

These are the conditions for feasibility, with the *complementarity* 112

condition $XSe = 0$ added. $c^T x = b^T y$ must occur at the optimum. Complementarity forces the optimal objectives of the primal and dual to be equal, $c^T x_{opt} = b^T y_{opt}$, as

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

therefore $0 = c^T x_{opt} - b^T y_{opt}$

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

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

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

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

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

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

To greatly improve performance, the third equation is changed to

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

where $\mu_k = (x^k)^T s^k / n$, the average complementarity, and
$0 \leq \sigma_k \leq 1$

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

to the optimum, substantial progress toward the optimum can usually be made in the next iteration.

The Central Path is crucial to why the Interior Point algorithm is so efficient. As μ is decreased, this path "guides" the algorithm to the optimum through the interior of feasible space. Without centering, the algorithm would find a series of solutions near each other close to the boundary of feasible space. Step lengths along the direction would be small and many more iterations would probably be required to reach the optimum.

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

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

$$A\Delta x = infeas_c$$
$$A^T\Delta y + \Delta s = infeas_d$$
$$S\Delta x + X\Delta s = -XSe + \sigma\mu e$$

Rearranging the second equation

$$\Delta s = infeas_d - A^T\Delta y$$

Rearranging the third equation

$$\Delta s = X^{-1}(-S\Delta x - XSe + \sigma\mu e)$$
$$\Delta s = -\Theta\Delta x - Se + X^{-1}\sigma\mu e$$

where $\Theta = SX^{-1}$

Equating these two expressions for Δs and rearranging

$$-\Theta\Delta x - Se + X^{-1}\sigma\mu e = infeas_d - A^T\Delta y$$
$$-\Theta\Delta x = Se - X^{-1}\sigma\mu e + infeas_d - A^T\Delta y$$
$$\Delta x = \Theta^{-1}(-Se + X^{-1}\sigma\mu e - infeas_d + A^T\Delta y)$$
$$\Delta x = \rho + \Theta^{-1}A^T\Delta y$$

where $\rho = \Theta^{-1}(-Se + X^{-1}\sigma\mu e - infeas_d)$

Substituting into the first direction equation

$$A\Delta x = infeas_c$$
$$A(\rho + \Theta^{-1}A^T\Delta y) = infeas_c$$

$$A\Theta^{-1}A^T\Delta y = infeas_c - A\rho$$
$$\Delta y = (A\Theta^{-1}A^T)^{-1}(infeas_c - A\rho)$$

Θ, ρ, Δy, Δx, and Δs are calculated in that order. The hardest term is the factorization of the $(A\Theta^{-1}A^T)$ matrix to determine Δy. Fortunately, although the *values* of $(A\Theta^{-1}A^T)$ are different for each iteration, the *locations* of the nonzeros in this matrix remain fixed; the nonzero locations are the same as those in the matrix (AA^T). This is because $\Theta^{-1} = XS^{-1}$ is a diagonal matrix that has the effect of merely scaling the columns of (AA^T).

The fact that the nonzeros in $A\Theta^{-1}A^T$ have a constant pattern is exploited by all Interior Point algorithms and is a major reason for their excellent performance. Before iterations begin, AA^T is examined and its rows and columns are symmetrically permutated so that during Cholesky factorization, the number of *fillins* created is smaller. A list of arithmetic operations to perform the factorization is saved in concise computer data structures (working with memory locations rather than actual numerical values). This is called *symbolic factorization*. During iterations, when memory has been initialized with numerical values, the operations list is performed sequentially. Determining how the factorization should be performed again and again is unnecessary.

The Primal-Dual Predictor-Corrector Interior Point Algorithm

The variant of the Interior Point algorithm implemented in PROC INTPOINT is a Primal-Dual Predictor-Corrector Interior Point algorithm. At first, Newton's method is used to find a direction $(\Delta x^k_{aff}, \Delta y^k_{aff}, \Delta s^k_{aff})$ to move, but calculated as if μ is zero, that is, as a step with no centering, known as an *affine* step:

$$A\Delta x^k_{aff} = infeas_c$$
$$A^T\Delta y^k_{aff} + \Delta s^k_{aff} = infeas_d$$
$$S^k\Delta x^k_{aff} + X^k\Delta s^k_{aff} = -X^kS^ke$$

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

α is the *step length* as before.

Complementarity x^Ts is calculated at $(x^k_{aff}, y^k_{aff}, s^k_{aff})$ and compared with the complementarity at the starting point (x^k, y^k, s^k), and the success of the affine step is gauged. If the affine step was successful in reducing the complementarity by a substantial amount, the need for centering is not great, and σ_k in the following linear system is assigned a value close to zero. If, however, the affine step was unsuccessful, centering would be beneficial, and σ_k in the following linear system is assigned a value closer to 1.0. The value of σ_k is therefore adaptively altered depending on the progress made toward the optimum.

A second linear system is solved to determine a centering vector $(\Delta x^k_c, \Delta y^k_c, \Delta s^k_c)$ from $(x^k_{aff}, y^k_{aff}, s^k_{aff})$:

$$A\Delta x^k_c = 0$$

$$A^T \Delta y_c^k + \Delta s_c^k = 0$$
$$S^k \Delta x_c^k + X^k \Delta s_c^k = -X_{aff}^k S_{aff}^k e + \sigma_k \mu_k e$$

then

$$(\Delta x^k, \Delta y^k, \Delta s^k) = (\Delta x_{aff}^k, \Delta y_{aff}^k, \Delta s_{aff}^k) + (\Delta x_c^k, \Delta y_c^k, \Delta s_c^k)$$

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

where, as before, α is the *step length* assigned a value as large as possible but not so large that a x_i^{k+1} or s_i^{k+1} is "too close" to zero.

Although the Predictor-Corrector variant entails solving two linear systems instead of one, fewer iterations are usually required to reach the optimum. The additional overhead of calculating the second linear system is small, as the factorization of the $(A\Theta^{-1}A^T)$ matrix has already been performed to solve the first linear system.

Interior Point: Upper Bounds

If the LP had upper bounds ($0 \leq x \leq u$ where u is the upper bound vector), then the primal and dual problems, the duality gap, and the KKT conditions would have to be expanded.

The primal Linear Program to be solved is

$$min\{c^T x\}$$
subject to $$Ax = b$$
$$0 \leq x \leq u$$

$0 \leq x \leq u$ is split into $x \geq 0$ and $x \leq u$. Let z be primal slack so that $x + z = u$, and associate dual variables w with these constraints. The Interior Point solves the system of equations to satisfy the Karush-Kuhn-Tucker (KKT) conditions for optimality:

$$Ax = b$$
$$x + z = u$$
$$A^T y + s - w = c$$
$$XSe = 0$$
$$ZWe = 0$$
$$x, s, z, w \geq 0$$

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

$$0 = z_{opt}^T w_{opt} = (u - x_{opt})^T w_{opt} = u^T w_{opt} - x_{opt}^T w_{opt}$$

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

therefore $0 = c^T x_{opt} - b^T y_{opt} + u^T w_{opt}$

Before the optimum is reached, a solution (x, y, s, z, w) might not satisfy the KKT conditions:

- Primal bound constraints can be broken, $infeas_b = u - x - z \neq 0$.
- Primal constraints can be broken, $infeas_c = b - Ax \neq 0$.
- Dual constraints can be broken, $infeas_d = c - A^T y - s + w \neq 0$.
- Complementarity conditions are unsatisfied, $x^T s \neq 0$ and $z^T w \neq 0$.

The calculations of the Interior point algorithm can easily be derived in a fashion similar to calculations for when an LP has no upper bounds. See the paper by Lustig, Marsten, and Shanno (1992).

In some iteration k, the *affine* step system that must be solved is:

$$\begin{aligned}
&\Delta x_{aff} + \Delta z_{aff} = infeas_b \\
&A\Delta x_{aff} = infeas_c \\
&A^T \Delta y_{aff} + \Delta s_{aff} - \Delta w_{aff} = infeas_d \\
&S\Delta x_{aff} + X\Delta s_{aff} = -XSe \\
&Z\Delta w_{aff} + W\Delta z_{aff} = -ZWe
\end{aligned}$$

Therefore, the computations involved in solving the affine step are:

$$\begin{aligned}
&\Theta = SX^{-1} + WZ^{-1} \\
&\rho = \Theta^{-1}(infeas_d + (S - W)e - Z^{-1}W infeas_b) \\
&\Delta y_{aff} = (A\Theta^{-1}A^T)^{-1}(infeas_c + A\rho) \\
&\Delta x_{aff} = \Theta^{-1}A^T \Delta y_{aff} - \rho \\
&\Delta z_{aff} = infeas_b - \Delta x_{aff} \\
&\Delta w_{aff} = -We - Z^{-1}W\Delta z_{aff} \\
&\Delta s_{aff} = -Se - X^{-1}S\Delta x_{aff}
\end{aligned}$$

$$\begin{aligned}
&(x_{aff}, y_{aff}, s_{aff}, z_{aff}, w_{aff}) = (x, y, s, z, w) + \\
&\alpha(\Delta x_{aff}, \Delta y_{aff}, \Delta s_{aff}, \Delta z_{aff}, \Delta w_{aff})
\end{aligned}$$

α is the *step length* as before.

A second linear system is solved to determine a centering vector: $(\Delta x_c, \Delta y_c, \Delta s_c, \Delta z_c, \Delta w_c)$ from $(x_{aff}, y_{aff}, s_{aff}, z_{aff}, w_{aff})$:

$$\begin{aligned}
&\Delta x_c + \Delta z_c = 0\\
&A\Delta x_c = 0\\
&A^T\Delta y_c + \Delta s_c - \Delta w_c = 0\\
&S\Delta x_c + X\Delta s_c = -X_{aff}S_{aff}e + \sigma\mu e\\
&Z\Delta w_c + W\Delta z_c = -Z_{aff}W_{aff}e + \sigma\mu e
\end{aligned}$$

where

$\zeta_{start} = x^T s + z^T w$, complementarity at the start of the iteration

$\zeta_{aff} = x_{aff}^T s_{aff} + z_{aff}^T w_{aff}$, the affine complementarity

$\mu = \zeta_{aff}/2n$, the average complementarity

$\sigma = (\zeta_{aff}/\zeta_{start})^3$

Therefore, the computations involved in solving the centering step are:

$$\begin{aligned}
&\rho = \Theta^{-1}(\sigma\mu(X^{-1} - Z^{-1})e - X^{-1}X_{aff}S_{aff}e + Z^{-1}Z_{aff}W_{aff}e)\\
&\Delta y_c = (A\Theta^{-1}A^T)^{-1}A\rho\\
&\Delta x_c = \Theta^{-1}A^T\Delta y_c - \rho\\
&\Delta z_c = -\Delta x_c\\
&\Delta w_c = \sigma\mu Z^{-1}e - Z^{-1}Z_{aff}W_{aff}e - Z^{-1}W_{aff}\Delta z_c\\
&\Delta s_c = \sigma\mu X^{-1}e - X^{-1}X_{aff}S_{aff}e - X^{-1}S_{aff}\Delta x_c
\end{aligned}$$

Then

$$\begin{aligned}
&(\Delta x, \Delta y, \Delta s, \Delta z, \Delta w) =\\
&(\Delta x_{aff}, \Delta y_{aff}, \Delta s_{aff}, \Delta z_{aff}, \Delta w_{aff}) +\\
&(\Delta x_c, \Delta y_c, \Delta s_c, \Delta z_c, \Delta w_c)
\end{aligned}$$

$$\begin{aligned}
&(x^{k+1}, y^{k+1}, s^{k+1}, z^{k+1}, w^{k+1}) =\\
&(x^k, y^k, s^k, z^k, w^k) +\\
&\alpha(\Delta x, \Delta y, \Delta s, \Delta z, \Delta w)
\end{aligned}$$

where, as before, α is the *step length* assigned a value as large as possible but not so large that a x_i^{k+1}, s_i^{k+1}, z_i^{k+1}, or w_i^{k+1} is "too close" to zero.

The algebra in this section has been simplified by assuming that *all* variables have finite upper bounds. If the number of variables with finite upper bounds $n_u < n$, you need to change the algebra to reflect that the "z" and "w" arrays has dimension $n_u x1$ or $n_u x n_u$. Other computations need slight modification. For example, the average complementarity

$$\mu = x_{aff}^T s_{aff}/n + z_{aff}^T w_{aff}/n_u$$

An important point is that any upper bounds can be handled by specializing the algorithm and *not* by generating the constraints $x \leq u$ and adding these to the main primal constraints $Ax = b$.

Network Models

The following are descriptions of some typical NPSC models.

Production, Inventory, and Distribution (Supply Chain) Problems

One common class of network model is the production-inventory-distribution or supply-chain problem. The diagram in Figure 4.1 illustrates this problem. The subscript on the Production, Inventory, and Sales nodes indicates the time period. By replicating sections of the model, the notion of time can be included.

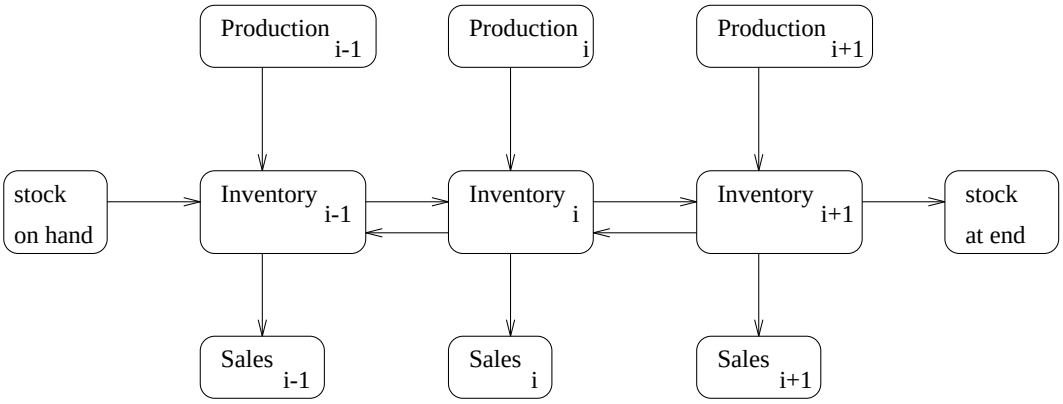


Figure 4.1. Production-Inventory-Distribution Problem

In this type of model, the nodes can represent a wide variety of facilities. Several examples are suppliers, spot markets, importers, farmers, manufacturers, factories, parts of a plant, production lines, waste disposal facilities, workstations, warehouses, coolstores, depots, wholesalers, export markets, ports, rail junctions, airports, road intersections, cities, regions, shops, customers, and consumers. The diversity of this selection demonstrates how rich the potential applications of this model are.

Depending upon the interpretation of the nodes, the objectives of the modeling exercise can vary widely. Some common types of objectives are

- to reduce collection or purchase costs of raw materials
- to reduce inventory holding or backorder costs. Warehouses and other storage facilities sometimes have capacities, and there can be limits on the amount of goods that can be placed on backorder.
- to decide where facilities should be located and what the capacity of these should be. Network models have been used to help decide where factories, hospitals, ambulance and fire stations, oil and water wells, and schools should be sited.

- to determine the assignment of resources (machines, production capability, workforce) to tasks, schedules, classes, or files
- to determine the optimal distribution of goods or services. This usually means minimizing transportation costs and reducing transit time or distances covered.
- to find the shortest path from one location to another
- to ensure that demands (for example, production requirements, market demands, contractual obligations) are met
- to maximize profits from the sale of products or the charge for services
- to maximize production by identifying bottlenecks

Some specific applications are

- car distribution models. These help determine which models and numbers of cars should be manufactured in which factories and where to distribute cars from these factories to zones in the United States in order to meet customer demand at least cost.
- models in the timber industry. These help determine when to plant and mill forests, schedule production of pulp, paper, and wood products, and distribute products for sale or export.
- military applications. The nodes can be theaters, bases, ammunition dumps, logistical suppliers, or radar installations. Some models are used to find the best ways to mobilize personnel and supplies and to evacuate the wounded in the least amount of time.
- communications applications. The nodes can be telephone exchanges, transmission lines, satellite links, and consumers. In a model of an electrical grid, the nodes can be transformers, powerstations, watersheds, reservoirs, dams, and consumers. The effect of high loads or outages might be of concern.

Proportional Constraints

In many models, you have the characteristic that a flow through an arc must be proportional to the flow through another arc. Side constraints are often necessary to model that situation. Such constraints are called *proportional constraints* and are useful in models where production is subject to refining or modification into different materials. The amount of each output, or any waste, evaporation, or reduction can be specified as a proportion of input.

Typically, the arcs near the supply nodes carry raw materials and the arcs near the demand nodes carry refined products. For example, in a model of the milling industry, the flow through some arcs may represent quantities of wheat. After the wheat is processed, the flow through other arcs might be flour. For others it might be bran. The side constraints model the relationship between the amount of flour or bran produced as a proportion of the amount of wheat milled. Some of the wheat can end up as neither flour, bran, nor any useful product, so this waste is drained away via arcs to a waste node.

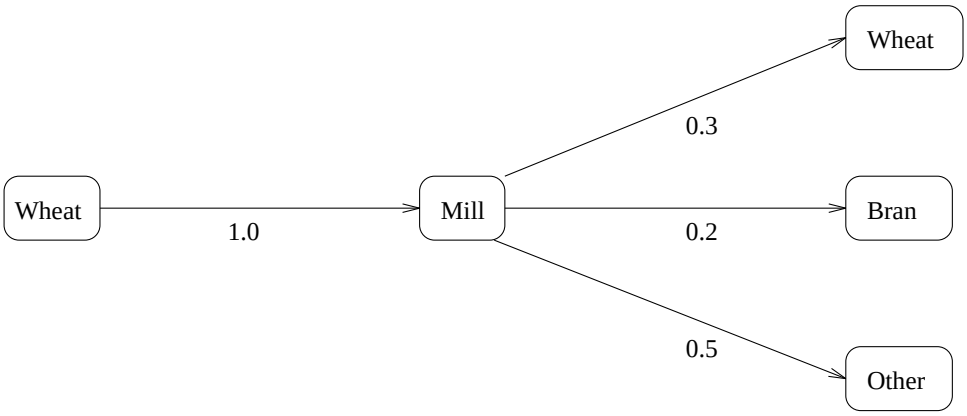


Figure 4.2. Proportional Constraints

In order for arcs to be specified in side constraints, they must be named. By default, PROC INTPOINT names arcs using the names of the nodes at the head and tail of the arc. An arc is named with its tail node name followed by an "_" followed by the name of its head node name. For example, an arc from node *from* to node *to* is called *from_to*.

Consider the network fragment in Figure 4.2. The arc Wheat_Mill conveys the wheat milled. The cost of flow on this arc is the milling cost. The capacity of this arc is the capacity of the mill. The lower flow bound on this arc is the minimum quantity that must be milled for the mill to operate economically. The constraints

$$0.3\ \text{Wheat\_Mill} - \text{Mill\_Flour} = 0.0$$
$$0.2\ \text{Wheat\_Mill} - \text{Mill\_Bran} = 0.0$$

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

$$0.5\ \text{Wheat\_Mill} - \text{Mill\_Other} = 0.0$$

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

Blending Constraints

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

The network fragment in Figure 4.3 shows an example of this.

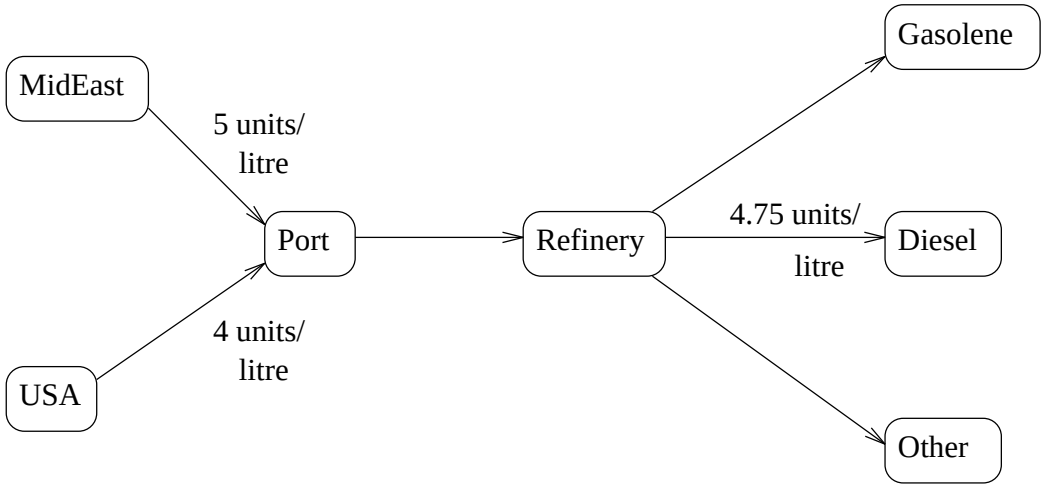


Figure 4.3. Blending Constraints

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

$$0.4\ \text{Port\_Refinery} - \text{Refinery\_Gasoline} = 0.0$$
$$0.2\ \text{Port\_Refinery} - \text{Refinery\_Diesel} = 0.0$$

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

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

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

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

If, for example, 120 units of crude from the Middle East is used, then at least 40 units of crude from the U.S.A. must be used. The preceding constraint is simplified

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

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

is used to obtain

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

which equals

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

An example similar to this oil industry problem is solved in the "Introductory NPSC Example" section on page 75.

Multicommodity Problems

Side constraints are also used in models in which there are capacities on transportation or some other shared resource, or there are limits on overall production or demand in multicommodity, multidivisional, or multiperiod problems. Each commodity, division, or period can have a separate network coupled to one main system by the side constraints. Side constraints are used to combine the outputs of subdivisions of a problem (either commodities, outputs in distinct time periods, or different process streams) to meet overall demands or to limit overall production or expenditures. This method is more desirable than doing separate *local* optimizations for individual commodity, process, or time networks and then trying to establish relationships between each when determining an overall policy if the *global* constraint is not satisfied. Of course, to make models more realistic, side constraints may be necessary in the local problems.

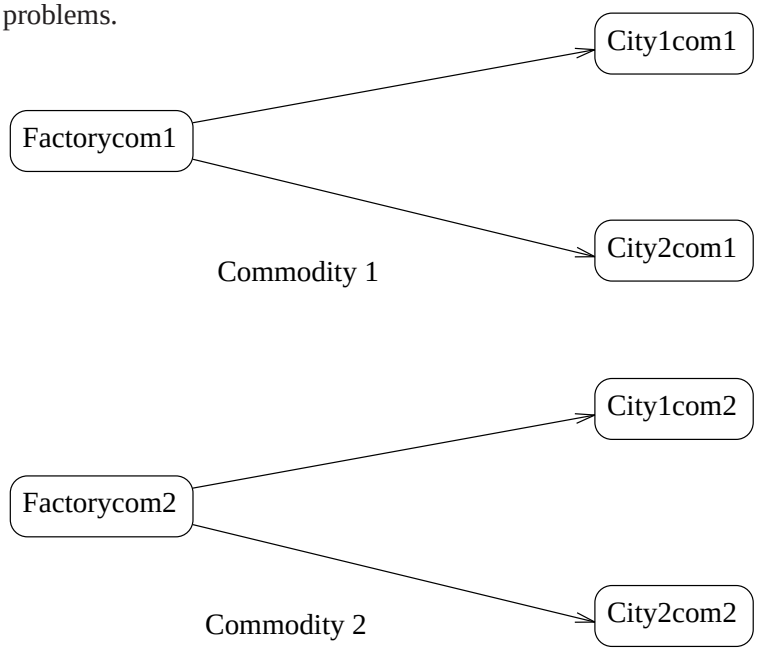


Figure 4.4. Multicommodity Problem

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

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

Large Modeling Strategy

In many cases, the flow through an arc might actually represent the flow or movement of a commodity from place to place or from time period to time period. However, sometimes an arc is included in the network as a method of capturing some aspect of the problem that you would not normally think of as part of a network model. There is no commodity movement associated with that arc. For example, in a multiprocess, multiproduct model (Figure 4.5), there might be subnetworks for each process and each product. The subnetworks can be joined together by a set of arcs that have flows that represent the amount of product j produced by process i. To model an upper-limit constraint on the total amount of product j that can be produced, direct all arcs carrying product j to a single node and from there through a single arc. The capacity of this arc is the upper limit of product j production. It is preferable to model this structure in the network rather than to include it in the side constraints because the efficiency of the optimizer may be less affected by a reasonable increase in the size of the network rather than increasing the number or complicating side constraints.

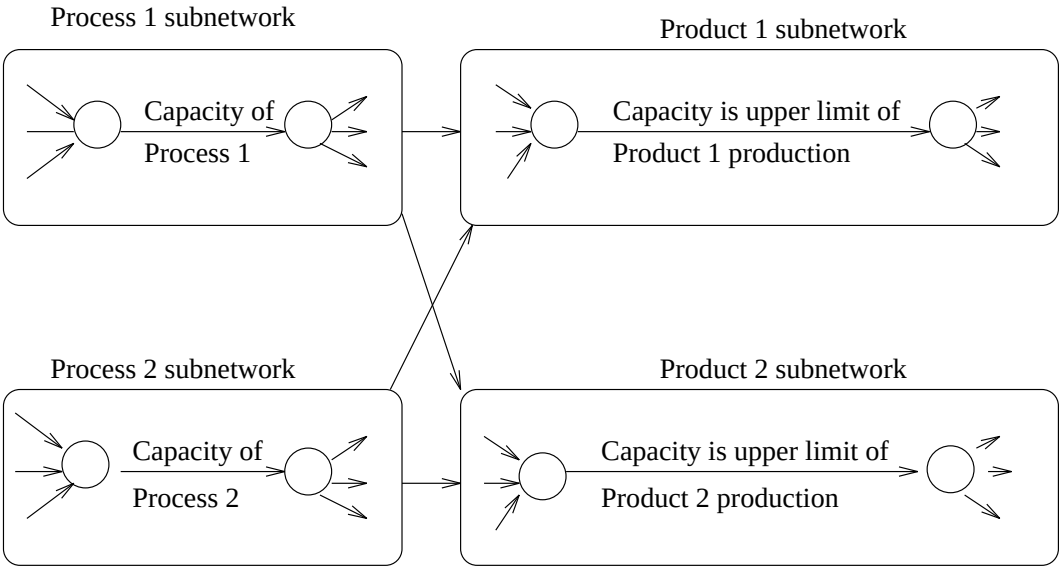


Figure 4.5. Multiprocess, Multiproduct Example

When starting a project, it is often a good strategy to use a small network formulation and then to use that model as a framework upon which to add detail. For example, in the multiprocess multiproduct model, you might start with the network depicted in Figure 4.5. Then, for example, the process subnetwork can be enhanced to include the distribution of products. Other phases of the operation could be included by adding more subnetworks. Initially, these subnetworks can be single nodes, but in subsequent studies they can be expanded to include greater detail.

Advantages of Network Models over LP Models

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

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

If a network structure is embedded in a linear program, the problem is an NPSC (see the "Mathematical Description of NPSC" section on page 56). When the network part of the problem is large compared to the non-network part, especially if the number of side constraints is small, it is worthwhile to exploit this structure to describe the model. Rather than generating the data for the flow conservation constraints, generate instead the data for the nodes and arcs of the network.

Flow Conservation Constraints

The constraints $Fx = b$ in NPSC (see the "Mathematical Description of NPSC" section on page 56) are referred to as the nodal flow conservation constraints. These constraints algebraically state that the sum of the flow through arcs directed toward a node plus that node's supply, if any, equals the sum of the flow through arcs directed away from that node plus that node's demand, if any. The flow conservation constraints are implicit in the network model and should not be specified explicitly in side constraint data when using PROC INTPOINT to solve NPSC problems.

Nonarc Variables

Nonarc variables can be used to simplify side constraints. For example, if a sum of flows appears in many constraints, it may be worthwhile to equate this expression with a nonarc variable and use this in the other constraints. This keeps the constraint coefficient matrix sparse. By assigning a nonarc variable a nonzero objective function, it is then possible to incur a cost for using resources above some lowest feasible limit. Similarly, a profit (a negative objective function coefficient value) can be made if all available resources are not used.

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

For example, consider a multicommodity problem of making television sets that have either 19- or 25-inch screens. In their manufacture, three and four chips, respectively, are used. Production occurs at two factories during March and April. The supplier of chips can supply only 2,600 chips to factory 1 and 3,750 chips to factory 2 each month. The names of arcs are in the form Prod*n_s_m*, where *n* is the factory number, *s* is the screen size, and *m* is the month. For example, Prod1_25_Apr is the arc that conveys the number of 25-inch TVs produced in factory 1 during April. You might have to determine similar systematic naming schemes for your application.

As described, the constraints are

$$\begin{aligned} 3\,\text{Prod1\_19\_Mar} + 4\,\text{Prod1\_25\_Mar} &\leq 2600 \\ 3\,\text{Prod2\_19\_Mar} + 4\,\text{Prod2\_25\_Mar} &\leq 3750 \\ 3\,\text{Prod1\_19\_Apr} + 4\,\text{Prod1\_25\_Apr} &\leq 2600 \\ 3\,\text{Prod2\_19\_Apr} + 4\,\text{Prod2\_25\_Apr} &\leq 3750 \end{aligned}$$

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

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

where F1_Kept_Since_Mar is the number of chips used during April at factory 1 that were obtained in March at either factory 1 or factory 2, and F2_Kept_Since_Mar is the number of chips used during April at factory 2 that were obtained in March. The last constraint ensures that the number of chips used during April that were obtained in March does not exceed the number of chips not used in March. There may be a cost to hold chips in inventory. This can be modeled having a positive objective function coefficient for the nonarc variables F1_Kept_Since_Mar and F2_Kept_Since_Mar. Moreover, nonarc variable upper bounds represent an upper limit on the number of chips that can be held in inventory between March and April.

See Example 4.1 through Example 4.5, which use this TV problem. The use of nonarc variables as described previously is illustrated.

Introduction

Getting Started: NPSC Problems

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

The NODEDATA= data set contains the names of the supply and demand nodes and the supply or demand associated with each. These are the elements in the column vector b in the NPSC problem (see the "Mathematical Description of NPSC" section on page 56).

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

An arc is identified by the names of its tail node (where it originates) and head node (where it is directed). Each observation can be used to identify an arc in the network and, optionally, the cost per flow unit across the arc, the arc's capacity, lower flow bound, and name. These data are associated with the matrix F and the vectors c, l, and u in the NPSC problem (see the "Mathematical Description of NPSC" section on page 56).

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

In addition, the ARCDATA= data set can be used to specify information about nonarc variables, including objective function coefficients, lower and upper value bounds, and names. These data are the elements of the vectors d, m, and v in the NPSC problem (see the "Mathematical Description of NPSC" section on page 56). Data for an arc or nonarc variable can be given in more than one observation.

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

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

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

If you use the dense side constraint input format (described in the "CONDATA= Data Set" section on page 126), and want to use these default arc names, these arc names are names of SAS variables in the VAR list of the CONDATA= data set.

If you use the sparse side constraint input format (see the "CONDATA= Data Set" section on page 126) and want to use these default arc names, these arc names are values of the COLUMN list variable of the CONDATA= data set.

PROC INTPOINT reads the data from the NODEDATA= data set, the ARCDATA= data set, and the CONDATA= data set. Error checking is performed, and the model is converted into an equivalent LP. This LP is preprocessed. Preprocessing is optional but highly recommended. Preprocessing analyzes the model and tries to determine before optimization whether variables can be "fixed" to their optimal values. Knowing that, the model can be modified and these variables dropped out. It can be determined that some constraints are redundant. Sometimes, preprocessing succeeds in reducing the size of the problem, thereby making the subsequent optimization easier and faster.

The optimal solution to the equivalent LP is then found. This LP is converted back to the original NPSC problem, and the optimum for this is derived from the optimum of the equivalent LP. If the problem was preprocessed, the model is now post-processed, where fixed variables are reintroduced. The solution can be saved in the CONOUT= data set.

Introductory NPSC Example

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

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

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

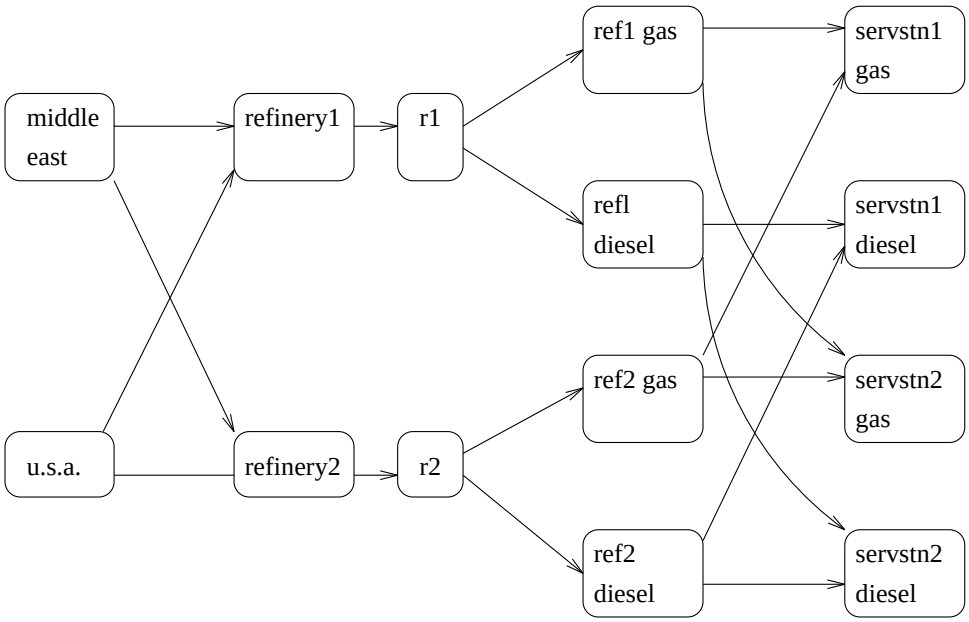


Figure 4.6. Oil Industry Example

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

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

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

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

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

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

Note that the SAS variable names in the CONDATA= data set are the names of arcs given in the ARCDATA= data set. These are the arcs that have nonzero constraint coefficients in side constraints. For example, the proportionality constraint that specifies that one unit of crude at each refinery yields three-fourths of a unit of gasoline and one-fourth of a unit of diesel fuel is given for refinery 1 in the third observation and for refinery 2 in the last observation. The third observation requires that each unit of flow on the arc thruput1 equals three-fourths of a unit of flow on the arc r1_gas. Because all crude processed at refinery 1 flows through thruput1 and all gasoline produced at refinery 1 flows through r1_gas, the constraint models the situation. It proceeds similarly for refinery 2 in the last observation.

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

```
proc intpoint
   bytes=1000000
   nodedata=noded        /* the supply and demand data */
   arcdata=arcd1         /* the arc descriptions       */
   condata=cond1         /* the side constraints       */
   conout=solution;      /* the solution data set      */
run;
```

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

```
NOTE: Number of nodes= 14 .
NOTE: Number of supply nodes= 2 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 180 , total demand= 180 .
NOTE: Number of arcs= 18 .
NOTE: Number of <= side constraints= 0 .
NOTE: Number of == side constraints= 2 .
NOTE: Number of >= side constraints= 2 .
NOTE: Number of side constraint coefficients= 8 .
NOTE: The following messages relate to the equivalent
      Linear Program solved by the Interior Point algorithm.
NOTE: Number of <= constraints= 0 .
NOTE: Number of == constraints= 16 .
NOTE: Number of >= constraints= 2 .
NOTE: Number of constraint coefficients= 44 .
NOTE: Number of variables= 18 .
NOTE: After preprocessing, number of <= constraints= 0.
NOTE: After preprocessing, number of == constraints= 6.
NOTE: After preprocessing, number of >= constraints= 2.
NOTE: The preprocessor eliminated 10 constraints from the
      problem.
NOTE: The preprocessor eliminated 22 constraint
      coefficients from the problem.
NOTE: After preprocessing, number of variables= 8.
NOTE: The preprocessor eliminated 10 variables from the
      problem.
NOTE: 2 columns, 0 rows and 2 coefficients were added to
      the problem to handle unrestricted variables,
      variables that are split, and constraint slack or
      surplus variables.
NOTE: There are 18 nonzero elements in A * A transpose.
NOTE: Of the 8 rows and columns, 2 are sparse.
NOTE: There are 8 nonzero superdiagonal elements in the
      sparse rows of the factored A * A transpose. This
      includes fill-in.
NOTE: There are 3 operations of the form
      u[i,j]=u[i,j]-u[q,j]*u[q,i]/u[q,q] to factorize the
      sparse rows of A * A transpose.
NOTE: Bound feasibility attained by iteration 1.
```

```
NOTE: Dual feasibility attained by iteration 1.
NOTE: Constraint feasibility attained by iteration 2.
NOTE: Primal-Dual Predictor-Corrector Interior point
      algorithm performed 12 iterations.
NOTE: The rate of complementarity reduction is slow. The
      optimum has probably been reached even though
      standard stopping conditions have not been met.
NOTE: Objective = 50875.006369.
NOTE: The data set WORK.SOLUTION has 18 observations and
      14 variables.
NOTE: There were 18 observations read from the data set
      WORK.ARCD1.
NOTE: There were 6 observations read from the data set
      WORK.NODED.
NOTE: There were 4 observations read from the data set
      WORK.COND1.
NOTE: The data set WORK.SOLUTION has 18 observations and
      14 variables.
```

The first set of messages shows the size of the problem. The next set of messages provides statistics on the size of the equivalent LP problem. The number of variables may not equal the number of arcs if the problem has nonarc variables. This example has none. To convert a network to the equivalent LP problem, a flow conservation constraint must be created for each node (including an excess or bypass node, if required). This explains why the number of equality constraints and the number of constraint coefficients differ from the number of equality side constraints and the number of coefficients in all side constraints.

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

The next set of messages describes aspects of the Interior Point algorithm. Of particular interest are those concerned with the Cholesky factorization of AA^T where A is the coefficient matrix of the final LP. It is crucial to preorder the rows and columns of this matrix to prevent *fill-in* and reduce the number of row operations to undertake the factorization. See the "Interior Point Algorithmic Details" section on page 58 for a more extensive explanation.

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

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

Oil Industry Example

Optimum

Obs	_from_	_to_	_cost_	_capa_	_low_	_name_	_SUPPLY_	_DEMAND_	_FLOW_	_FCOST_
1	refinery 1	r1	200	175	50	thruput1	.	.	145.000	28999.99
2	refinery 2	r2	220	100	35	thruput2	.	.	35.000	7700.01
3	r1	ref1 diesel	0	75	0		.	.	36.250	0.00
4	r1	ref1 gas	0	140	0	r1_gas	.	.	108.750	0.00
5	r2	ref2 diesel	0	75	0		.	.	8.750	0.00
6	r2	ref2 gas	0	100	0	r2_gas	.	.	26.250	0.00
7	middle east	refinery 1	63	95	20	m_e_ref1	100	.	80.000	5039.99
8	u.s.a.	refinery 1	55	99999999	0		80	.	65.000	3575.00
9	middle east	refinery 2	81	80	10	m_e_ref2	100	.	20.000	1620.01
10	u.s.a.	refinery 2	49	99999999	0		80	.	15.000	735.00
11	ref1 diesel	servstn1 diesel	18	99999999	0		.	30	30.000	540.00
12	ref2 diesel	servstn1 diesel	36	99999999	0		.	30	0.000	0.00
13	ref1 gas	servstn1 gas	15	70	0		.	95	68.750	1031.25
14	ref2 gas	servstn1 gas	17	35	5		.	95	26.250	446.25
15	ref1 diesel	servstn2 diesel	17	99999999	0		.	15	6.250	106.25
16	ref2 diesel	servstn2 diesel	23	99999999	0		.	15	8.750	201.25
17	ref1 gas	servstn2 gas	22	60	0		.	40	40.000	879.99
18	ref2 gas	servstn2 gas	31	99999999	0		.	40	0.000	0.01
										50875.01

Figure 4.7. CONOUT=SOLUTION

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

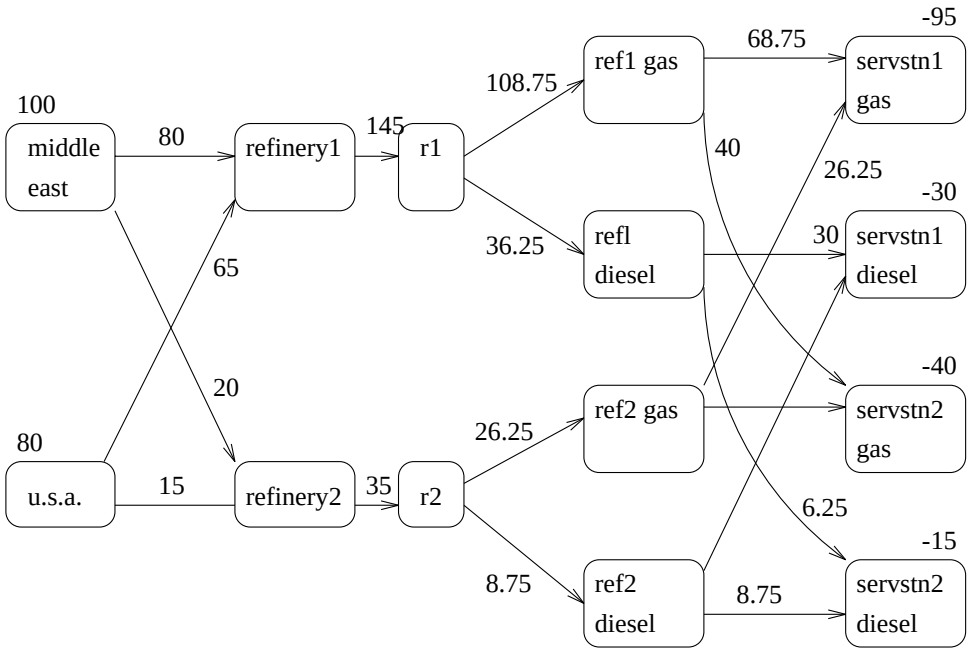


Figure 4.8. Oil Industry Solution

Getting Started: LP Problems

Data for an LP problem resembles the data for side constraints and nonarc variables supplied to PROC INTPOINT when solving an NPSC problem. It is also very similar to the data required by the LP procedure.

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

The ARCDATA= data set contains information about the LP variables of the problem. Although this data set is called ARCDATA, it contains data for no arcs. Instead, all data in this data set are related to LP variables. This data set has no SAS variables containing values that are node names.

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

The CONDATA= data set describes the constraints and their right-hand-sides. These data are elements of the matrix Q and the vector r.

Constraint types are also specified in the CONDATA= data set. You can include in this data set LP variable data such as upper bound values, lower value bounds, and

objective function coefficients. It is possible to give all information about some or all LP variables in the CONDATA= data set.

Because PROC INTPOINT evolved from PROC NETFLOW, another procedure in SAS/OR software that was originally designed to solve models with networks, the ARCDATA= data set is always expected. If the ARCDATA= data set is not specified, by default the last data set created before PROC INTPOINT is invoked is assumed to be the ARCDATA= data set. However, these characteristics of PROC INTPOINT are not helpful when an LP problem is being solved and all data is provided in a single data set specified by the CONDATA= data set, and that data set is not the last data set created before PROC INTPOINT starts. In this case, you must specify that the ARCDATA= data set and the CONDATA= data set are both equal to the input data set. PROC INTPOINT then knows that an LP problem is to be solved and that the data reside in one data set.

An LP variable is identified in this data set by its name. If you specify an LP variable's name in the ARCDATA= data set, then this name is used to associate data in the CONDATA= data set with that LP variable.

If you use the dense constraint input format (described in the "CONDATA= Data Set" section on page 126), these LP variable names are names of SAS variables in the VAR list of the CONDATA= data set.

If you use the sparse constraint input format (described in the "CONDATA= Data Set" section on page 126), these LP variable names are values of the SAS variables in the COLUMN list of the CONDATA= data set.

PROC INTPOINT reads the data from the ARCDATA= data set (if there is one) and the CONDATA= data set. Error checking is performed, and the LP is preprocessed. Preprocessing is optional but highly recommended. The preprocessor analyzes the model and tries to determine before optimization whether LP variables can be "fixed" to their optimal values. Knowing that, the model can be modified and these LP variables dropped out. Some constraints may be found to be redundant. Sometimes, preprocessing succeeds in reducing the size of the problem, thereby making the subsequent optimization easier and faster.

The optimal solution is then found for the resulting LP. If the problem was preprocessed, the model is now post-processed, where fixed LP variables are reintroduced. The solution can be saved in the CONOUT= data set.

Introductory LP Example

Consider the Linear Programming problem in the chapter on the LP procedure. The SAS data set in that section is created the same way here:

```
title  'Linear Programming Example';
title3 'Setting Up Condata = dcon1 For PROC INTPOINT';
   data dcon1;
      input _id_ $14.
            a_light a_heavy brega naphthal naphthai
            heatingo jet_1 jet_2
            _type_ $ _rhs_;
      datalines;
   profit         -175 -165 -205  0  0  0 300 300 max     .
   naphtha_l_conv .035 .030 .045 -1  0  0   0   0  eq     0
   naphtha_i_conv .100 .075 .135  0 -1  0   0   0  eq     0
   heating_o_conv .390 .300 .430  0  0 -1   0   0  eq     0
   recipe_1         0    0    0  0 .3 .7  -1   0  eq     0
   recipe_2         0    0    0 .2  0 .8   0  -1  eq     0
   available      110  165   80  .  .  .   .   . upperbd .
   ;
```

To solve this problem, use:

```
   proc intpoint
      bytes=1000000
      condata=dcon1
      conout=solutn1;
   run;
```

Note how it is possible to use an input SAS data set of PROC LP and, without requiring any changes to be made to the data set, to use that as an input data set for PROC INTPOINT.

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

```
   NOTE: Number of variables= 8 .
   NOTE: Number of <= constraints= 0 .
   NOTE: Number of == constraints= 5 .
   NOTE: Number of >= constraints= 0 .
   NOTE: Number of constraint coefficients= 18 .
   NOTE: After preprocessing, number of <= constraints= 0.
   NOTE: After preprocessing, number of == constraints= 0.
   NOTE: After preprocessing, number of >= constraints= 0.
   NOTE: The preprocessor eliminated 5 constraints from the
         problem.
   NOTE: The preprocessor eliminated 18 constraint
         coefficients from the problem.
   NOTE: After preprocessing, number of variables= 0.
   NOTE: The preprocessor eliminated 8 variables from the
         problem.
   WARNING: Optimization is unnecessary as the problem has 0
            variables and 0 rows.
```

```
NOTE: Preprocessing could have caused that.
NOTE: Objective = 1544.
NOTE: The data set WORK.SOLUTN1 has 8 observations and 6
      variables.
NOTE: There were 7 observations read from the data set
      WORK.DCON1.
NOTE: The data set WORK.SOLUTN1 has 8 observations and 6
      variables.
```

Notice that the preprocessor succeeded in fixing *all* LP variables to their optimal values, eliminating the need to do any actual optimization.

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

```
title3 'LP Optimum';
proc print data=solutn1;
   var _name_ _cost_ _capac_ _lo_ _flow_ _fcost_;
   sum _fcost_;
run;
```

Notice that in the CONOUT=SOLUTION (Figure 4.9) the optimal value through each variable in the LP is given in the variable named _FLOW_, and that the cost of value for each variable is given in the variable _FCOST_.

Linear Programming Example

LP Optimum

Obs	_NAME_	_COST_	_CAPAC_	_LO_	_FLOW_	_FCOST_
1	a_heavy	-165	165	0	0.00	0
2	a_light	-175	110	0	110.00	-19250
3	brega	-205	80	0	80.00	-16400
4	heatingo	0	99999999	0	77.30	0
5	jet_1	300	99999999	0	60.65	18195
6	jet_2	300	99999999	0	63.33	18999
7	naphthai	0	99999999	0	21.80	0
8	naphthal	0	99999999	0	7.45	0
						=======
						1544

Figure 4.9. CONOUT=SOLUTN1

The same model can be specified in the sparse format as in the following scon2 dataset. This format enables you to omit the zero coefficients:

```
title3 'Setting Up Condata = scon2 For PROC INTPOINT';
data scon2;
   input _type_ $ @10 _col_ $13. @24 _row_ $16. _coef_;
   datalines;
max       .             profit                  .
eq        .             napha_l_conv            .
eq        .             napha_i_conv            .
eq        .             heating_oil_conv        .
eq        .             recipe_1                .
eq        .             recipe_2                .
upperbd   .             available               .
.         a_light       profit                -175
.         a_light       napha_l_conv          .035
.         a_light       napha_i_conv          .100
.         a_light       heating_oil_conv      .390
.         a_light       available              110
.         a_heavy       profit                -165
.         a_heavy       napha_l_conv          .030
.         a_heavy       napha_i_conv          .075
.         a_heavy       heating_oil_conv      .300
.         a_heavy       available              165
.         brega         profit                -205
.         brega         napha_l_conv          .045
.         brega         napha_i_conv          .135
.         brega         heating_oil_conv      .430
.         brega         available               80
.         naphthal      napha_l_conv           -1
.         naphthal      recipe_2               .2
.         naphthai      napha_i_conv           -1
.         naphthai      recipe_1               .3
.         heatingo      heating_oil_conv       -1
.         heatingo      recipe_1               .7
.         heatingo      recipe_2               .8
.         jet_1         profit                300
.         jet_1         recipe_1               -1
.         jet_2         profit                300
.         jet_2         recipe_2               -1
;
```

To find the minimum cost solution, invoke PROC INTPOINT (note the SPARSECONDATA option which must be specified) as follows:

```
proc intpoint
   bytes=1000000
   sparsecondata
   condata=scon2
   conout=solutn2;
run;
```

A data set that can be used as the ARCDATA= data set can be initialized as follows:

```
data vars3;
   input _name_ $ profit available;
   datalines;
a_heavy  -165 165
a_light  -175 110
brega    -205  80
heatingo    0   .
jet_1     300   .
jet_2     300   .
naphthai    0   .
naphthal    0   .
;
```

The following CONDATA= data set is the original dense format CONDATA= dcon1 data set after the LP variable's nonconstraint information has been removed. (You could have left some or all of that information in CONDATA as PROC INTPOINT "merges" data, but doing that and checking for consistency takes time.)

```
data dcon3;
   input _id_ $14.
         a_light a_heavy brega naphthal naphthai
         heatingo jet_1 jet_2
         _type_ $ _rhs_;
   datalines;
naphtha_l_conv .035 .030 .045 -1  0  0   0   0  eq     0
naphtha_i_conv .100 .075 .135  0 -1  0   0   0  eq     0
heating_o_conv .390 .300 .430  0  0 -1   0   0  eq     0
recipe_1          0    0    0  0 .3 .7  -1   0  eq     0
recipe_2          0    0    0 .2  0 .8   0  -1  eq     0
;
```

Note: You must now specify the MAXIMIZE option; otherwise, PROC INTPOINT will optimize to the minimum (which, incidentally, has a total objective = -3539.25). You must indicate that the SAS variable profit in the ARCDATA= vars3 data set has values that are objective function coefficients, by specifying the OBJFN statement. The UPPERBD must be specified as the SAS variable available that has as values upper bounds:

```
proc intpoint
   maximize             /* ***** necessary ***** */
   bytes=1000000
   arcdata=vars3
   condata=dcon3
   conout=solutn3;
objfn profit;
upperbd available;
run;
```

The ARCDATA=vars3 data set can become more concise by noting that the model variables heatingo, naphthai, and naphthal have zero objective function coeffi-

cients (the default) and default upper bounds, so those observations need not be present:

```
data vars4;
   input _name_ $ profit available;
   datalines;
a_heavy  -165 165
a_light  -175 110
brega    -205  80
jet_1     300   .
jet_2     300   .
;
```

The CONDATA=dcon3 data set can become more concise by noting that all the constraints have the same type (eq) and zero (the default) rhs values. This model is a good candidate for using the DEFCONTYPE= options.

The DEFCONTYPE= option can be useful not only when *all* constraints have the same type as is the case here, but also when *most* constraints have the same type and you want to change the default type from $\leq$ to = or $\geq$. The essential constraint type data in the CONDATA= data set is that which overrides the DEFCONTYPE= type you specified:

```
data dcon4;
   input _id_ $14.
         a_light a_heavy brega naphthal naphthai
         heatingo jet_1 jet_2;
   datalines;
naphtha_l_conv .035 .030 .045 -1  0  0   0   0
naphtha_i_conv .100 .075 .135  0 -1  0   0   0
heating_o_conv .390 .300 .430  0  0 -1   0   0
recipe_1          0    0    0  0 .3 .7  -1   0
recipe_2          0    0    0 .2  0 .8   0  -1
;

proc intpoint
   maximize defcontype=eq
   arcdata=vars3
   condata=dcon3
   conout=solutn3;
objfn profit;
upperbd available;
run;
```

Here are several different ways of using the ARCDATA= data set and a sparse format CONDATA= data set for this LP. The following CONDATA= data set is the result of removing the profit and available data from the original sparse format CONDATA=scon2 data set.

```
data scon5;
   input _type_ $ @10 _col_ $13. @24 _row_ $16. _coef_;
   datalines;
eq       .            napha_l_conv             .
eq       .            napha_i_conv             .
eq       .            heating_oil_conv         .
eq       .            recipe_1                 .
eq       .            recipe_2                 .
.        a_light      napha_l_conv          .035
.        a_light      napha_i_conv          .100
.        a_light      heating_oil_conv      .390
.        a_heavy      napha_l_conv          .030
.        a_heavy      napha_i_conv          .075
.        a_heavy      heating_oil_conv      .300
.        brega        napha_l_conv          .045
.        brega        napha_i_conv          .135
.        brega        heating_oil_conv      .430
.        naphthal     napha_l_conv            -1
.        naphthal     recipe_2                .2
.        naphthai     napha_i_conv            -1
.        naphthai     recipe_1                .3
.        heatingo     heating_oil_conv        -1
.        heatingo     recipe_1                .7
.        heatingo     recipe_2                .8
.        jet_1        recipe_1                -1
.        jet_2        recipe_2                -1
;

proc intpoint
   maximize
   sparsecondata
   arcdata=vars3        /* or arcdata=vars4 */
   condata=scon5
   conout=solutn5;
objfn profit;
upperbd available;
run;
```

The CONDATA=scon5 data set can become more concise by noting that all the constraints have the same type (eq) and zero (the default) rhs values. Use the DEFCONTYPE= option again. Once the first five observations of the CONDATA=scon5 data set are removed, the _type_ SAS variable has values that are missing in all of the remaining observations. Therefore, this SAS variable can be removed.

```
data scon6;
   input _col_ $ _row_&$16. _coef_;
   datalines;
a_light  napha_l_conv          .035
a_light  napha_i_conv          .100
a_light  heating_oil_conv      .390
a_heavy  napha_l_conv          .030
a_heavy  napha_i_conv          .075
a_heavy  heating_oil_conv      .300
brega    napha_l_conv          .045
brega    napha_i_conv          .135
brega    heating_oil_conv      .430
naphthal napha_l_conv           -1
naphthal recipe_2               .2
naphthai napha_i_conv           -1
naphthai recipe_1               .3
heatingo heating_oil_conv       -1
heatingo recipe_1               .7
heatingo recipe_2               .8
jet_1    recipe_1               -1
jet_2    recipe_2               -1
;

proc intpoint
   maximize
   defcontype=eq
   sparsecondata
   arcdata=vars4
   condata=scon6
   conout=solutn6;
objfn profit;
upperbd available;
run;
```

Typical PROC INTPOINT Run

You start PROC INTPOINT by giving the PROC INTPOINT statement. You can specify many options that control the procedure in the PROC INTPOINT statement, or you can rely on default settings and specify no options at all. However, there are some options you must specify, or will probably have to specify:

- You must specify the BYTES= parameter indicating the size of the working memory that the procedure is allowed to use. This option has no default.
- In many instances (and certainly when solving NPSC problems), you need to specify the ARCDATA= data set. This option has a default (which is the SAS data set that was created last before PROC INTPOINT began running), but that may need to be overridden.
- The CONDATA= data set must also be specified if the problem is NPSC and has side constraints, or if it is an LP problem.
- When solving a network problem, you have to specify the NODEDATA= data set, if some model data is given in such a data set.

Some options, while optional, are frequently required. To have the optimal solution output to a SAS data set, you have to specify the CONOUT= data set. You may want to indicate reasons why optimization should stop (for example, you can indicate the maximum number of iterations that can be performed), or you might want to alter stopping criteria so that optimization does not stop prematurely. Some options enable you to control other aspects of the Interior Point algorithm. Specifying certain values for these options can reduce the time it takes to solve a problem.

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

What usually follows is a RUN statement, which indicates that all information that you, the user, need to supply to PROC INTPOINT has been given, and the procedure is to start running. This also happens if you specify a statement in your SAS program that PROC INTPOINT does not recognize as one of its own, the next DATA step or procedure.

The QUIT statement indicates that PROC INTPOINT must immediately finish.

For example, a PROC INTPOINT run might look something like this:

```
proc intpoint
    bytes=   /* working memory size */
    arcdata= /* data set */
    condata= /* data set */
    /* other options */
;
variable list specifications;  /* if necessary */
run;     /* start running, read data, */
         /* and do the optimization.  */
```

Syntax

Below are statements used in PROC INTPOINT, listed in alphabetical order as they appear in the text that follows.

PROC INTPOINT *options* **;**
CAPACITY *variable* **;**
COEF *variables* **;**
COLUMN *variable* **;**
COST *variable* **;**
DEMAND *variable* **;**
HEADNODE *variable* **;**
ID *variables* **;**
LO *variable* **;**
NAME *variable* **;**
NODE *variable* **;**
QUIT;
RHS *variable* **;**
ROW *variables* **;**
RUN;
SUPDEM *variable* **;**
SUPPLY *variable* **;**
TAILNODE *variable* **;**
TYPE *variable* **;**
VAR *variables* **;**

Most of the statements of the INTPOINT procedure are variable lists. The variables specified in the variable lists must be present in one of three input SAS data sets:

PROC INTPOINT options;	required statement
CAPACITY variable; *COST variable*; *DEMAND variable*; *HEADNODE variable*; *ID variables*; *LO variable*; *NAME variable*; *SUPPLY variable*; *TAILNODE variable*;	optional ARCDATA lists
NODE variable; *SUPDEM variable*;	optional NODEDATA lists
COEF variables; *COLUMN variable*; *RHS variable*; *ROW variables*; *TYPE variable*; *VAR variables*;	optional CONDATA lists
RUN; **QUIT**;	optional statements

Functional Summary

The following tables outline the options that can be specified in the PROC INTPOINT statement. This statement invokes the procedure.

Table 4.1. Input Data Set Options

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

Table 4.2. Options for Networks

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

Table 4.3. Miscellaneous Options

Description	Statement	Option
infinity value	INTPOINT	INFINITY=
scale constraint row, or nonarc variable column coefficients, or both	INTPOINT	SCALE=
maximization instead of minimization	INTPOINT	MAXIMIZE

Table 4.4. Data Set Read Options

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

Table 4.5. Problem Size (approx.) Options

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

Table 4.6. Memory Control Options

Description	Statement	Option
issue memory usage messages to SASLOG	INTPOINT	MEMREP
number of bytes to use for main memory	INTPOINT	BYTES=

Table 4.7. Output Data Set Option

Description	Statement	Option
constrained solution data set	INTPOINT	CONOUT=

Table 4.8. Miscellaneous Options

Description	Statement	Option
zero tolerance - optimization	INTPOINT	ZERO2=
zero tolerance - real number comparisons	INTPOINT	ZEROTOL=
display this number of similar SAS log messages, suppress the rest	INTPOINT	VERBOSE=

Table 4.9. Interior Point Algorithm Options

Description	Statement	Option
allowed amount of dual infeasibility	INTPOINT	TOLDINF=
allowed amount of primal infeasibility	INTPOINT	TOLPINF=
allowed total amount of dual infeasibility	INTPOINT	TOLTOTDINF=
allowed total amount of primal infeasibility	INTPOINT	TOLTOTPINF=
cut-off tolerance for Cholesky factorization	INTPOINT	CHOLTINYTOL=
density threshold for Cholesky processing	INTPOINT	DENSETHR=
step-length multiplier	INTPOINT	PDSTEPMULT=
preprocessing type	INTPOINT	PRSLTYPE=
print optimization progress on SAS log	INTPOINT	PRINTLEVEL2=
ratio test zero tolerance	INTPOINT	RTTOL=

Table 4.10. Interior Point Algorithm Options: Stopping Criteria

Description	Statement	Option
maximum number of Interior Point algorithm iterations	INTPOINT	MAXITERB=
Primal-Dual (Duality) gap tolerance	INTPOINT	PDGAPTOL=
stop because of complementarity	INTPOINT	STOP_C=
stop because of duality gap	INTPOINT	STOP_DG=
stop because of $infeas_b$	INTPOINT	STOP_IB=
stop because of $infeas_c$	INTPOINT	STOP_IC=
stop because of $infeas_d$	INTPOINT	STOP_ID=
stop because of complementarity	INTPOINT	AND_STOP_C=
stop because of duality gap	INTPOINT	AND_STOP_DG=
stop because of $infeas_b$	INTPOINT	AND_STOP_IB=
stop because of $infeas_c$	INTPOINT	AND_STOP_IC=
stop because of $infeas_d$	INTPOINT	AND_STOP_ID=
stop because of complementarity	INTPOINT	KEEPGOING_C=
stop because of duality gap	INTPOINT	KEEPGOING_DG=
stop because of $infeas_b$	INTPOINT	KEEPGOING_IB=
stop because of $infeas_c$	INTPOINT	KEEPGOING_IC=
stop because of $infeas_d$	INTPOINT	KEEPGOING_ID=
stop because of complementarity	INTPOINT	AND_KEEPGOING_C=
stop because of duality gap	INTPOINT	AND_KEEPGOING_DG=
stop because of $infeas_b$	INTPOINT	AND_KEEPGOING_IB=
stop because of $infeas_c$	INTPOINT	AND_KEEPGOING_IC=
stop because of $infeas_d$	INTPOINT	AND_KEEPGOING_ID=

PROC INTPOINT Statement

PROC INTPOINT *options* **;**

This statement invokes the procedure. The following options can be specified in the PROC INTPOINT statement.

Data Set Options

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

ARCDATA=*SAS-data-set*

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

If your problem is an LP, the ARCDATA= data set is optional. You can specify LP variable information such as objective function coefficients, and lower and upper bounds.

CONDATA=*SAS-data-set*

names the data set that contains the side constraint data. The data set can also contain other data such as arc costs, capacities, lower flow bounds, nonarc variable upper and lower bounds, and objective function coefficients. PROC INTPOINT needs a CONDATA= data set to solve a constrained problem. See the "CONDATA= Data Set" section on page 126 for more information.

If your problem is an LP, this data set contains the constraint data, and can also contain other data such as objective function coefficients, and lower and upper bounds. PROC INTPOINT needs a CONDATA= data set to solve an LP.

CONOUT=*SAS-data-set*

COUT=*SAS-data-set*

names the output data set that receives an optimal solution. See the "CONOUT= Data Set" section on page 135 for more information.

If PROC INTPOINT is outputting observations to the output data set and you want this to stop, press the keys used to stop SAS procedures.

NODEDATA=*SAS-data-set*

names the data set that contains the node supply and demand specifications. You do not need observations in the NODEDATA= data set for transshipment nodes. (Transshipment nodes neither supply nor demand flow.) All nodes are assumed to be trans-

shipment nodes until supply or demand data indicate otherwise. It is acceptable for some arcs to be directed toward supply nodes or away from demand nodes.

This data set is used only when you are solving network problems (not when solving LP problems), in which case the use of the NODEDATA= data set is optional provided that, if the NODEDATA= data set is not used, supply and demand details are specified by other means. Other means include using the MAXFLOW or SHORTPATH option, SUPPLY or DEMAND variable list (or both) in the ARCDATA= data set, and the SOURCE=, SUPPLY=, SINK=, or DEMAND= option in the PROC INTPOINT statement.

General Options

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

ARCS_ONLY_ARCDATA
indicates that data for arcs only are in the ARCDATA= data set. When PROC INTPOINT reads the data in the ARCDATA= data set, memory would not be wasted to receive data for nonarc variables. The read might then be performed faster. See the "How to Make the Data Read of PROC INTPOINT More Efficient" section on page 146.

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

When solving an LP, specifying the ARC_SINGLE_OBS option indicates that for all LP variables, data for each LP variable is found in only one observation of the ARCDATA= data set. When reading the data in the ARCDATA= data set, PROC INTPOINT knows that the data in an observation is for an LP variable that has not had data previously read and that needs to be checked for consistency. The read might then be performed faster.

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

See the "How to Make the Data Read of PROC INTPOINT More Efficient" section on page 146.

BYPASSDIVIDE=*b*
BYPASSDIV=*b*
BPD=*b*
should be used only when the MAXFLOW option has been specified; that is, PROC INTPOINT is solving a maximal flow problem. PROC INTPOINT prepares to solve maximal flow problems by setting up a bypass arc. This arc is directed from the SOURCE= to the SINK= and will eventually convey flow equal to INFINITY minus the maximal flow through the network. The cost of the bypass arc must be great enough to drive flow through the network, rather than through the bypass arc. Also,

the cost of the bypass arc must be greater than the eventual total cost of the maximal flow, which can be nonzero if some network arcs have nonzero costs. The cost of the bypass is set to the value of the INFINITY= option. Valid values for the BYPASSDIV= option must be greater than or equal to 1.1.

If there are no nonzero costs of arcs in the MAXFLOW problem, the cost of the bypass arc is set to 1.0 (-1.0 if maximizing) if you do not specify the BYPASSDIV= option. The default value for the BYPASSDIV= option (in the presence of nonzero arc costs) is 100.0.

BYTES=*b*

indicates the size of the main working memory (in bytes) that PROC INTPOINT will allocate. Specifying this option is mandatory. The working memory is used to store all the arrays and buffers used by PROC INTPOINT. If this memory has a size smaller than what is required to store all arrays and buffers, PROC INTPOINT uses various schemes that page information between auxiliary memory (often your machine's disk) and RAM.

For small problems, specify BYTES=100000. For large problems (those with hundreds of thousands or millions of variables), BYTES=1000000 might do. For solving problems of that size, if you are running on a machine with an inadequate amount of RAM, PROC INTPOINT's performance will suffer since it will be forced to page or to rely on virtual memory.

If you specify the MEMREP option, PROC INTPOINT will issue messages on the SAS log informing you of its memory usage; that is, how much memory is required to prevent paging, and details about the amount of paging that must be performed, if applicable.

CON_SINGLE_OBS

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

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

If the CONDATA= data set has a sparse format, and data for each arc, nonarc variable, or LP variable can be found in only one observation of the CONDATA, then specify the CON_SINGLE_OBS option. If there are n SAS variables in the ROW and COEF list, then each arc or nonarc can have at most n constraint coefficients in the model. See the "How to Make the Data Read of PROC INTPOINT More Efficient" section on page 146.

DEFCAPACITY=*c*

DC=*c*

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

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

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

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

The values do not need to be enclosed in quotes.

DEFCOST=*c*

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

DEFMINFLOW=*m*
DMF=*m*

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

DEMAND=*d*

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

GROUPED=*c*

PROC INTPOINT can take a much shorter time to read data if the data have been grouped prior to the PROC INTPOINT call. This enables PROC INTPOINT to conclude that, for instance, a new NAME list variable value seen in the ARCDATA= data set grouped by the values of the NAME list variable before PROC INTPOINT was called is new. PROC INTPOINT does not need to check that the NAME has been read in a previous observation. See the "How to Make the Data Read of PROC INTPOINT More Efficient" section on page 146.

- GROUPED=ARCDATA indicates that the ARCDATA= data set has been grouped by values of the NAME list variable. If _NAME_ is the name of the NAME list variable, you could use:

  ```
  proc sort data=arcdata;by _name_;
  ```

  prior to calling PROC INTPOINT. Technically, you do not have to sort the data, only to ensure that all similar values of the NAME list variable are grouped

together. If you specify the ARCS_ONLY_ARCDATA option, PROC INTPOINT automatically works as if GROUPED=ARCDATA is also specified.

- GROUPED=CONDATA indicates that the CONDATA= data set has been grouped.

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

  ```
  proc sort data=condata;by _row_;
  ```

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

  If the CONDATA has the sparse format, GROUPED=CONDATA indicates that the CONDATA has been grouped by values of the COLUMN list variable. If _COL_ is the name of the COLUMN list variable, you could use:

  ```
  proc sort data=condata;by _col_;
  ```

  prior to calling PROC INTPOINT. Technically, you do not have to sort the data, only to ensure that all similar values of the COLUMN list variable are grouped together.

- GROUPED=BOTH indicates that both GROUPED=ARCDATA and GROUPED=CONDATA are TRUE.

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

A data set like

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

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

INFINITY=*i*
INF=*i*

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

MAXFLOW
MF

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

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

MAXIMIZE
MAX

specifies that PROC INTPOINT find the maximum cost flow through the network. If both the MAXIMIZE and the SHORTPATH options are specified, the solution obtained is the longest path between the SOURCE= and SINK= nodes. Similarly, MAXIMIZE and MAXFLOW together cause PROC INTPOINT to find the minimum flow between these two nodes; this is zero if there are no nonzero lower flow bounds. If solving an LP, specifying the MAXIMIZE option is necessary if you want the maximal optimal solution found instead of the minimal optimum.

MEMREP

indicates that information on the memory usage and paging schemes (if necessary) is reported by PROC INTPOINT on the SAS log.

NAMECTRL=*i*

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

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

In the CONDATA= data set, if the dense data format is used (described in the "CONDATA= Data Set" section on page 126), a name of an arc or a nonarc variable is the *name* of a SAS variable listed in the VAR list specification. If the sparse data format of the CONDATA= data set is used, a name of an arc or a nonarc variable is a *value* of the SAS variable listed in the COLUMN list specification.

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

If you specify NAMECTRL=1, a name that is not defined in the ARCDATA= data set is assumed to be the name of a nonarc variable. NAMECTRL=2 treats *tail_head* as the name of the arc with these endnodes, provided no other name is used to associate data in the CONDATA= data set with this arc. If the arc does have other names that appear in the CONDATA= data set, *tail_head* is assumed to be the name of a nonarc variable. If you specify NAMECTRL=3, *tail_head* is assumed to be a name of the arc with these end nodes, whether the arc has other names or not. The default value of NAMECTRL is 3.

If the dense format is used for the CONDATA= data set, there are two circumstances that affect how this data set is read:

1. if you are running SAS Version 6, or a previous version to that, or if you are running SAS Version 7 onward and you specify

   ```
   options validvarname=v6;
   ```

   in your SAS session. Let's refer to this as *case 1*.
2. if you are running SAS Version 7 onward and you do not specify

   ```
   options validvarname=v6;
   ```

   in your SAS session. Let's refer to this as *case 2*.

If *case 1* (and the text in this and the five paragraphs that follow are relevant only for *case 1*), the SAS System converts SAS variable names in a SAS program to uppercase. The VAR list variable names are uppercased. Because of this, PROC INTPOINT automatically uppercases names of arcs and nonarc variables or LP variables (the values of the NAME list variable) in the ARCDATA= data set. The names of arcs and nonarc variables or LP variables (the values of the NAME list variable) appear uppercased in the CONOUT= data set.

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

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

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

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

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

as really being

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

The SAS variables named FROM_TO1 and FROM_TO2 are *not* associated with any of the arcs in the preceding ARCDATA= data set. The values FROM_TO1 and FROM_TO2 are different from all of the arc names from_to1, from_to2, TAIL_TO3, and ARC2. FROM_TO1 and FROM_TO2 could end up being the names of two nonarc variables.

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

If *case 2*, the SAS System does not convert SAS variable names in a SAS program to uppercase. The VAR list variable names are not uppercased. PROC INTPOINT does not automatically uppercase names of arcs and nonarc variables or LP variables (the values of the NAME list variable) in the ARCDATA= data set. PROC INTPOINT does not uppercase any SAS variable names, data set values, or indeed anything. Therefore, PROC INTPOINT respects case, and characters in the data if compared must have the right case if you mean them to be the same. Note how the input

statement in the data step that initialized the data set densecon below is specified in the following code:

```
data arcdata;
   input _from_ $ _to_ $ _name $ ;
   datalines;
from to1 .
from to2 arc2
TAIL TO3 .
;
data densecon;
   input from_to1 from_to2 arc2 TAIL_TO3;
   datalines;
2 3 5
;
proc intpoint
   arcdata=arcdata condata=densecon;
run;
```

NARCS=*n*
specifies the approximate number of arcs. See the "How to Make the Data Read of PROC INTPOINT More Efficient" section on page 146.

NCOEFS=*n*
specifies the approximate number of constraint coefficients. See the "How to Make the Data Read of PROC INTPOINT More Efficient" section on page 146.

NCONS=*n*
specifies the approximate number of constraints. See the "How to Make the Data Read of PROC INTPOINT More Efficient" section on page 146.

NNAS=*n*
specifies the approximate number of nonarc variables. See the "How to Make the Data Read of PROC INTPOINT More Efficient" section on page 146.

NNODES=*n*
specifies the approximate number of nodes. See the "How to Make the Data Read of PROC INTPOINT More Efficient" section on page 146.

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

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

See the "How to Make the Data Read of PROC INTPOINT More Efficient" section on page 146.

RHSOBS=*charstr*

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

SCALE=*s*

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

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

SHORTPATH
SP

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

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

SINK=*sinkname*
SINKNODE=*sinkname*

identifies the demand node. The SINK= option is useful when you specify the MAXFLOW option or the SHORTPATH option and you need to specify toward which node the shortest path or maximum flow is directed. The SINK= option also can be used when a minimum cost problem has only one demand node. Rather than having this information in the ARCDATA= data set or the NODEDATA= data set, use the SINK= option with an accompanying DEMAND= specification for this node.

The SINK= option must be the name of a head node of at least one arc; thus, it must have a character value. If the value of the SINK= option is not a valid SAS character variable name (if, for example, it contains embedded blanks), it must be enclosed in quotes.

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

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

Note: If the SPARSECONDATA option is not specified, and you are running SAS software Version 6 or you have specified

```
options validvarname=v6;
```

all NAME list variable values in the ARCDATA= data set are uppercased. See the "Case Sensitivity" section on page 136.

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

THRUNET
tells PROC INTPOINT to force through the network any excess supply (the amount by which total supply exceeds total demand) or any excess demand (the amount by which total demand exceeds total supply) as is required. If a network problem has unequal total supply and total demand and the THRUNET option is not specified, PROC INTPOINT drains away the excess supply or excess demand in an optimal manner. The consequences of specifying or not specifying THRUNET are discussed in the "Balancing Total Supply and Total Demand" section on page 144.

TYPEOBS=*charstr*

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

Options to Halt Optimization

VERBOSE=*v*

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

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

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

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

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

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

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

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

is issued. The value of v is the smallest value that should be specified for the VERBOSE= option so that *all* messages are displayed if PROC INTPOINT is run again with the same data and everything else.

The default value for the VERBOSE= option is 12.

ZERO2=*z*

Z2=*z*

specifies the zero tolerance level used when determining whether the final solution has been reached. ZERO2= is also used when outputting the solution to the CONOUT= data set. Values within z of zero are set to 0.0, where z is the value of the ZERO2= option. Flows close to the lower flow bound or capacity of arcs are reassigned those exact values. If there are nonarc variables, values close to the lower or upper value bound of nonarc variables are reassigned those exact values. When solving an LP problem, values close to the lower or upper value bound of LP variables are reassigned those exact values.

The ZERO2= option works when determining whether optimality has been reached or whether an element in the vector $(\Delta x^k, \Delta y^k, \Delta s^k)$ is less than or greater than zero. It is crucial to know that when determining the maximal value for the step length α in the formula

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

See the description of the PDSTEPMULT= option for more details on this computation.

Two values are deemed to be close if one is within z of the other. The default value for the ZERO2= option is 0.000001. Any value specified for the ZERO2= option that is < 0.0 or > 0.0001 is not valid.

ZEROTOL=*z*

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

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

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

TOLDINF=*t*
RTOLDINF=*t*

specifies the allowed amount of dual infeasibility. In the "Interior Point Algorithmic Details" section on page 58, the vector $infeas_d$ is defined. If all elements of this vector are $\leq t$, the solution is considered dual feasible. $infeas_d$ is replaced by a zero vector, which makes computations faster. This option is the dual equivalent to the TOLPINF= option. Increasing the value of the TOLDINF= option too much can lead to instability, but a modest increase can give the algorithm added flexibility and decrease the iteration count. Valid values for *t* are greater than $1.0E-12$. The default is $1.0E-7$.

TOLPINF=*t*
RTOLPINF=*t*

specifies the allowed amount of primal infeasibility. This option is the dual equivalent to the TOLDINF= option. In the "Interior Point: Upper Bounds" section on page 63, the vector $infeas_b$ is defined. In the "Interior Point Algorithmic Details" section on page 58, the vector $infeas_c$ is defined. If all elements in these vectors are $\leq t$, the solution is considered primal feasible. $infeas_b$ and $infeas_c$ are replaced by zero

vectors, which makes computations faster. Increasing the value of the TOLPINF= option too much can lead to instability, but a modest increase can give the algorithm added flexibility and decrease the iteration count. Valid values for *t* are greater than $1.0E-12$. The default is $1.0E-7$.

TOLTOTDINF=*t*
RTOLTOTDINF=*t*

specifies the allowed total amount of dual infeasibility. In the "Interior Point Algorithmic Details" section on page 58, the vector $infeas_d$ is defined. If $\sum_{i=1}^{n} infeas_{di} \leq t$, the solution is considered dual feasible. $infeas_d$ is replaced by a zero vector, which makes computations faster. This option is the dual equivalent to the TOLTOTPINF= option. Increasing the value of the TOLTOTDINF= option too much can lead to instability, but a modest increase can give the algorithm added flexibility and decrease the iteration count. Valid values for *t* are greater than $1.0E-12$. The default is $1.0E-7$.

TOLTOTPINF=*t*
RTOLTOTPINF=*t*

specifies the allowed total amount of primal infeasibility. This option is the dual equivalent to the TOLTOTDINF= option. In the "Interior Point: Upper Bounds" section on page 63, the vector $infeas_b$ is defined. In the "Interior Point Algorithmic Details" section on page 58, the vector $infeas_c$ is defined. If $\sum_{i=1}^{n} infeas_{bi} \leq t$ and $\sum_{i=1}^{m} infeas_{ci} \leq t$, the solution is considered primal feasible. $infeas_b$ and $infeas_c$ are replaced by zero vectors, which makes computations faster. Increasing the value of the TOLTOTPINF= option too much can lead to instability, but a modest increase can give the algorithm added flexibility and decrease the iteration count. Valid values for *t* are greater than $1.0E-12$. The default is $1.0E-7$.

CHOLTINYTOL=*c*
RCHOLTINYTOL=*c*

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

DENSETHR=*d*
RDENSETHR=*d*

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

PDSTEPMULT=*p*
RPDSTEPMULT=*p*

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

In the "Interior Point Algorithmic Details" section on page 58, the solution of the next iteration is obtained by moving along a direction from the current iteration's solution.
$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \alpha(\Delta x^k, \Delta y^k, \Delta s^k)$

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

PRSLTYPE=*p*
IPRSLTYPE=*p*

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

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

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

0 Given upper and lower bounds on each variable, the greatest and least contribution to the row activity of each variable is computed. If these are within the limits set by the upper and lower bounds on the row activity, then the row is redundant and can be discarded. Otherwise, whenever possible, the bounds on any of the variables are tightened. For example, if all coefficients in a constraint are positive and all variables have zero lower bounds, then the row's smallest contribution is zero. If the rhs value of this constraint is zero, then if the constraint type is $=$ or $\leq$, all the variables in that constraint are fixed to zero. These variables and the constraint are removed. If the constraint type is $\geq$, the constraint is redundant. If the rhs is negative and the constraint is $\leq$, the problem is infeasible. If just one variable in a row is not fixed, the row to used to impose an implicit upper or lower bound on the variable and then this row is eliminated. The preprocessor also tries to tighten the bounds on constraint right hand sides.

1 When there are exactly two unfixed variables with coefficients in an equality constraint, one variable is solved in terms of the other.

The problem will have one less variable. The new matrix will have at least two fewer coefficients and one less constraint. In other constraints where both variables appear, two coefficients are combined into one. PRSLTYPE=0 reductions are also done.

2 It may be possible to determine that an equality constraint is not constraining a variable. That is, if all variables are non-negative, then $x - \sum_i y_i = 0$ does not constrain x, since it must be non-negative if all the y_i's are non-negative. In this case, x is eliminated by subtracting this equation from all others containing x. This is useful when the only other entry for x is in the objective function. This reduction is performed if there is at most one other nonobjective coefficient. PRSLTYPE=0 reductions are also done.

3 All possible reductions are performed. PRSLTYPE=3 is the default.

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

PRINTLEVEL2=*p*

is used when you want to see PROC INTPOINT's progress to the optimum. PROC INTPOINT will produce a table on the SAS log. A row of the table is generated during each iteration and may consist of values of

- the affine step complementarity
- the complementarity of the solution for the next iteration
- the total bound infeasibility $\sum_{i=1}^{n} infeas_{bi}$ (see the $infeas_b$ array in the "Interior Point: Upper Bounds" section on page 63)
- the total constraint infeasibility $\sum_{i=1}^{m} infeas_{ci}$ (see the $infeas_c$ array in the "Interior Point Algorithmic Details" section on page 58)
- the total dual infeasibility $\sum_{i=1}^{n} infeas_{di}$ (see the $infeas_d$ array in the "Interior Point Algorithmic Details" section on page 58)

As optimization progresses, the values in all columns should converge to zero. If you specify PRINTLEVEL2=2, all columns will appear in the table. If PRINTLEVEL2=1 is specified, only the affine step complementarity, the complementarity of the solution for the next iteration, will appear. Some time is saved by not calculating the infeasibility values.

PRINTLEVEL2=2 is specified in all PROC INTPOINT runs in the "Examples" section on page 155.

RTTOL=*r*

specifies the zero tolerance used during the ratio test of the Interior Point algorithm. The ratio test determines α, the maximum feasible step-length.

Valid values for *r* are greater than 1.0e-14. The default value is 1.0e-10.

In the "Interior Point Algorithmic Details" section on page 58, the solution of the next iteration is obtained by moving along a direction from the current iteration's solution.
$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \alpha(\Delta x^k, \Delta y^k, \Delta s^k)$

Let $\alpha = Min_i\{\alpha : x_i^k + \alpha\Delta x_i = 0 | s_i^k + \alpha\Delta s_i = 0\}$. If $\alpha \leq 1$, then α is reduced slightly by multiplying it by PDSTEPMULT=. α is a value as large as possible but ≤ 1.0 and not so large that a x_i^{k+1} or s_i^{k+1} of some variable i is negative. When determining α, only negative elements of Δx and Δs are important.

RTTOL=r indicates a number close to zero so that another number n is considered truly negative if $n \leq -r$. Even though $n < 0$, if $n > -r$, n may be too close to zero and may have the wrong sign due to rounding error.

Interior Point Algorithm Options: Stopping Criteria

MAXITERB=*m*
IMAXITERB=*m*

specifies the maximum number of iterations that the Interior point algorithm can perform. The default value for *m* is 100. One of the most remarkable aspects of the Interior Point algorithm is that for most problems, it usually needs to do a small number of iterations, *no matter the size of the problem*.

PDGAPTOL=*p*
RPDGAPTOL=*p*

specifies the Primal-Dual gap or Duality gap tolerance. Duality gap is defined in the "Interior Point Algorithmic Details" section on page 58. If the relative gap $(dualitygap/(c^T x))$ between the primal and dual objectives is smaller than the value of the PDGAPTOL= option and both the primal and dual problems are feasible, then PROC INTPOINT stops optimization with a solution that is deemed optimal. Valid values for *p* are between $1.0E-12$ and $1.0E-1$. the default is $1.0E-7$.

STOP_C=*s*

is used to determine whether optimization should stop. At the beginning of each iteration, if complementarity (the value of the Complem-ity column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $<=$ *s*, optimization will stop. This option is discussed in the "Stopping Criteria" section on page 151.

STOP_DG=*s*

is used to determine whether optimization should stop. At the beginning of each iteration, if the duality gap (the value of the Duality_gap column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $<=$ *s*, optimization will stop. This option is discussed in the "Stopping Criteria" section on page 151.

STOP_IB=*s*

is used to determine whether optimization should stop. At the beginning of each iteration, if total bound infeasibility $\sum_{i=1}^{n} infeas_{bi}$ (see the $infeas_b$ array in the "Interior Point: Upper Bounds" section on page 63; this value appears in the Tot_infeasb column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $<=$ *s*, optimization will stop. This option is discussed in the

"Stopping Criteria" section on page 151.

STOP_IC=*s*

is used to determine whether optimization should stop. At the beginning of each iteration, if total constraint infeasibility $\sum_{i=1}^{m} infeas_{ci}$ (see the $infeas_c$ array in the "Interior Point Algorithmic Details" section on page 58; this value appears in the Tot_infeasc column in the table produced when you specify PRINTLEVEL2=2) is $<=$ *s*, optimization will stop. This option is discussed in the "Stopping Criteria" section on page 151.

STOP_ID=*s*

is used to determine whether optimization should stop. At the beginning of each iteration, if total dual infeasibility $\sum_{i=1}^{n} infeas_{di}$ (see the $infeas_d$ array in the "Interior Point Algorithmic Details" section on page 58; this value appears in the Tot_infeasd column in the table produced when you specify PRINTLEVEL2=2) is $<=$ *s*, optimization will stop. This option is discussed in the "Stopping Criteria" section on page 151.

AND_STOP_C=*s*

is used to determine whether optimization should stop. At the beginning of each iteration, if complementarity (the value of the Complem-ity column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $<=$ *s*, *and* the other conditions related to other AND_STOP parameters are also satisfied, optimization will stop. This option is discussed in the "Stopping Criteria" section on page 151.

AND_STOP_DG=*s*

is used to determine whether optimization should stop. At the beginning of each iteration, if the duality gap (the value of the Duality_gap column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $<=$ *s*, *and* the other conditions related to other AND_STOP parameters are also satisfied, optimization will stop. This option is discussed in the "Stopping Criteria" section on page 151.

AND_STOP_IB=*s*

is used to determine whether optimization should stop. At the beginning of each iteration, if total bound infeasibility $\sum_{i=1}^{n} infeas_{bi}$ (see the $infeas_b$ array in the "Interior Point: Upper Bounds" section on page 63; this value appears in the Tot_infeasb column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $<=$ *s*, *and* the other conditions related to other AND_STOP parameters are also satisfied, optimization will stop. This option is discussed in the "Stopping Criteria" section on page 151.

AND_STOP_IC=*s*

is used to determine whether optimization should stop. At the beginning of each iteration, if total constraint infeasibility $\sum_{i=1}^{m} infeas_{ci}$ (see the $infeas_c$ array in the "Interior Point Algorithmic Details" section on page 58; this value appears in the Tot_infeasc column in the table produced when you specify PRINTLEVEL2=2) is $<=$ *s*, *and* the other conditions related to other AND_STOP parameters are also satisfied, optimization will stop. This option is discussed in the "Stopping Criteria" section on page 151.

AND_STOP_ID=*s*

is used to determine whether optimization should stop. At the beginning of each iteration, if total dual infeasibility $\sum_{i=1}^{n} infeas_{di}$ (see the $infeas_d$ array in the "Interior Point Algorithmic Details" section on page 58; this value appears in the Tot_infeasd column in the table produced when you specify PRINTLEVEL2=2) is $<= s$, *and* the other conditions related to other AND_STOP parameters are also satisfied, optimization will stop. This option is discussed in the "Stopping Criteria" section on page 151.

KEEPGOING_C=*s*

is used to determine whether optimization should stop. If a stopping condition is met, if complementarity (the value of the Complem-ity column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $> s$, optimization will continue. This option is discussed in the "Stopping Criteria" section on page 151.

KEEPGOING_DG=*s*

is used to determine whether optimization should stop. If a stopping condition is met, if the duality gap (the value of the Duality_gap column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $> s$, optimization will continue. This option is discussed in the "Stopping Criteria" section on page 151.

KEEPGOING_IB=*s*

is used to determine whether optimization should stop. If a stopping condition is met, if total bound infeasibility $\sum_{i=1}^{n} infeas_{bi}$ (see the $infeas_b$ array in the "Interior Point: Upper Bounds" section on page 63; this value appears in the Tot_infeasb column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $> s$, optimization will continue. This option is discussed in the "Stopping Criteria" section on page 151.

KEEPGOING_IC=*s*

is used to determine whether optimization should stop. If a stopping condition is met, if total constraint infeasibility $\sum_{i=1}^{m} infeas_{ci}$ (see the $infeas_c$ array in the "Interior Point Algorithmic Details" section on page 58; this value appears in the Tot_infeasc column in the table produced when you specify PRINTLEVEL2=2) is $> s$, optimization will continue. This option is discussed in the "Stopping Criteria" section on page 151.

KEEPGOING_ID=*s*

is used to determine whether optimization should stop. If a stopping condition is met, if total dual infeasibility $\sum_{i=1}^{n} infeas_{di}$ (see the $infeas_d$ array in the "Interior Point Algorithmic Details" section on page 58; this value appears in the Tot_infeasd column in the table produced when you specify PRINTLEVEL2=2) is $> s$, optimization will continue. This option is discussed in the "Stopping Criteria" section on page 151.

AND_KEEPGOING_C=*s*

is used to determine whether optimization should stop. If a stopping condition is met, if complementarity (the value of the Complem-ity column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $> s$, *and* the other conditions related to other AND_KEEPGOING parameters are also satisfied, optimization will continue. This option is discussed in the "Stopping Criteria" section on page 151.

AND_KEEPGOING_DG=*s*
is used to determine whether optimization should stop. If a stopping condition is met, if the duality gap (the value of the Duality_gap column in the table produced when you specify PRINTLEVEL2=1 or PRINTLEVEL2=2) is $>$ *s*, *and* the other conditions related to other AND_KEEPGOING parameters are also satisfied, optimization will continue. This option is discussed in the "Stopping Criteria" section on page 151.

AND_KEEPGOING_IB=*s*
is used to determine whether optimization should stop. If a stopping condition is met, if total bound infeasibility $\sum_{i=1}^{n} infeas_{bi}$ (see the $infeas_b$ array in the "Interior Point: Upper Bounds" section on page 63; this value appears in the Tot_infeasb column in the table produced when you specify PRINTLEVEL2=2) is $>$ *s*, *and* the other conditions related to other AND_KEEPGOING parameters are also satisfied, optimization will continue. This option is discussed in the "Stopping Criteria" section on page 151.

AND_KEEPGOING_IC=*s*
is used to determine whether optimization should stop. If a stopping condition is met, if total constraint infeasibility $\sum_{i=1}^{m} infeas_{ci}$ (see the $infeas_c$ array in the "Interior Point Algorithmic Details" section on page 58; this value appears in the Tot_infeasc column in the table produced when you specify PRINTLEVEL2=2) is $>$ *s*, *and* the other conditions related to other AND_KEEPGOING parameters are also satisfied, optimization will continue. This option is discussed in the "Stopping Criteria" section on page 151.

AND_KEEPGOING_ID=*s*
is used to determine whether optimization should stop. If a stopping condition is met, if total dual infeasibility $\sum_{i=1}^{n} infeas_{di}$ (see the $infeas_d$ array in the "Interior Point Algorithmic Details" section on page 58; this value appears in the Tot_infeasd column in the table produced when you specify PRINTLEVEL2=2) is $>$ *s*, *and* the other conditions related to other AND_KEEPGOING parameters are also satisfied, optimization will continue. This option is discussed in the "Stopping Criteria" section on page 151.

CAPACITY Statement

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

UPPERBD *variable* **;**

The CAPACITY statement identifies the SAS variable in the ARCDATA= data set that contains the maximum feasible flow or capacity of the network arcs. If an observation contains nonarc variable information, the CAPACITY list variable is the upper value bound for the nonarc variable named in the NAME list variable in that observation.

When solving an LP, the CAPACITY statement identifies the SAS variable in the

ARCDATA= data set that contains the maximum feasible value of the LP variables.

The CAPACITY list variable must have numeric values. It is not necessary to have a CAPACITY statement if the name of the SAS variable is _CAPAC_, _UPPER_, _UPPERBD, or _HI_.

COEF Statement

COEF *variables* **;**

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

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

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

COLUMN Statement

COLUMN *variable* **;**

The COLUMN list is used with the sparse input format of the CONDATA= data set.

This list consists of one SAS variable in the CONDATA= data set that has as values the names of arc variables, nonarc variables, or missing values. When solving an LP, this list consists of one SAS variable in the CONDATA= data set that has as values the names of LP variables, or missing values. Some, if not all, of these values also can be values of the NAME list variables of the ARCDATA= data set. The COLUMN list variable can have other special values (Refer to the TYPEOBS= and RHSOBS= options). If the COLUMN list is not specified after the PROC INTPOINT statement, the CONDATA= data set is searched and a SAS variable named _COLUMN_ is

used. The COLUMN list variable must have character values.

COST Statement

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

The COST statement identifies the SAS variable in the ARCDATA= data set that contains the per unit flow cost through an arc. If an observation contains nonarc variable information, the value of the COST list variable is the objective function coefficient of the nonarc variable named in the NAME list variable in that observation.

If solving an LP, the COST statement identifies the SAS variable in the ARCDATA= data set that contains the per unit objective function coefficient of an LP variable named in the NAME list variable in that observation.

The COST list variable must have numeric values. It is not necessary to specify a COST statement if the name of the SAS variable is _COST_ or _LENGTH_.

DEMAND Statement

DEMAND *variable* **;**

The DEMAND statement identifies the SAS variable in the ARCDATA= data set that contains the demand at the node named in the corresponding HEADNODE list variable. The DEMAND list variable must have numeric values. It is not necessary to have a DEMAND statement if the name of this SAS variable is _DEMAND_. See the "Missing S Supply and Missing D Demand Values" section on page 139 for cases when the SUPDEM list variable values can have other values. There should be no DEMAND statement if you are solving an LP.

HEADNODE Statement

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

TONODE *variable* **;**

TO *variable* **;**

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

is _HEAD_ or _TO_. The HEADNODE variable must have character values.

There should be no HEAD statement if you are solving an LP.

ID Statement

ID *variables* **;**

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

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

LO Statement

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

MINFLOW *variable* **;**

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

NAME Statement

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

VARNAME *variable* **;**

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

NODE Statement

NODE *variable* **;**

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

QUIT Statement

QUIT;

The QUIT statement indicates that PROC INTPOINT is to stop immediately. The solution is not saved in the CONOUT= data set. The QUIT statement has no options.

RHS Statement

RHS *variable* **;**

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

ROW Statement

ROW *variables* **;**

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

If the dense data format is used, there must be only one SAS variable in this list. In this case, if a ROW list is not specified, the CONDATA= data set is searched and the SAS variable with the name _ROW_ or _CON_ is used. If that search fail to find a suitable SAS variable, data for each constraint must reside in only one observation.

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

RUN Statement

RUN;

The RUN statement causes optimization to be started. The RUN statement has no options. If PROC INTPOINT is called and is not terminated because of an error or a QUIT statement, and you have not used a RUN statement, a RUN statement is assumed implicitly as the last statement of PROC INTPOINT. Therefore, PROC INTPOINT reads that data, performs optimization, and saves the optimal solution in the CONOUT= data set.

SUPDEM Statement

SUPDEM *variable* **;**

The SAS variable in this list, which must be present in the NODEDATA= data set, contains supply and demand information for the nodes in the NODE list. A positive SUPDEM list variable value s ($s > 0$) denotes that the node named in the NODE list variable can supply s units of flow. A negative SUPDEM list variable value $-d$ ($d > 0$) means that this node demands d units of flow. If a SAS variable is not explicitly specified, a SAS variable with the name _SUPDEM_ or _SD_ in the NODEDATA=

data set is used as the SUPDEM variable. If a node is a transshipment node (neither a supply nor a demand node), an observation associated with this node need not be present in the NODEDATA= data set. If present, the SUPDEM list variable value must be zero or a missing value. See the "Missing S Supply and Missing D Demand Values" section on page 139 for cases when the SUPDEM list variable values can have other values.

SUPPLY Statement

SUPPLY *variable* **;**

The SUPPLY statement identifies the SAS variable in the ARCDATA= data set that contains the supply at the node named in that observation's TAILNODE list variable. If a tail node does not supply flow, use zero or a missing value for the observation's SUPPLY list variable value. If a tail node has supply capability, a missing value indicates that the supply quantity is given in another observation. It is not necessary to have a SUPPLY statement if the name of this SAS variable is _SUPPLY_. See the "Missing S Supply and Missing D Demand Values" section on page 139 for cases when the SUPDEM list variable values can have other values. There should be no SUPPLY statement if you are solving an LP.

TAILNODE Statement

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

FROMNODE *variable* **;**

FROM *variable* **;**

The TAILNODE statement specifies the SAS variable that must (when solving an NPSC problem) be present in the ARCDATA= data set that has as values the names of tail nodes of arcs. The TAILNODE variable must have character values. It is not necessary to have a TAILNODE statement if the name of the SAS variable is _TAIL_ or _FROM_. If the TAILNODE list variable value is missing, it is assumed that the observation of the ARCDATA= data set contains information concerning a nonarc variable. There should be no TAILNODE statement if you are solving an LP.

TYPE Statement

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

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

TYPE List Variable Values

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

The valid TYPE list variable values are

<	less than or equal to ($\leq$)
=	equal to (=)
>	greater than or equal to ($\geq$)
CAPAC	capacity
COST	cost
EQ	equal to
FREE	free row (used only for Linear Programs solved by Interior Point)
GE	greater than or equal to
LE	less than or equal to
LOWERBD	lower flow or value bound
LOW*blank*	lower flow or value bound
MAXIMIZE	maximize (opposite of cost)
MINIMIZE	minimize (same as cost)
OBJECTIVE	objective function (same as cost)
RHS	rhs of constraint
TYPE	type of constraint
UPPCOST	reserved for future use
UNREST	unrestricted variable (used only for Linear Programs solved by Interior Point)
UPPER	upper value bound or capacity; second letter must not be N

The valid TYPE list variable values in function order are

- **LE** less than or equal to ($\leq$)
- **EQ** Equal to (=)
- **GE** Greater than or equal to ($\geq$)
- **CO**ST
  MINIMIZE
  MAXIMIZE
  OBJECTIVE
  cost or objective function coefficient
- **CA**PAC
  UPPER capacity or upper value bound

- **LOW**ERBD
  LOW*blank* Lower flow or value bound
- **R**HS rhs of constraint
- **T**YPE type of constraint

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

VAR Statement

VAR *variables* **;**

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

Details

Input Data Sets

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

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

ARCDATA= Data Set

See the "Getting Started: NPSC Problems" section on page 74 and the "Introductory NPSC Example" section on page 75 for a description of this input data set.

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

tail	_head_	_cost_	_capac_	_lo_
A	B	50	.	.
A	B	.	100	.
A	B	.	.	10
A	B	50	100	.
A	B	.	100	10
A	B	50	.	10
A	B	50	100	10

Similarly, for a nonarc variable that has an upper bound of 100, a lower bound of 10, and an objective function coefficient of 50, the _TAIL_ and _HEAD_ values are missing.

When solving an LP that has an LP variable named my_var with an upper bound of 100, a lower bound of 10, and an objective function coefficient of 50, some possible observations in the ARCDATA= data set are

name	_cost_	_capac_	_lo_
my_var	50	.	.
my_var	.	100	.
my_var	.	.	10
my_var	50	100	.
my_var	.	100	10
my_var	50	.	10
my_var	50	100	10

CONDATA= Data Set

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

Dense Input Format

If the dense format is used, most SAS variables in the CONDATA= data set belong to the VAR list. The names of the SAS variables belonging to this list have names of arc and nonarc variables or, if solving an LP, names of the LP variables. These names can be values of the SAS variables in the ARCDATA= data set that belong to the NAME list, or names of nonarc variables, or names in the form *tail_head*, or any combination of these three forms. Names in the form *tail_head* are default arc names, and if you use them, you must specify node names in the ARCDATA= data set (values of the TAILNODE and HEADNODE list variables).

The CONDATA= data set can have three other SAS variables belonging, respectively, to the ROW, the TYPE, and the RHS lists. The CONDATA= data set of the oil industry example in the "Introductory NPSC Example" section on page 75 uses the dense data format.

Consider the SAS code that creates a dense format CONDATA= data set that has data for three constraints. This data set was used in the "Introductory NPSC Example" section on page 75:

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

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

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

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

```
data cond1b;
   input _row_ $
         m_e_ref1 m_e_ref2 thruput1 r1_gas thruput2 r2_gas
         _type_ $ _rhs_;
   datalines;
costrow 63  .   .   .   .   . .      .
costrow  . 81 200   .   .   . cost   .
.        .  .   .   . 220   . cost   .
caprow   .  .   .   .   .   . capac  .
caprow  95  . 175   . 100 100 .      .
caprow   . 80 175 140   .   . .      .
lorow   20 10  50   .  35   . lo     .
con1    -2  .   1   .   .   . .      .
con1     .  .   .   .   .   . >=   -15
con2     . -2   .   .   1   . GE   -15
con3     .  .  -3   4   .   . EQ     0
con4     .  .   .   .  -3   4  =     0
;
```

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

constraint or special row type, it is treated as a constraint with DEFCONTYPE= (or DEFCONTYPE= default) type. Associated VAR list variable values are coefficients of that constraint.

Sparse Input Format

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

A sparse format CONDATA= data set for the oil industry example in the "Introductory NPSC Example" section on page 75 is displayed below.

```
title 'Setting Up Condata = Cond2 for PROC INTPOINT';
data cond2;
   input _column_ $ _row1 $ _coef1 _row2 $ _coef2 ;
   datalines;
m_e_ref1  con1  -2     .   .
m_e_ref2  con2  -2     .   .
thruput1  con1   1  con3  -3
r1_gas       .   .  con3   4
thruput2  con2   1  con4  -3
r2_gas       .   .  con4   4
_type_    con1   1  con2   1
_type_    con3   0  con4   0
_rhs_     con1 -15  con2 -15
;
```

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

An alternative, using a TYPE list variable, is

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

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

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

Also note that DEFCONTYPE=EQ could have been specified, saving you from having to include in the data that con3 and con4 are of this type.

In the oil industry example, arc costs, capacities, and lower flow bounds are presented in the ARCDATA= data set. Alternatively, you could have used the following input data sets. The arcd2 data set has only two SAS variables. For each arc, there is an observation in which the arc's tail and head node is specified:

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

title 'Setting Up Condata = Cond4 for PROC INTPOINT';
data cond4;
   input _column_&$27. _row1 $ _coef1 _row2 $ _coef2 _type_ $ ;
   datalines;
.                              con1 -15   con2 -15    ge
.                            costrow   .      .   .  cost
.                                  .   . caprow   . capac
middle east_refinery 1         con1  -2      .   .     .
middle east_refinery 2         con2  -2      .   .     .
refinery 1_r1                  con1   1   con3  -3     .
r1_ref1 gas                       .   .   con3   4     =
refinery 2_r2                  con2   1   con4  -3     .
r2_ref2 gas                       .   .   con4   4    eq
middle east_refinery 1       costrow  63 caprow  95     .
middle east_refinery 2       costrow  81 caprow  80     .
u.s.a._refinery 1            costrow  55      .   .     .
u.s.a._refinery 2            costrow  49      .   .     .
refinery 1_r1                costrow 200 caprow 175     .
refinery 2_r2                costrow 220 caprow 100     .
r1_ref1 gas                        .   . caprow 140     .
r1_ref1 diesel                     .   . caprow  75     .
r2_ref2 gas                        .   . caprow 100     .
r2_ref2 diesel                     .   . caprow  75     .
ref1 gas_servstn1 gas        costrow  15 caprow  70     .
ref1 gas_servstn2 gas        costrow  22 caprow  60     .
ref1 diesel_servstn1 diesel  costrow  18      .   .     .
ref1 diesel_servstn2 diesel  costrow  17      .   .     .
ref2 gas_servstn1 gas        costrow  17 caprow  35     .
ref2 gas_servstn2 gas        costrow  31      .   .     .
ref2 diesel_servstn1 diesel  costrow  36      .   .     .
```

```
ref2 diesel_servstn2 diesel  costrow  23      .   .     .
middle east_refinery 1               .  20      .   .    lo
middle east_refinery 2               .  10      .   .    lo
refinery 1_r1                        .  50      .   .    lo
refinery 2_r2                        .  35      .   .    lo
ref2 gas_servstn1 gas                .   5      .   .    lo
;
```

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

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

```
title 'Setting Up Condata = Cond5 for PROC INTPOINT';
data cond5;
   input _column_&$27. _row1 $ _coef1 _row2 $ _coef2 _type_ $ ;
   datalines;
middle east_refinery 1          con1  -2 costrow  63     .
middle east_refinery 2          con2  -2   lorow  10     .
refinery 1_r1                      .   .    con3  -3     =
r1_ref1 gas                   caprow 140    con3   4     .
refinery 2_r2                   con2   1    con4  -3     .
r2_ref2 gas                        .   .    con4   4    eq
.                               CON1 -15    CON2 -15    GE
ref2 diesel_servstn1 diesel        .  36 costrow   .  cost
.                                  .   .  caprow   . capac
.                              lorow   .       .   .    lo
middle east_refinery 1        caprow  95   lorow  20     .
middle east_refinery 2        caprow  80 costrow  81     .
u.s.a._refinery 1                  .   .       .  55  cost
u.s.a._refinery 2             costrow  49       .   .     .
refinery 1_r1                   con1   1  caprow 175     .
refinery 1_r1                  lorow  50 costrow 200     .
refinery 2_r2                 costrow 220  caprow 100     .
refinery 2_r2                      .  35       .   .    lo
r1_ref1 diesel                caprow2  75       .   . capac
r2_ref2 gas                        .   .  caprow 100     .
r2_ref2 diesel                caprow2  75       .   .     .
ref1 gas_servstn1 gas         costrow  15  caprow  70     .
ref1 gas_servstn2 gas         caprow2  60 costrow  22     .
ref1 diesel_servstn1 diesel        .   . costrow  18     .
ref1 diesel_servstn2 diesel   costrow  17       .   .     .
```

```
ref2 gas_servstn1 gas         costrow  17   lorow   5     .
ref2 gas_servstn1 gas               .   . caprow2  35     .
ref2 gas_servstn2 gas               .  31       .   .  cost
ref2 diesel_servstn2 diesel         .   . costrow  23     .
;
```

Converting from an NPSC to an LP Problem

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

1. Identify the nodal flow conservation constraints. The coefficient matrix of these constraints (a submatrix of the LP's constraint coefficient matrix) has only two nonzero elements in each column, -1 and 1.
2. Assign a node to each nodal flow conservation constraint.
3. The rhs values of conservation constraints are the corresponding node's supplies and demands. Use this information to create the NODEDATA= data set.
4. Assign an arc to each column of the flow conservation constraint coefficient matrix. The arc is directed from the node associated with the row that has the 1 element in it and directed toward to the node associated with the row that has the -1 element in it. Set up the ARCDATA= data set that has two SAS variables. This data set could resemble ARCDATA=arcd2. These will eventually be the TAILNODE and HEADNODE list variables when PROC INTPOINT is used. Each observation consists of the tail and head node of each arc.
5. Remove from the data of the linear program all data concerning the nodal flow conservation constraints.
6. Put the remaining data into a CONDATA= data set. This data set will probably resemble CONDATA=cond4 or CONDATA=cond5.

The Sparse Format Summary

The following list illustrates possible CONDATA= data set observation sparse formats. a1, b1, b2, b3 and c1 have as a _COLUMN_ variable value either the name of an arc (possibly in the form *tail_head*) or the name of a nonarc variable (if you are solving an NPSC), or the name of the LP variable (if you are solving an LP). These are collectively referred to as variable in the tables that follow.

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

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

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

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

	COLUMN	_TYPE_	_COEFx_
b1	variable	UPPERBD	capacity
b2	variable	LOWERBD	lower flow
b3	variable	COST	cost

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

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

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

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

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

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

```
_COLUMN_          _TYPE_          _ROWx_          _COEFx_

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

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

```
_COLUMN_ _TYPE_ _ROW1_ _COEF1_ _ROW2_ _COEF2_ _ROW3_ _COEF3_

 var_c     .    up_row  upval  lo_row  loval  co_row   cost
```

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

NODEDATA= Data Set

See the "Getting Started: NPSC Problems" section on page 74 and the "Introductory NPSC Example" section on page 75 for a description of this input data set.

Output Data Set

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

For LP problems, the procedure determines the value that should be assigned to each variable. The goal is that the lower and upper value bounds and the constraints are not violated. This goal is reached when the total cost incurred by such a value assignment is feasible and optimal.

The CONOUT= data set can be produced and contains a solution obtained after performing optimization.

CONOUT= Data Set

The CONOUT= data set contains the following variables, and their possible values in an observation are

FROM	a tail node of an arc. This is a missing value if an observation has information about a nonarc variable.
_TO_	a head node of an arc. This is a missing value if an observation has information about a nonarc variable.
COST	the cost of an arc or the objective function coefficient of a nonarc variable.
CAPAC	the capacity of an arc or upper value bound of a nonarc variable
_LO_	the lower flow bound of an arc or lower value bound of a nonarc variable
NAME	a name of an arc or nonarc variable

SUPPLY	the supply of the tail node of the arc in the observation. This is a missing value if an observation has information about a nonarc variable.
DEMAND	the demand of the head node of the arc in the observation. This is a missing value if an observation has information about a nonarc variable.
FLOW	the flow through the arc or value of the nonarc variable
FCOST	flow cost, the product of _COST_ and _FLOW_
RCOST	the reduced cost of the arc or nonarc variable
STATUS	the status of the arc or nonarc variable

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

```
from tail;
```

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

Case Sensitivity

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

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

```
proc intpoint source=NotableNode
```

then PROC INTPOINT looks for a value of the TAILNODE list variable that is NotableNode.

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

```
proc intpoint source=NotableNode
```

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

```
options validvarname=v6;
```

If the SPARSECONDATA option is not specified, and you are running SAS software Version 6, or you are running SAS software Version 7 onward and have specified:

```
options validvarname=v6;
```

all values of the SAS variables that belong to the NAME list are uppercased. This is because the SAS System has uppercased all SAS variable names, particularly those in the VAR list of the CONDATA= data set.

Entities that contain blanks must be enclosed in quotes.

Loop Arcs

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

Multiple Arcs

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

Flow and Value Bounds

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

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

When solving an LP, the upper and lower value bound of an LP variable can be equal. Negative upper and lower bounds are permitted. If both are negative, the lower bound must be at least as negative as the upper bound.

In short, for any problem to be feasible, a lower bound must be $\leq$ the associated upper bound.

Tightening Bounds and Side Constraints

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

- The bounds of arcs and nonarc variables, or the bounds of LP variables, are made as tight as possible. If several different values are given for the lower flow bound of an arc, the greatest value is used. If several different values are given for the lower bound of a nonarc or LP variable, the greatest value is used. If several different values are given for the capacity of an arc, the smallest value is used. If several different values are given for the upper bound of a nonarc or LP variable, the smallest value is used.
- Several values can be given for inequality constraint right-hand-sides. For a particular constraint, the lowest rhs value is used for the rhs if the constraint is of *less than or equal to* type. For a particular constraint, the greatest rhs value is used for the rhs if the constraint is of *greater than or equal to* type.

Reasons for Infeasibility

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

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

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

Consider the network illustrated in Figure 4.10.

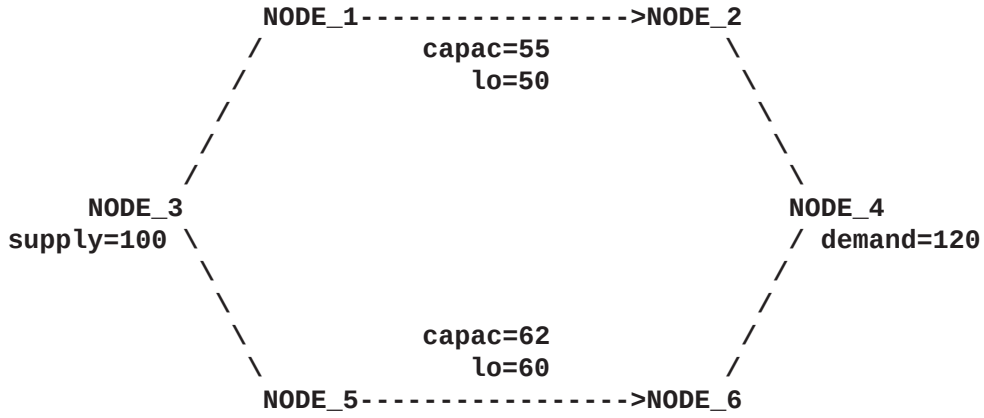


Figure 4.10. An Infeasible Network

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

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

Missing S Supply and Missing D Demand Values

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

Suppose the oil example in the "Introductory NPSC Example" section on page 75 is changed so that crude oil can be obtained from either the Middle East or U.S.A. in any amounts. You should specify that the node middle east is a supply node, but you do not want to stipulate that it supplies 100 units, as before. The node u.s.a. should also remain a supply node, but you do not want to stipulate that it supplies 80 units. You must specify that these nodes have missing S supply capabilities:

```
title  'Oil Industry Example';
title3 'Crude Oil can come from anywhere';
data miss_s;
   missing S;
   input   _node_&$15. _sd_;
   datalines;
middle east         S
u.s.a.              S
servstn1 gas      -95
servstn1 diesel   -30
servstn2 gas      -40
servstn2 diesel   -15
;
```

The following PROC INTPOINT run uses the same ARCDATA= and CONDATA= data sets used in the "Introductory NPSC Example" section on page 75:

```
proc intpoint
   bytes=100000
   nodedata=miss_s       /* the supply (missing S) and */
                         /* demand data                */
   arcdata=arcd1         /* the arc descriptions       */
   condata=cond1         /* the side constraints       */
   conout=solution;      /* the solution data set      */
run;
proc print;
   var _from_ _to_ _cost_ _capac_ _lo_ _flow_ _fcost_;
   sum _fcost_;
run;
```

The following messages appear on the SAS log:

```
NOTE: Number of nodes= 14 .
NOTE: Number of supply nodes= 2 .
NOTE: Of these, 2 have unspecified (.S) supply capability.
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 0 , total demand= 180 .
NOTE: Number of arcs= 18 .
NOTE: Number of <= side constraints= 0 .
NOTE: Number of == side constraints= 2 .
NOTE: Number of >= side constraints= 2 .
NOTE: Number of side constraint coefficients= 8 .
NOTE: The following messages relate to the equivalent
      Linear Program solved by the Interior Point
      algorithm.
NOTE: Number of <= constraints= 0 .
NOTE: Number of == constraints= 17 .
NOTE: Number of >= constraints= 2 .
NOTE: Number of constraint coefficients= 48 .
NOTE: Number of variables= 20 .
NOTE: After preprocessing, number of <= constraints= 0.
NOTE: After preprocessing, number of == constraints= 8.
```

```
NOTE: After preprocessing, number of >= constraints= 2.
NOTE: The preprocessor eliminated 9 constraints from the
      problem.
NOTE: The preprocessor eliminated 20 constraint
      coefficients from the problem.
NOTE: After preprocessing, number of variables= 11.
NOTE: The preprocessor eliminated 9 variables from the
      problem.
NOTE: 2 columns, 0 rows and 2 coefficients were added to
      the problem to handle unrestricted variables,
      variables that are split, and constraint slack or
      surplus variables.
NOTE: There are 21 nonzero elements in A * A transpose.
NOTE: Of the 10 rows and columns, 4 are sparse.
NOTE: There are 15 nonzero superdiagonal elements in the
      sparse rows of the factored A * A transpose. This
      includes fill-in.
NOTE: There are 5 operations of the form
      u[i,j]=u[i,j]-u[q,j]*u[q,i]/u[q,q] to factorize the
      sparse rows of A * A transpose.
NOTE: Bound feasibility attained by iteration 1.
NOTE: Dual feasibility attained by iteration 1.
NOTE: Constraint feasibility attained by iteration 2.
NOTE: Primal-Dual Predictor-Corrector Interior point
      algorithm performed 7 iterations.
NOTE: Objective = 50075.
NOTE: The data set WORK.SOLUTION has 18 observations and
      14 variables.
NOTE: There were 18 observations read from the data set
      WORK.ARCD1.
NOTE: There were 6 observations read from the data set
      WORK.MISS_S.
NOTE: There were 4 observations read from the data set
      WORK.COND1.
NOTE: The data set WORK.SOLUTION has 18 observations and
      14 variables.
```

The CONOUT= data set is shown in Figure 4.11.

Oil Industry Example

Crude Oil can come from anywhere

Obs	_from_	_to_	_cost_	_capac_	_lo_	_FLOW_	_FCOST_
1	refinery 1	r1	200	175	50	145.00	29000.00
2	refinery 2	r2	220	100	35	35.00	7700.00
3	r1	ref1 diesel	0	75	0	36.25	0.00
4	r1	ref1 gas	0	140	0	108.75	0.00
5	r2	ref2 diesel	0	75	0	8.75	0.00
6	r2	ref2 gas	0	100	0	26.25	0.00
7	middle east	refinery 1	63	95	20	20.00	1260.00
8	u.s.a.	refinery 1	55	99999999	0	125.00	6875.00
9	middle east	refinery 2	81	80	10	10.00	810.00
10	u.s.a.	refinery 2	49	99999999	0	25.00	1225.00
11	ref1 diesel	servstn1 diesel	18	99999999	0	30.00	540.00
12	ref2 diesel	servstn1 diesel	36	99999999	0	0.00	0.00
13	ref1 gas	servstn1 gas	15	70	0	68.75	1031.25
14	ref2 gas	servstn1 gas	17	35	5	26.25	446.25
15	ref1 diesel	servstn2 diesel	17	99999999	0	6.25	106.25
16	ref2 diesel	servstn2 diesel	23	99999999	0	8.75	201.25
17	ref1 gas	servstn2 gas	22	60	0	40.00	880.00
18	ref2 gas	servstn2 gas	31	99999999	0	0.00	0.00
							50075.00

Figure 4.11. Missing S SUPDEM values in NODEDATA

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

```
data miss_s_x;
   missing S;
   input   _node_&$15. _sd_;
   datalines;
middle east         1
u.s.a.              1
servstn1 gas      -95
servstn1 diesel   -30
servstn2 gas      -40
servstn2 diesel   -15
;
proc intpoint
   bytes=100000
   thrunet
   nodedata=miss_s_x     /* No supply (missing S)      */
   arcdata=arcd1         /* the arc descriptions       */
   condata=cond1         /* the side constraints       */
   conout=solution;      /* the solution data set      */
run;
proc print;
   var _from_ _to_ _cost_ _capac_ _lo_ _flow_ _fcost_;
   sum _fcost_;
run;
```

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

```
NOTE: Number of nodes= 14 .
NOTE: Number of supply nodes= 2 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 2 , total demand= 180 .
NOTE: Number of arcs= 18 .
NOTE: Number of <= side constraints= 0 .
NOTE: Number of == side constraints= 2 .
NOTE: Number of >= side constraints= 2 .
NOTE: Number of side constraint coefficients= 8 .
NOTE: The following messages relate to the equivalent
      Linear Program solved by the Interior Point
      algorithm.
NOTE: Number of <= constraints= 0 .
NOTE: Number of == constraints= 17 .
NOTE: Number of >= constraints= 2 .
NOTE: Number of constraint coefficients= 48 .
NOTE: Number of variables= 20 .
NOTE: After preprocessing, number of <= constraints= 0.
NOTE: After preprocessing, number of == constraints= 8.
NOTE: After preprocessing, number of >= constraints= 2.
NOTE: The preprocessor eliminated 9 constraints from the
      problem.
NOTE: The preprocessor eliminated 20 constraint
      coefficients from the problem.
NOTE: After preprocessing, number of variables= 11.
NOTE: The preprocessor eliminated 9 variables from the
```

```
      problem.
NOTE: 2 columns, 0 rows and 2 coefficients were added to
      the problem to handle unrestricted variables,
      variables that are split, and constraint slack or
      surplus variables.
NOTE: There are 21 nonzero elements in A * A transpose.
NOTE: Of the 10 rows and columns, 4 are sparse.
NOTE: There are 15 nonzero superdiagonal elements in the
      sparse rows of the factored A * A transpose. This
      includes fill-in.
NOTE: There are 5 operations of the form
      u[i,j]=u[i,j]-u[q,j]*u[q,i]/u[q,q] to factorize the
      sparse rows of A * A transpose.
NOTE: Bound feasibility attained by iteration 1.
NOTE: Dual feasibility attained by iteration 1.
NOTE: Constraint feasibility attained by iteration 2.
NOTE: Primal-Dual Predictor-Corrector Interior point
      algorithm performed 7 iterations.
NOTE: Objective = 50075.
NOTE: The data set WORK.SOLUTION has 18 observations and
      14 variables.
NOTE: There were 18 observations read from the data set
      WORK.ARCD1.
NOTE: There were 6 observations read from the data set
      WORK.MISS_S_X.
NOTE: There were 4 observations read from the data set
      WORK.COND1.
NOTE: The data set WORK.SOLUTION has 18 observations and
      14 variables.
```

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

Balancing Total Supply and Total Demand

When Total Supply Exceeds Total Demand

When total supply of a network problem exceeds total demand, PROC INTPOINT adds an extra node (called the *excess node*) to the problem and sets the demand at that node equal to the difference between total supply and total demand. There are three ways that this excess node can be joined to the network. All three ways entail PROC INTPOINT generating a set of arcs (henceforth referred to as the *generated* arcs) that are directed toward the excess node. The total amount of flow in generated arcs equals the demand of the excess node. The generated arcs originate from one of three sets of nodes.

When you specify the THRUNET option, the set of nodes that generated arcs originate from are all demand nodes, even those demand nodes with unspecified demand capability. You indicate that a node has unspecified demand capability by using a missing D value instead of an actual value for demand data (discussed in the "Missing S Supply and Missing D Demand Values" section on page 139). The value specified as the demand of a demand node is in effect a lower bound of the number of flow units that node can actually demand. For missing D demand nodes, this lower bound is zero.

If you do not specify the THRUNET option, the way in which the excess node is joined to the network depends on whether there are demand nodes with unspecified demand capability (nodes with missing D demand) or not.

If there are missing D demand nodes, these nodes are the set of nodes that generated arcs originate from. The value specified as the demand of a demand node, if not missing D, is the number of flow units that node can actually demand. For a missing D demand node, the actual demand of that node may be zero or greater.

If there are no missing D demand nodes, the set of nodes that generated arcs originate from are the set of supply nodes. The value specified as the supply of a supply node is in effect an upper bound of the number of flow units that node can actually supply. For missing S supply nodes (discussed in the "Missing S Supply and Missing D Demand Values" section on page 139), this upper bound is zero, so missing S nodes when total supply exceeds total demand are transhippment nodes, that is, nodes that neither supply nor demand flow.

When Total Supply Is Less Than Total Demand

When total supply of a network problem is less than total demand, PROC INTPOINT adds an extra node (called the *excess node*) to the problem and sets the supply at that node equal to the difference between total demand and total supply. There are three ways that this excess node can be joined to the network. All three ways entail PROC INTPOINT generating a set of arcs (henceforth referred to as the *generated* arcs) that originate from the excess node. The total amount of flow in generated arcs equals the supply of the excess node. The generated arcs are directed toward one of three sets of nodes.

When you specify the THRUNET option, the set of nodes that generated arcs are directed toward are all supply nodes, even those supply nodes with unspecified supply capability. You indicate that a node has unspecified supply capability by using a missing S value instead of an actual value for supply data (discussed in the "Missing S Supply and Missing D Demand Values" section on page 139). The value specified as the supply of a supply node is in effect a lower bound of the number of flow units that the node can actually supply. For missing S supply nodes, this lower bound is zero.

If you do not specify the THRUNET option, the way in which the excess node is joined to the network depends on whether there are supply nodes with unspecified supply capability (nodes with missing S supply) or not.

If there are missing S supply nodes, these nodes are the set of nodes that generated arcs are directed toward. The value specified as the supply of a supply node, if not missing S, is the number of flow units that the node can actually supply. For a missing S supply node, the actual supply of that node may be zero or greater.

If there are no missing S supply nodes, the set of nodes that generated arcs are directed toward are the set of demand nodes. The value specified as the demand of a demand node is in effect an upper bound of the number of flow units that node can actually demand. For missing D demand nodes (discussed in the "Missing S Supply and Missing D Demand Values" section on page 139), this upper bound is zero, so missing D nodes when total supply is less than total demand are transhippment nodes, that is, nodes that neither supply nor demand flow.

How to Make the Data Read of PROC INTPOINT More Efficient

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

Large Constrained Network Problems

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

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

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

Paging

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

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

The Order of Observations

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

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

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

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

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

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

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

Better Memory Utilization

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

These are the problem size options:

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

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

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

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

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

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

Use the memory usage parameters:

- BYTES= size of PROC INTPOINT main working memory in number of bytes
- MEMREP indicates that memory usage report is to be displayed on the SAS log

Specifying an appropriate value for the BYTES= parameter is particularly important. Specify as large a number as possible, but not so large a number that will cause PROC INTPOINT (that is, the SAS System running underneath PROC INTPOINT) to run out of memory.

PROC INTPOINT reports its memory requirements on the SAS log if you specify the MEMREP option.

Use Defaults to Reduce the Amount of Data

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

- DEFCOST= default cost of arcs, objective function of nonarc variables or LP variables
- DEFMINFLOW= default lower flow bound of arcs, lower bound of nonarc variables or LP variables
- DEFCAPACITY= default capacity of arcs, upper bound of nonarc variables or LP variables
- DEFCONTYPE= LE or DEFCONTYPE= <=
  DEFCONTYPE= EQ or DEFCONTYPE= =
  DEFCONTYPE= GE or DEFCONTYPE= >=

DEFCONTYPE=LE is the default.

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

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

- SOURCE= name of single node that has supply capability
- SUPPLY= the amount of supply of SOURCE

- SINK= name of single node that demands flow
- DEMAND= the amount of flow SINK demands

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

Names of Things

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

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

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

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

Other Ways to Speed-up Data Reads

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

Stopping Criteria

There are several reasons why PROC INTPOINT stops Interior Point optimization. Optimization stops when:

- the number of iteration equals MAXITERB=m
- the relative gap ($dualitygap/(c^T x)$) between the primal and dual objectives is smaller than the value of the PDGAPTOL= option, and both the primal and dual problems are feasible. Duality gap is defined in the "Interior Point Algorithmic Details" section on page 58.

PROC INTPOINT may stop optimization when it detects that the rate at which the complementarity or duality gap is being reduced is too slow; that is, that there are consecutive iterations when the complementarity or duality gap has stopped getting smaller and the infeasibilities, if nonzero, have also stalled. Sometimes this indicates that the problem is infeasible.

The reasons to stop optimization outlined in the previous paragraph will be termed the *usual* stopping conditions in the following explanation.

However, when solving some problems, especially if the problems are large, the usual stopping criteria are inappropriate. PROC INTPOINT might stop optimizing prematurely. If it were allowed to perform additional optimization, a better solution would be found. On other occasions, PROC INTPOINT might do too much work. A sufficiently good solution might be reached several iterations before PROC INTPOINT eventually stops.

You can see PROC INTPOINT's progress to the optimum by specifying PRINTLEVEL2=2. PROC INTPOINT will produce a table on the SAS log. A row of the table is generated during each iteration and consists of values of the affine step complementarity, the complementarity of the solution for the next iteration, the total bound infeasibility $\sum_{i=1}^{n} infeas_{bi}$ (see the $infeas_b$ array in the "Interior Point: Upper Bounds" section on page 63), and the total constraint infeasibility $\sum_{i=1}^{m} infeas_{ci}$ (see the $infeas_c$ array in the "Interior Point Algorithmic Details" section on page 58), the total dual infeasibility $\sum_{i=1}^{n} infeas_{di}$ (see the $infeas_d$ array in the "Interior Point Algorithmic Details" section on page 58). As optimization progresses, the values in all columns should converge to zero.

To tailor stopping criteria to your problem, you can use two sets of parameters: the STOP_x and the KEEPGOING_x parameters. The STOP_x parameters (STOP_C, STOP_DG, STOP_IB, STOP_IC, STOP_ID) are used to test for some condition at the beginning of each iteration and if met, to stop optimizing immediately. The KEEPGOING_x parameters (KEEPGOING_C, KEEPGOING_DG, KEEPGOING_IB, KEEPGOING_IC, KEEPGOING_ID) are used when PROC INTPOINT would ordinarily stop optimizing but does not if some conditions are not met.

For the sake of conciseness, a set of options might be referred to as the part of the option name they have in common followed by the suffix x. For example, STOP_C, STOP_DG, STOP_IB, STOP_IC, and STOP_ID will collectively be referred to as STOP_x.

At the beginning of each iteration, PROC INTPOINT will test whether complementarity is <= STOP_C (provided you specified a STOP_C parameter) and if it is, PROC INTPOINT will stop optimizing. If the duality gap is <= STOP_DG (provided you specified a STOP_DG parameter), PROC INTPOINT will stop optimizing immediately. This is true as well for the other STOP_x parameters that are related to infeasibilities, STOP_IB, STOP_IC, and STOP_ID.

For example, if you want PROC INTPOINT to stop optimizing for the usual stopping conditions, plus the additional condition, complementarity <= 100 *or* duality gap <= 0.001, then use:

```
proc intpoint stop_c=100 stop_dg=0.001
```

If you want PROC INTPOINT to stop optimizing for the usual stopping conditions, plus the additional condition, complementarity <= 1000 *and* duality gap <= 0.001 *and* constraint infeasibility <= 0.0001, then use:

```
proc intpoint
   and_stop_c=1000 and_stop_dg=0.01 and_stop_ic=0.0001
```

Unlike the STOP_x parameters that cause PROC INTPOINT to stop optimizing when any one of them is satisfied, the corresponding AND_STOP_x parameters (AND_STOP_C, AND_STOP_DG, AND_STOP_IB, AND_STOP_IC, and AND_STOP_ID) cause PROC INTPOINT to stop only if all (more precisely, all that are specified) options are satisfied. For example, if PROC INTPOINT should stop optimizing when

- complementarity <= 100 *or* duality gap <= 0.001 or
- complementarity <= 1000 *and* duality gap <= 0.001 *and* constraint infeasibility <= 0.000

then use:

```
proc intpoint
   stop_c=100 stop_dg=0.001
   and_stop_c=1000 and_stop_dg=0.01 and_stop_ic=0.0001
```

Just as the STOP_x parameters have AND_STOP_x partners, the KEEPGOING_x parameters have AND_KEEPGOING_x partners. The role of the KEEPGOING_x and AND_KEEPGOING_x parameters is to prevent optimization from stopping too early, even though a usual stopping criteria is met.

When PROC INTPOINT detects that it should stop optimizing for a usual stopping condition,

- it will test whether complementarity is > KEEPGOING_C (provided you specified a KEEPGOING_C parameter), and if it is, PROC INTPOINT will perform more optimization.

- Otherwise, PROC INTPOINT will then test whether the primal-dual gap is $>$ KEEPGOING_DG (provided you specified a KEEPGOING_DG parameter), and if it is, PROC INTPOINT will perform more optimization.
- Otherwise, PROC INTPOINT will then test whether the total bound infeasibility $\sum_{i=1}^{n} infeas_{bi} >$ KEEPGOING_IB (provided you specified a KEEPGOING_IB parameter), and if it is, PROC INTPOINT will perform more optimization.
- Otherwise, PROC INTPOINT will then test whether the total constraint infeasibility $\sum_{i=1}^{m} infeas_{ci} >$ KEEPGOING_IC (provided you specified a KEEPGOING_IC parameter), and if it is, PROC INTPOINT will perform more optimization.
- Otherwise, PROC INTPOINT will then test whether the total dual infeasibility $\sum_{i=1}^{n} infeas_{di} >$ KEEPGOING_ID (provided you specified a KEEPGOING_ID parameter), and if it is, PROC INTPOINT will perform more optimization.
- Otherwise it will test whether complementarity is $>$ AND_KEEPGOING_C (provided you specified an AND_KEEPGOING_C parameter), *and* the primal-dual gap is $>$ AND_KEEPGOING_DG (provided you specified an AND_KEEPGOING_DG parameter), *and* the total bound infeasibility $\sum_{i=1}^{n} infeas_{bi} >$ AND_KEEPGOING_IB (provided you specified an AND_KEEPGOING_IB parameter), *and* the total constraint infeasibility $\sum_{i=1}^{m} infeas_{ci} >$ AND_KEEPGOING_IC (provided you specified an AND_KEEPGOING_IC parameter), *and* the total dual infeasibility $\sum_{i=1}^{n} infeas_{di} >$ AND_KEEPGOING_ID (provided you specified an AND_KEEPGOING_ID parameter), and if it is, PROC INTPOINT will perform more optimization.

If all these tests to decide whether more optimization should be performed are false, optimization is stopped.

The following PROC INTPOINT example will be used to illustrate how several stopping criteria options can be used together:

```
proc intpoint
   stop_c=1000
   and_stop_c=2000 and_stop_dg=0.01
   and_stop_ib=1 and_stop_ic=1 and_stop_id=1
   keepgoing_c=1500
   and_keepgoing_c=2500 and_keepgoing_dg=0.05
   and_keepgoing_ib=1 and_keepgoing_ic=1 and_keepgoing_id=1
```

At the beginning of each iteration, PROC INTPOINT will stop optimizing if

- complementarity $<= 1000$ or
- complementarity $<= 2000$ and duality gap $<= 0.01$ and the total bound, constraint, and dual infeasibilities are each $<= 1$

When PROC INTPOINT determines it should stop optimizing because a usual stopping condition is met, it will stop optimizing only if

- complementarity $<= 1500$ or
- complementarity $<= 2500$ and duality gap $<= 0.05$ and the total bound, constraint, and dual infeasibilities are each $<= 1$

Examples

The following examples illustrate some of the capabilities of PROC INTPOINT. These examples, together with the other SAS/OR examples, can be found in the SAS sample library.

In order to illustrate variations in the use of the INTPOINT procedure, Example 4.1 through Example 4.5 use data from a company that produces two sizes of televisions. The company makes televisions with a diagonal screen measurement of either 19 inches or 25 inches. These televisions are made between March and May at both of the company's two factories. Each factory has a limit on the total number of televisions of each screen dimension that can be made during those months.

The televisions are distributed to one of two shops, stored at the factory where they were made, and sold later or shipped to the other factory. Some sets can be used to fill back-orders from the previous months. Each shop demands a number of each type of TV for the months March through May. The following network in Figure 4.12 illustrates the model. Arc costs can be interpreted as production costs, storage costs, back-order penalty costs, inter-factory transportation costs, and sales profits. The arcs can have capacities and lower flow bounds.

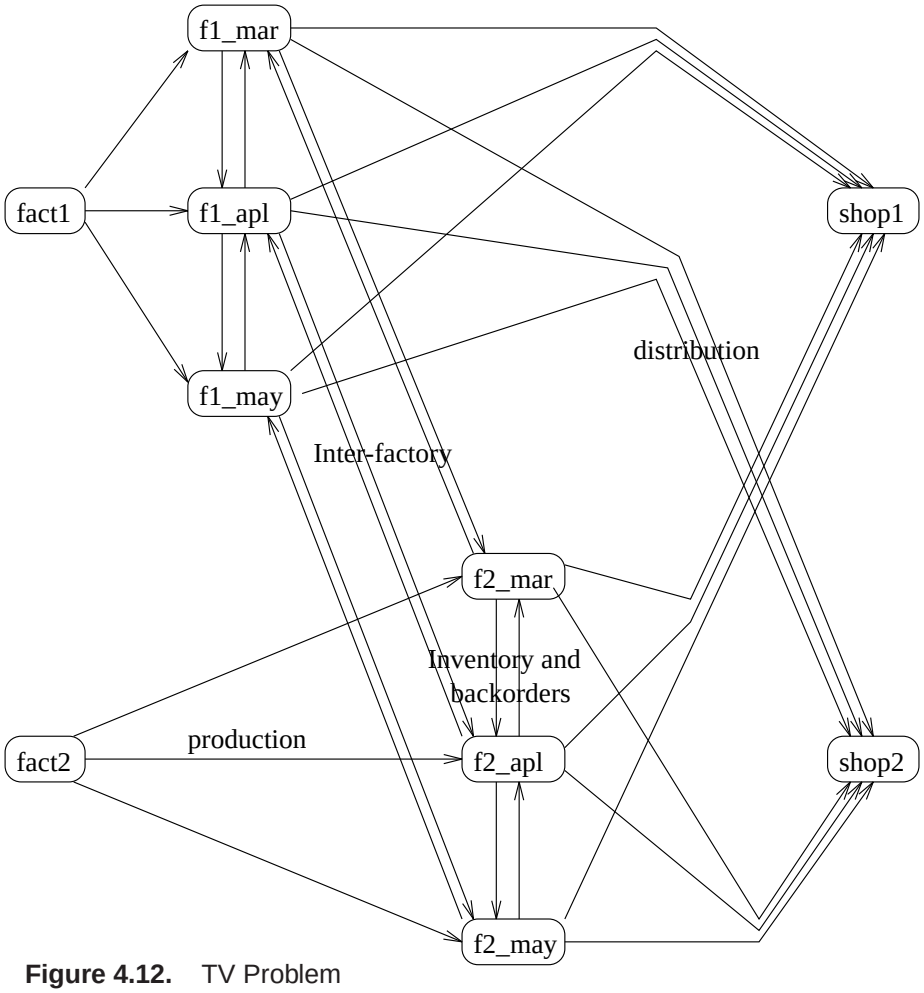


Figure 4.12. TV Problem

There are two similarly structured networks, one for the 19-inch televisions and the other for the 25-inch screen TVs. The minimum cost production, inventory, and distribution plan for both TV types can be determined in the same run of PROC INTPOINT. To ensure that node names are unambiguous, the names of nodes in the 19-inch network have suffix _1, and the node names in the 25-inch network have suffix _2.

Example 4.1. Production, Inventory, Distribution Problem

The following code shows how to save a specific problem's data in data sets and solve the model with PROC INTPOINT:

```
title 'Production Planning/Inventory/Distribution';
title2 'Minimum Cost Flow problem';
title3;
data node0;
   input _node_ $ _supdem_ ;
   datalines;
fact1_1   1000
fact2_1    850
fact1_2   1000
fact2_2   1500
shop1_1   -900
shop2_1   -900
shop1_2   -900
shop2_2  -1450
;
data arc0;
   input _tail_ $ _head_ $ _cost_ _capac_ _lo_ diagonal factory
         key_id $10. mth_made $ _name_&$17. ;
   datalines;
fact1_1  f1_mar_1  127.9  500 50 19 1 production March prod f1 19 mar
fact1_1  f1_apr_1   78.6  600 50 19 1 production April prod f1 19 apl
fact1_1  f1_may_1   95.1  400 50 19 1 production May   .
f1_mar_1 f1_apr_1   15     50  . 19 1 storage    March .
f1_apr_1 f1_may_1   12     50  . 19 1 storage    April .
f1_apr_1 f1_mar_1   28     20  . 19 1 backorder  April back f1 19 apl
f1_may_1 f1_apr_1   28     20  . 19 1 backorder  May   back f1 19 may
f1_mar_1 f2_mar_1   11      .  . 19 . f1_to_2    March .
f1_apr_1 f2_apr_1   11      .  . 19 . f1_to_2    April .
f1_may_1 f2_may_1   16      .  . 19 . f1_to_2    May   .
f1_mar_1 shop1_1  -327.65 250  . 19 1 sales      March .
f1_apr_1 shop1_1  -300    250  . 19 1 sales      April .
f1_may_1 shop1_1  -285    250  . 19 1 sales      May   .
f1_mar_1 shop2_1  -362.74 250  . 19 1 sales      March .
f1_apr_1 shop2_1  -300    250  . 19 1 sales      April .
f1_may_1 shop2_1  -245    250  . 19 1 sales      May   .
fact2_1  f2_mar_1   88.0  450 35 19 2 production March prod f2 19 mar
fact2_1  f2_apr_1   62.4  480 35 19 2 production April prod f2 19 apl
fact2_1  f2_may_1  133.8  250 35 19 2 production May   .
f2_mar_1 f2_apr_1   18     30  . 19 2 storage    March .
f2_apr_1 f2_may_1   20     30  . 19 2 storage    April .
f2_apr_1 f2_mar_1   17     15  . 19 2 backorder  April back f2 19 apl
f2_may_1 f2_apr_1   25     15  . 19 2 backorder  May   back f2 19 may
f2_mar_1 f1_mar_1   10     40  . 19 . f2_to_1    March .
f2_apr_1 f1_apr_1   11     40  . 19 . f2_to_1    April .
```

```
f2_may_1 f1_may_1   13     40  . 19 . f2_to_1    May   .
f2_mar_1 shop1_1  -297.4  250  . 19 2 sales      March .
f2_apr_1 shop1_1  -290    250  . 19 2 sales      April .
f2_may_1 shop1_1  -292    250  . 19 2 sales      May   .
f2_mar_1 shop2_1  -272.7  250  . 19 2 sales      March .
f2_apr_1 shop2_1  -312    250  . 19 2 sales      April .
f2_may_1 shop2_1  -299    250  . 19 2 sales      May   .
fact1_2  f1_mar_2  217.9  400 40 25 1 production March prod f1 25 mar
fact1_2  f1_apr_2  174.5  550 50 25 1 production April prod f1 25 apl
fact1_2  f1_may_2  133.3  350 40 25 1 production May   .
f1_mar_2 f1_apr_2   20     40  . 25 1 storage    March .
f1_apr_2 f1_may_2   18     40  . 25 1 storage    April .
f1_apr_2 f1_mar_2   32     30  . 25 1 backorder  April back f1 25 apl
f1_may_2 f1_apr_2   41     15  . 25 1 backorder  May   back f1 25 may
f1_mar_2 f2_mar_2   23      .  . 25 . f1_to_2    March .
f1_apr_2 f2_apr_2   23      .  . 25 . f1_to_2    April .
f1_may_2 f2_may_2   26      .  . 25 . f1_to_2    May   .
f1_mar_2 shop1_2  -559.76   .  . 25 1 sales      March .
f1_apr_2 shop1_2  -524.28   .  . 25 1 sales      April .
f1_may_2 shop1_2  -475.02   .  . 25 1 sales      May   .
f1_mar_2 shop2_2  -623.89   .  . 25 1 sales      March .
f1_apr_2 shop2_2  -549.68   .  . 25 1 sales      April .
f1_may_2 shop2_2  -460.00   .  . 25 1 sales      May   .
fact2_2  f2_mar_2  182.0  650 35 25 2 production March prod f2 25 mar
fact2_2  f2_apr_2  196.7  680 35 25 2 production April prod f2 25 apl
fact2_2  f2_may_2  201.4  550 35 25 2 production May   .
f2_mar_2 f2_apr_2   28     50  . 25 2 storage    March .
f2_apr_2 f2_may_2   38     50  . 25 2 storage    April .
f2_apr_2 f2_mar_2   31     15  . 25 2 backorder  April back f2 25 apl
f2_may_2 f2_apr_2   54     15  . 25 2 backorder  May   back f2 25 may
f2_mar_2 f1_mar_2   20     25  . 25 . f2_to_1    March .
f2_apr_2 f1_apr_2   21     25  . 25 . f2_to_1    April .
f2_may_2 f1_may_2   43     25  . 25 . f2_to_1    May   .
f2_mar_2 shop1_2  -567.83 500  . 25 2 sales      March .
f2_apr_2 shop1_2  -542.19 500  . 25 2 sales      April .
f2_may_2 shop1_2  -461.56 500  . 25 2 sales      May   .
f2_mar_2 shop2_2  -542.83 500  . 25 2 sales      March .
f2_apr_2 shop2_2  -559.19 500  . 25 2 sales      April .
f2_may_2 shop2_2  -489.06 500  . 25 2 sales      May   .
;
proc intpoint
   bytes=1000000
   printlevel2=2
   nodedata=node0
   arcdata=arc0
   conout=arc1;
run;
```

```
proc print data=arc1;
   var _from_ _to_ _cost_ _capac_ _lo_ _flow_ _fcost_
       diagonal factory key_id mth_made;
   sum _fcost_;
run;
```

The following notes appear on the SAS log:

```
NOTE: Number of nodes= 20 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 4350 , total demand= 4150 .
NOTE: Number of arcs= 64 .
NOTE: The following messages relate to the equivalent
      Linear Program solved by the Interior Point
      algorithm.
NOTE: Number of <= constraints= 0 .
NOTE: Number of == constraints= 21 .
NOTE: Number of >= constraints= 0 .
NOTE: Number of constraint coefficients= 136 .
NOTE: Number of variables= 68 .
NOTE: 0 columns, 0 rows and 0 coefficients were added to
      the problem to handle unrestricted variables,
      variables that are split, and constraint slack or
      surplus variables.
NOTE: There are 54 nonzero elements in A * A transpose.
NOTE: Of the 21 rows and columns, 14 are sparse.
NOTE: There are 59 nonzero superdiagonal elements in the
      sparse rows of the factored A * A transpose. This
      includes fill-in.
NOTE: There are 80 operations of the form
      u[i,j]=u[i,j]-u[q,j]*u[q,i]/u[q,q] to factorize the
      sparse rows of A * A transpose.
Iter Complem_aff Complem-ity  Duality_gap Tot_infeasb  Tot_infeasc Tot_infeasd
   0   -1.000000   167086681     0.835362       52835        26238       34922
   1    35446536    17335378     0.912917 1663.290349   825.996205           0
   2     1957002      878825     0.416636           0 1.956907E-12           0
   3      550439      237247     0.155174           0            0           0
   4      126652       53958     0.039021           0            0           0
   5       27526       18386     0.013499           0            0           0
   6 9153.811245 3029.529789     0.002242           0            0           0
   7  905.846996  614.526357     0.000455           0            0           0
   8  219.526873   73.592013  0.000054542           0            0           0
   9   11.383058    2.111334  0.000001565           0            0           0
  10    0.089558    0.000361 2.672549E-10           0            0           0
NOTE: Primal-Dual Predictor-Corrector Interior point
      algorithm performed 10 iterations.
NOTE: Objective = -1281110.35.
NOTE: The data set WORK.ARC1 has 64 observations and 18
      variables.
NOTE: There were 64 observations read from the data set
      WORK.ARC0.
NOTE: There were 8 observations read from the data set
      WORK.NODE0.
NOTE: The data set WORK.ARC1 has 64 observations and 18
      variables.
```

The solution is given in the CONOUT=arc1 data sets. In the CONOUT= data set, shown in Output 4.1.1, the variables diagonal, factory, key_id, and mth_made form an implicit ID list. The diagonal variable has one of two values, 19 or 25. factory also has one of two values, 1 or 2, to denote the factory where either production or storage occurs, from where TVs are either sold to shops or satisfy back-orders. production, storage, sales, and backorder are values of the key_id variable.

Other values of this variable, f1_to_2 and f2_to_1, are used when flow through arcs represents the transportation of TVs between factories. The mth_made variable has values March, April, and May, the months when TVs that are modeled as flow through an arc were made (assuming that no televisions are stored for more than one month and none manufactured in May are used to fill March back-orders).

These ID variables can be used after the PROC INTPOINT run to produce reports and perform analysis on particular parts of the company's operation. For example, reports can be generated for production numbers for each factory; optimal sales figures for each shop; and how many TVs should be stored, used to fill back-orders, sent to the other factory, or any combination of these, for TVs with a particular screen, those produced in a particular month, or both.

Output 4.1.1. CONOUT=ARC1

_tail	_head	_cost	_capa_lo	_FLOW	_FCOST	diagonal	factory	key_id	mth_made
fact1_1	f1_apr_1	78.6	600 50	600.000	47160.00	19	1	production	April
f1_mar_1	f1_apr_1	15.0	50 0	0.000	0.00	19	1	storage	March
f1_may_1	f1_apr_1	28.0	20 0	0.000	0.00	19	1	backorder	May
f2_apr_1	f1_apr_1	11.0	40 0	0.000	0.00	19	.	f2_to_1	April
fact1_2	f1_apr_2	174.5	550 50	550.000	95975.00	25	1	production	April
f1_mar_2	f1_apr_2	20.0	40 0	0.000	0.00	25	1	storage	March
f1_may_2	f1_apr_2	41.0	15 0	15.000	615.00	25	1	backorder	May
f2_apr_2	f1_apr_2	21.0	25 0	0.000	0.00	25	.	f2_to_1	April
fact1_1	f1_mar_1	127.9	500 50	345.000	44125.49	19	1	production	March
f1_apr_1	f1_mar_1	28.0	20 0	20.000	560.00	19	1	backorder	April
f2_mar_1	f1_mar_1	10.0	40 0	40.000	400.00	19	.	f2_to_1	March
fact1_2	f1_mar_2	217.9	400 40	400.000	87160.00	25	1	production	March
f1_apr_2	f1_mar_2	32.0	30 0	30.000	960.00	25	1	backorder	April
f2_mar_2	f1_mar_2	20.0	25 0	25.000	500.00	25	.	f2_to_1	March
fact1_1	f1_may_1	95.1	400 50	50.000	4755.01	19	1	production	May
f1_apr_1	f1_may_1	12.0	50 0	50.000	600.00	19	1	storage	April
f2_may_1	f1_may_1	13.0	40 0	0.000	0.00	19	.	f2_to_1	May
fact1_2	f1_may_2	133.3	350 40	40.000	5332.00	25	1	production	May
f1_apr_2	f1_may_2	18.0	40 0	0.000	0.00	25	1	storage	April
f2_may_2	f1_may_2	43.0	25 0	0.000	0.00	25	.	f2_to_1	May
f1_apr_1	f2_apr_1	11.0	99999999 0	30.000	330.00	19	.	f1_to_2	April
fact2_1	f2_apr_1	62.4	480 35	480.000	29952.00	19	2	production	April
f2_mar_1	f2_apr_1	18.0	30 0	0.000	0.00	19	2	storage	March
f2_may_1	f2_apr_1	25.0	15 0	0.000	0.00	19	2	backorder	May
f1_apr_2	f2_apr_2	23.0	99999999 0	0.000	0.00	25	.	f1_to_2	April
fact2_2	f2_apr_2	196.7	680 35	680.000	133756.00	25	2	production	April
f2_mar_2	f2_apr_2	28.0	50 0	0.000	0.00	25	2	storage	March
f2_may_2	f2_apr_2	54.0	15 0	15.000	810.00	25	2	backorder	May
f1_mar_1	f2_mar_1	11.0	99999999 0	0.000	0.00	19	.	f1_to_2	March
fact2_1	f2_mar_1	88.0	450 35	290.000	25520.00	19	2	production	March
f2_apr_1	f2_mar_1	17.0	15 0	0.000	0.00	19	2	backorder	April
f1_mar_2	f2_mar_2	23.0	99999999 0	0.000	0.00	25	.	f1_to_2	March

tail	_head_	_cost_	_capac_	_l_o_	_FLOW_	_FCOST_	diag	afac	key_id	mth_made
fact2_2	f2_mar_2	182.00	650	35	645.000	117390.00	25	2	production	March
f2_apr_2	f2_mar_2	31.00	15	0	0.000	0.00	25	2	backorder	April
f1_may_1	f2_may_1	16.00	99999999	0	100.000	1600.00	19	.	f1_to_2	May
fact2_1	f2_may_1	133.80	250	35	35.000	4683.00	19	2	production	May
f2_apr_1	f2_may_1	20.00	30	0	15.000	300.00	19	2	storage	April
f1_may_2	f2_may_2	26.00	99999999	0	0.000	0.00	25	.	f1_to_2	May
fact2_2	f2_may_2	201.40	550	35	35.000	7049.00	25	2	production	May
f2_apr_2	f2_may_2	38.00	50	0	0.000	0.00	25	2	storage	April
f1_mar_1	shop1_1	-327.65	250	0	155.000	-50785.73	19	1	sales	March
f1_apr_1	shop1_1	-300.00	250	0	250.000	-75000.00	19	1	sales	April
f1_may_1	shop1_1	-285.00	250	0	0.000	0.00	19	1	sales	May
f2_mar_1	shop1_1	-297.40	250	0	250.000	-74350.00	19	2	sales	March
f2_apr_1	shop1_1	-290.00	250	0	245.000	-71050.02	19	2	sales	April
f2_may_1	shop1_1	-292.00	250	0	0.000	0.00	19	2	sales	May
f1_mar_2	shop1_2	-559.76	99999999	0	0.000	0.00	25	1	sales	March
f1_apr_2	shop1_2	-524.28	99999999	0	0.000	0.00	25	1	sales	April
f1_may_2	shop1_2	-475.02	99999999	0	25.000	-11875.50	25	1	sales	May
f2_mar_2	shop1_2	-567.83	500	0	500.000	-283915.00	25	2	sales	March
f2_apr_2	shop1_2	-542.19	500	0	375.000	-203321.25	25	2	sales	April
f2_may_2	shop1_2	-461.56	500	0	0.000	0.00	25	2	sales	May
f1_mar_1	shop2_1	-362.74	250	0	250.000	-90685.00	19	1	sales	March
f1_apr_1	shop2_1	-300.00	250	0	250.000	-75000.00	19	1	sales	April
f1_may_1	shop2_1	-245.00	250	0	0.000	0.00	19	1	sales	May
f2_mar_1	shop2_1	-272.70	250	0	0.000	0.00	19	2	sales	March
f2_apr_1	shop2_1	-312.00	250	0	250.000	-78000.00	19	2	sales	April
f2_may_1	shop2_1	-299.00	250	0	150.000	-44850.00	19	2	sales	May
f1_mar_2	shop2_2	-623.89	99999999	0	455.000	-283869.95	25	1	sales	March
f1_apr_2	shop2_2	-549.68	99999999	0	535.000	-294078.80	25	1	sales	April
f1_may_2	shop2_2	-460.00	99999999	0	0.000	0.00	25	1	sales	May
f2_mar_2	shop2_2	-542.83	500	0	120.000	-65139.60	25	2	sales	March
f2_apr_2	shop2_2	-559.19	500	0	320.000	-178940.80	25	2	sales	April
f2_may_2	shop2_2	-489.06	500	0	20.000	-9781.20	25	2	sales	May

Example 4.2. Altering Arc Data

This example examines the effect of changing some of the arc costs. The backorder penalty costs are increased by 20 percent. The sales profit of 25-inch TVs sent to the shops in May is increased by 30 units. The backorder penalty costs of 25-inch TVs manufactured in May for April consumption is decreased by 30 units. The production cost of 19-inch and 25-inch TVs made in May are decreased by 5 units and 20 units, respectively. How does the optimal solution of the network after these arc cost alterations compare with the optimum of the original network?

These SAS statements produce the new NODEDATA= and ARCDATA= data sets:

```
title2 'Minimum Cost Flow problem- Altered Arc Data';
data arc2;
   set arc1;
      oldcost=_cost_;
      oldfc=_fcost_;
      oldflow=_flow_;
      if key_id='backorder'
         then _cost_=_cost_*1.2;
         else if _tail_='f2_may_2' then _cost_=_cost_-30;
      if key_id='production' & mth_made='May' then
         if diagonal=19 then _cost_=_cost_-5;
                        else _cost_=_cost_-20;
```

```
proc intpoint
   bytes=100000
   printlevel2=2
   nodedata=node0
   arcdata=arc2
   conout=arc3;
run;

proc print data=arc3;
   var _tail_ _head_ _capac_ _lo_ _supply_ _demand_ _name_
       _cost_ _flow_ _fcost_ oldcost oldflow oldfc
       diagonal factory key_id mth_made;
       /* to get this variable order */
           sum oldfc _fcost_;
run;
```

The following notes appear on the SAS log:

```
NOTE: Number of nodes= 20 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 4350 , total demand= 4150 .
NOTE: Number of arcs= 64 .
NOTE: The following messages relate to the equivalent Linear
      Program solved by the Interior Point algorithm.
NOTE: Number of <= constraints= 0 .
NOTE: Number of == constraints= 21 .
NOTE: Number of >= constraints= 0 .
NOTE: Number of constraint coefficients= 136 .
NOTE: Number of variables= 68 .
NOTE: 0 columns, 0 rows and 0 coefficients were added to
      the problem to handle unrestricted variables,
      variables that are split, and constraint slack or
      surplus variables.
NOTE: There are 54 nonzero elements in A * A transpose.
NOTE: Of the 21 rows and columns, 14 are sparse.
NOTE: There are 59 nonzero superdiagonal elements in the
      sparse rows of the factored A * A transpose. This
      includes fill-in.
NOTE: There are 80 operations of the form
      u[i,j]=u[i,j]-u[q,j]*u[q,i]/u[q,q] to factorize the
      sparse rows of A * A transpose.
Iter Complem_aff Complem-ity  Duality_gap Tot_infeasb  Tot_infeasc Tot_infeasd
   0   -1.000000   167898972     0.834344       52835        26238       35092
   1    35575010    17420416     0.911474 1664.001693   826.349461           0
   2     1969763      739523     0.369979           0 1.267542E-12           0
   3      373828      189905     0.127240           0            0           0
   4      117843       45042     0.032721           0            0           0
   5       26090       15668     0.011493           0            0           0
   6 8608.105391 2994.589912     0.002212           0            0           0
   7 1477.385197  411.268625     0.000304           0            0           0
   8  103.724839   24.733676  0.000018304           0            0           0
   9    2.140932    0.002805 2.0758339E-9           0            0           0
NOTE: Primal-Dual Predictor-Corrector Interior point algorithm
      performed 9 iterations.
NOTE: Objective = -1285086.45.
NOTE: The data set WORK.ARC3 has 64 observations and 21
      variables.
```

```
NOTE: There were 64 observations read from the data set
      WORK.ARC2.
NOTE: There were 8 observations read from the data set
      WORK.NODE0.
NOTE: The data set WORK.ARC3 has 64 observations and 21
      variables.
```

The solution is displayed in Output 4.2.1.

Output 4.2.1. CONOUT=ARC3

tail	_head_	_capac_	_lo_	_SUPPLY_	_DEMAND_	_name_	_cost_	_FLOW_
fact1_1	f1_apr_1	600	50	1000	.	prod f1 19 apl	78.6	540.000
f1_mar_1	f1_apr_1	50	0	.	.		15.0	0.000
f1_may_1	f1_apr_1	20	0	.	.	back f1 19 may	33.6	0.000
f2_apr_1	f1_apr_1	40	0	.	.		11.0	0.000
fact1_2	f1_apr_2	550	50	1000	.	prod f1 25 apl	174.5	250.000
f1_mar_2	f1_apr_2	40	0	.	.		20.0	0.000
f1_may_2	f1_apr_2	15	0	.	.	back f1 25 may	49.2	15.000
f2_apr_2	f1_apr_2	25	0	.	.		21.0	0.000
fact1_1	f1_mar_1	500	50	1000	.	prod f1 19 mar	127.9	340.000
f1_apr_1	f1_mar_1	20	0	.	.	back f1 19 apl	33.6	20.000
f2_mar_1	f1_mar_1	40	0	.	.		10.0	40.000
fact1_2	f1_mar_2	400	40	1000	.	prod f1 25 mar	217.9	400.000
f1_apr_2	f1_mar_2	30	0	.	.	back f1 25 apl	38.4	30.000
f2_mar_2	f1_mar_2	25	0	.	.		20.0	25.000
fact1_1	f1_may_1	400	50	1000	.		90.1	115.000
f1_apr_1	f1_may_1	50	0	.	.		12.0	0.000
f2_may_1	f1_may_1	40	0	.	.		13.0	0.000
fact1_2	f1_may_2	350	40	1000	.		113.3	350.000
f1_apr_2	f1_may_2	40	0	.	.		18.0	0.000
f2_may_2	f1_may_2	25	0	.	.		13.0	0.000
f1_apr_1	f2_apr_1	99999999	0	.	.		11.0	20.000
fact2_1	f2_apr_1	480	35	850	.	prod f2 19 apl	62.4	480.000
f2_mar_1	f2_apr_1	30	0	.	.		18.0	0.000
f2_may_1	f2_apr_1	15	0	.	.	back f2 19 may	30.0	0.000
f1_apr_2	f2_apr_2	99999999	0	.	.		23.0	0.000
fact2_2	f2_apr_2	680	35	1500	.	prod f2 25 apl	196.7	680.000
f2_mar_2	f2_apr_2	50	0	.	.		28.0	0.000

FCOST	oldcost	oldflow	oldfc	diagonal	factory	key_id	mth_made
42443.99	78.6	600.000	47160.00	19	1	production	April
0.00	15.0	0.000	0.00	19	1	storage	March
0.00	28.0	0.000	0.00	19	1	backorder	May
0.00	11.0	0.000	0.00	19	.	f2_to_1	April
43625.00	174.5	550.000	95975.00	25	1	production	April
0.00	20.0	0.000	0.00	25	1	storage	March
738.00	41.0	15.000	615.00	25	1	backorder	May
0.00	21.0	0.000	0.00	25	.	f2_to_1	April
43486.01	127.9	345.000	44125.49	19	1	production	March
672.00	28.0	20.000	560.00	19	1	backorder	April
400.00	10.0	40.000	400.00	19	.	f2_to_1	March
87160.00	217.9	400.000	87160.00	25	1	production	March
1152.00	32.0	30.000	960.00	25	1	backorder	April
500.00	20.0	25.000	500.00	25	.	f2_to_1	March
10361.50	95.1	50.000	4755.01	19	1	production	May
0.00	12.0	50.000	600.00	19	1	storage	April
0.00	13.0	0.000	0.00	19	.	f2_to_1	May
39655.00	133.3	40.000	5332.00	25	1	production	May
0.00	18.0	0.000	0.00	25	1	storage	April
0.00	43.0	0.000	0.00	25	.	f2_to_1	May
220.00	11.0	30.000	330.00	19	.	f1_to_2	April
29952.00	62.4	480.000	29952.00	19	2	production	April
0.00	18.0	0.000	0.00	19	2	storage	March
0.00	25.0	0.000	0.00	19	2	backorder	May
0.00	23.0	0.000	0.00	25	.	f1_to_2	April
133756.00	196.7	680.000	133756.00	25	2	production	April
0.00	28.0	0.000	0.00	25	2	storage	March

tail	_head_	_capac_	_lo_	_SUPPLY_	_DEMAND_	_name_	_cost_	_FLOW_
f2_may_2	f2_apr_2	15	0	.	.	back f2 25 may	64.8	0.000
f1_mar_1	f2_mar_1	99999999	0	.	.		11.0	0.000
fact2_1	f2_mar_1	450	35	850	.	prod f2 19 mar	88.0	290.000
f2_apr_1	f2_mar_1	15	0	.	.	back f2 19 apl	20.4	0.000
f1_mar_2	f2_mar_2	99999999	0	.	.		23.0	0.000

FCOST	oldcost	oldflow	oldfc	diagonal	factory	key_id	mth_made
0.00	54.0	15.000	810.00	25	2	backorder	May
0.00	11.0	0.000	0.00	19	.	f1_to_2	March
25520.00	88.0	290.000	25520.00	19	2	production	March
0.00	17.0	0.000	0.00	19	2	backorder	April
0.00	23.0	0.000	0.00	25	.	f1_to_2	March

tail	_head_	_capac_	_lo_	_SUPPLY_	_DEMAND_	_name_	_cost_	_FLOW_
fact2_2	f2_mar_2	650	35	1500	.	prod f2 25 mar	182.00	635.000
f2_apr_2	f2_mar_2	15	0	.	.	back f2 25 apl	37.20	0.000
f1_may_1	f2_may_1	99999999	0	.	.		16.00	115.000
fact2_1	f2_may_1	250	35	850	.		128.80	35.000
f2_apr_1	f2_may_1	30	0	.	.		20.00	0.000
f1_may_2	f2_may_2	99999999	0	.	.		26.00	335.000
fact2_2	f2_may_2	550	35	1500	.		181.40	35.000
f2_apr_2	f2_may_2	50	0	.	.		38.00	0.000
f1_mar_1	shop1_1	250	0	.	900		-327.65	150.000
f1_apr_1	shop1_1	250	0	.	900		-300.00	250.000
f1_may_1	shop1_1	250	0	.	900		-285.00	0.000
f2_mar_1	shop1_1	250	0	.	900		-297.40	250.000
f2_apr_1	shop1_1	250	0	.	900		-290.00	250.000
f2_may_1	shop1_1	250	0	.	900		-292.00	0.000
f1_mar_2	shop1_2	99999999	0	.	900		-559.76	0.000
f1_apr_2	shop1_2	99999999	0	.	900		-524.28	0.000
f1_may_2	shop1_2	99999999	0	.	900		-475.02	0.000
f2_mar_2	shop1_2	500	0	.	900		-567.83	500.000
f2_apr_2	shop1_2	500	0	.	900		-542.19	400.000
f2_may_2	shop1_2	500	0	.	900		-491.56	0.000
f1_mar_1	shop2_1	250	0	.	900		-362.74	250.000
f1_apr_1	shop2_1	250	0	.	900		-300.00	250.000
f1_may_1	shop2_1	250	0	.	900		-245.00	0.000
f2_mar_1	shop2_1	250	0	.	900		-272.70	0.000
f2_apr_1	shop2_1	250	0	.	900		-312.00	250.000
f2_may_1	shop2_1	250	0	.	900		-299.00	150.000
f1_mar_2	shop2_2	99999999	0	.	1450		-623.89	455.000

FCOST	oldcost	oldflow	oldfc	diagonal	factory	key_id	mth_made
115570.00	182.00	645.000	117390.00	25	2	production	March
0.00	31.00	0.000	0.00	25	2	backorder	April
1840.00	16.00	100.000	1600.00	19	.	f1_to_2	May
4508.00	133.80	35.000	4683.00	19	2	production	May
0.00	20.00	15.000	300.00	19	2	storage	April
8710.00	26.00	0.000	0.00	25	.	f1_to_2	May
6349.00	201.40	35.000	7049.00	25	2	production	May
0.00	38.00	0.000	0.00	25	2	storage	April
-49147.54	-327.65	155.000	-50785.73	19	1	sales	March
-75000.00	-300.00	250.000	-75000.00	19	1	sales	April
0.00	-285.00	0.000	0.00	19	1	sales	May
-74350.00	-297.40	250.000	-74350.00	19	2	sales	March
-72499.97	-290.00	245.000	-71050.02	19	2	sales	April
0.00	-292.00	0.000	0.00	19	2	sales	May
0.00	-559.76	0.000	0.00	25	1	sales	March
0.00	-524.28	0.000	0.00	25	1	sales	April
-0.02	-475.02	25.000	-11875.50	25	1	sales	May
-283915.00	-567.83	500.000	-283915.00	25	2	sales	March
-216875.98	-542.19	375.000	-203321.25	25	2	sales	April
0.00	-461.56	0.000	0.00	25	2	sales	May
-90685.00	-362.74	250.000	-90685.00	19	1	sales	March
-75000.00	-300.00	250.000	-75000.00	19	1	sales	April
0.00	-245.00	0.000	0.00	19	1	sales	May
0.00	-272.70	0.000	0.00	19	2	sales	March
-78000.00	-312.00	250.000	-78000.00	19	2	sales	April
-44850.00	-299.00	150.000	-44850.00	19	2	sales	May
-283869.95	-623.89	455.000	-283869.95	25	1	sales	March

tail	_head_	_capac_	_lo_	_SUPPLY_	_DEMAND_	_name_	_cost_	_FLOW_
f1_apr_2	shop2_2	99999999	0	.	1450		-549.68	235.000
f1_may_2	shop2_2	99999999	0	.	1450		-460.00	0.000
f2_mar_2	shop2_2	500	0	.	1450		-542.83	110.000
f2_apr_2	shop2_2	500	0	.	1450		-559.19	280.000
f2_may_2	shop2_2	500	0	.	1450		-519.06	370.000

FCOST	oldcost	oldflow	oldfc	diagonal	factory	key_id	mth_made
-129174.80	-549.68	535.000	-294078.80	25	1	sales	April
0.00	-460.00	0.000	0.00	25	1	sales	May
-59711.31	-542.83	120.000	-65139.60	25	2	sales	March
-156573.22	-559.19	320.000	-178940.80	25	2	sales	April
-192052.18	-489.06	20.000	-9781.20	25	2	sales	May

Example 4.3. Adding Side Constraints

The manufacturer of Gizmo chips, which are parts needed to make televisions, can supply only 2,600 chips to factory 1 and 3,750 chips to factory 2 in time for production in each of the months March and April. However, Gizmo chips will not be in short supply in May. Three chips are required to make each 19-inch TV while the 25-inch TVs require four chips each. To limit the production of televisions produced at factory 1 in March so that the TVs have the correct number of chips, a side constraint called FACT1 MAR GIZMO is used. The form of this constraint is

```
3 * prod f1 19 mar   +   4 * prod f1 25 mar   <=   2600
```

prod f1 19 mar is the name of the arc directed from the node fact1_1 toward node f1_mar_1 and, in the previous constraint, designates the flow assigned to this arc. The ARCDATA= and CONOUT= data sets have arc names in a variable called _name_.

The other side constraints (shown below) are called FACT2 MAR GIZMO, FACT1 APL GIZMO, and FACT2 APL GIZMO.

```
3 * prod f2 19 mar   +   4 * prod f2 25 mar   <=   3750
3 * prod f1 19 apl   +   4 * prod f1 25 apl   <=   2600
3 * prod f2 19 apl   +   4 * prod f2 25 apl   <=   3750
```

To maintain customer goodwill, the total number of backorders is not to exceed 50 sets. The side constraint TOTAL BACKORDER that models this restriction is

```
back f1 19 apl  +  back f1 25 apl  +
back f2 19 apl  +  back f2 25 apl  +
back f1 19 may  +  back f1 25 may  +
back f2 19 may  +  back f2 25 may   <=   50
```

The sparse CONDATA= data set format is used. All side constraints are of less than or equal type. Because this is the default type value for the DEFCONTYPE= option, type information is not necessary in the following CONDATA=con3. Also, DEFCONTYPE='<=' does not have to be specified in the PROC INTPOINT statement that follows. Notice that the _column_ variable value CHIP/BO LIMIT indicates that an observation of the con3 data set contains rhs information. Therefore, specify RHSOBS='CHIP/BO LIMIT':

```
title2 'Adding Side Constraints';
   data con3;
      input _column_ &$14. _row_ &$15. _coef_ ;
      datalines;
   prod f1 19 mar  FACT1 MAR GIZMO  3
   prod f1 25 mar  FACT1 MAR GIZMO  4
   CHIP/BO LIMIT   FACT1 MAR GIZMO  2600
   prod f2 19 mar  FACT2 MAR GIZMO  3
   prod f2 25 mar  FACT2 MAR GIZMO  4
   CHIP/BO LIMIT   FACT2 MAR GIZMO  3750
   prod f1 19 apl  FACT1 APL GIZMO  3
```

```
prod f1 25 apl  FACT1 APL GIZMO  4
CHIP/BO LIMIT   FACT1 APL GIZMO  2600
prod f2 19 apl  FACT2 APL GIZMO  3
prod f2 25 apl  FACT2 APL GIZMO  4
CHIP/BO LIMIT   FACT2 APL GIZMO  3750
back f1 19 apl  TOTAL BACKORDER  1
back f1 25 apl  TOTAL BACKORDER  1
back f2 19 apl  TOTAL BACKORDER  1
back f2 25 apl  TOTAL BACKORDER  1
back f1 19 may  TOTAL BACKORDER  1
back f1 25 may  TOTAL BACKORDER  1
back f2 19 may  TOTAL BACKORDER  1
back f2 25 may  TOTAL BACKORDER  1
CHIP/BO LIMIT   TOTAL BACKORDER  50
;
```

The four pairs of data sets that follow can be used as ARCDATA= and NODEDATA= data sets in the following PROC INTPOINT run. The set used depends on which cost information the arcs are to have.

```
ARCDATA=arc0    NODEDATA=node0
ARCDATA=arc1    NODEDATA=node0
ARCDATA=arc2    NODEDATA=node0
ARCDATA=arc3    NODEDATA=node0
```

arc0, node0, and arc1 were created in Example 4.1. The first two data sets are the original input data sets.

In the previous example, arc2 was created by modifying arc1 to reflect different arc costs. arc2 and node0 can also be used as the ARCDATA= and NODEDATA= data sets in a PROC INTPOINT run.

If you are going to continue optimization using the changed arc costs, it is probably best to use arc3 and node0 as the ARCDATA= and NODEDATA= data sets.

PROC INTPOINT is used to find the changed cost network solution that obeys the chip limit and backorder side constraints. An explicit ID list has also been specified so that the variables oldcost, oldfc, and oldflow do not appear in the subsequent output data sets:

```
proc intpoint
   bytes=1000000
   printlevel2=2
   nodedata=node0 arcdata=arc3
   condata=con3 sparsecondata rhsobs='CHIP/BO LIMIT'
   conout=arc4;
      id diagonal factory key_id mth_made;
run;
```

```
proc print data=arc4;
   var _tail_ _head_ _cost_ _capac_ _lo_ _flow_ _fcost_;
       /* to get this variable order */
   sum _fcost_;
run;
```

The following messages appear on the SAS log:

```
NOTE: The following variables in ARCDATA do not belong to any
      SAS variable list. These will be ignored.
      _FLOW_
      _FCOST_
      _RCOST_
      _ANUMB_
      _TNUMB_
      _STATUS_
      oldcost
      oldfc
      oldflow
NOTE: Number of nodes= 20 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 4350 , total demand= 4150 .
NOTE: Number of arcs= 64 .
NOTE: Number of <= side constraints= 5 .
NOTE: Number of == side constraints= 0 .
NOTE: Number of >= side constraints= 0 .
NOTE: Number of side constraint coefficients= 16 .
NOTE: The following messages relate to the equivalent
      Linear Program solved by the Interior Point
      algorithm.
NOTE: Number of <= constraints= 5 .
NOTE: Number of == constraints= 21 .
NOTE: Number of >= constraints= 0 .
NOTE: Number of constraint coefficients= 152 .
NOTE: Number of variables= 68 .
NOTE: 5 columns, 0 rows and 5 coefficients were added to
      the problem to handle unrestricted variables,
      variables that are split, and constraint slack or
      surplus variables.
NOTE: There are 82 nonzero elements in A * A transpose.
NOTE: Of the 26 rows and columns, 14 are sparse.
NOTE: There are 80 nonzero superdiagonal elements in the
      sparse rows of the factored A * A transpose. This
      includes fill-in.
NOTE: There are 68 operations of the form
      u[i,j]=u[i,j]-u[q,j]*u[q,i]/u[q,q] to factorize the
      sparse rows of A * A transpose.
Iter Complem_aff Complem-ity  Duality_gap Tot_infeasb  Tot_infeasc Tot_infeasd
   0   -1.000000   174740627     0.834344       52835        40217       46058
   1    48947018    21567675     0.910848 2958.654286  2252.108767 2470.152448
   2     4201633     1377647     0.520587           0 1.331557E-11   46.321197
   3      377792      253074     0.163432           0            0    8.455959
   4      124457       71148     0.051146           0            0    1.425193
   5       46811       28865     0.021063           0            0    0.532792
   6       11689 6377.059370     0.004710           0            0    0.092989
   7 3201.780084 1923.406854     0.001424           0            0    0.019251
   8  468.068267  245.400716     0.000182           0            0    0.002654
```

```
 9   29.762950    5.252240  0.000003889           0            0           0
10    0.005544    0.000290 -4.924982E-9           0            0           0
 NOTE: Primal-Dual Predictor-Corrector Interior point algorithm
       performed 10 iterations.
 NOTE: Objective = -1282708.625.
 NOTE: The data set WORK.ARC4 has 64 observations and 14
       variables.
 NOTE: There were 64 observations read from the data set
       WORK.ARC3.
 NOTE: There were 8 observations read from the data set
       WORK.NODE0.
 NOTE: There were 21 observations read from the data set
       WORK.CON3.
 NOTE: The data set WORK.ARC4 has 64 observations and 14
       variables.
```

Output 4.3.1. CONOUT=ARC4

Adding Side Constraints

Obs	_tail_	_head_	_cost_	_capac_	_lo_	_FLOW_	_FCOST_
1	fact1_1	f1_apr_1	78.60	600	50	533.333	41920.00
2	f1_mar_1	f1_apr_1	15.00	50	0	0.000	0.00
3	f1_may_1	f1_apr_1	33.60	20	0	0.000	0.00
4	f2_apr_1	f1_apr_1	11.00	40	0	0.000	0.00
5	fact1_2	f1_apr_2	174.50	550	50	250.000	43625.00
6	f1_mar_2	f1_apr_2	20.00	40	0	0.000	0.00
7	f1_may_2	f1_apr_2	49.20	15	0	0.000	0.00
8	f2_apr_2	f1_apr_2	21.00	25	0	0.000	0.00
9	fact1_1	f1_mar_1	127.90	500	50	333.333	42633.33
10	f1_apr_1	f1_mar_1	33.60	20	0	20.000	672.00
11	f2_mar_1	f1_mar_1	10.00	40	0	40.000	400.00
12	fact1_2	f1_mar_2	217.90	400	40	400.000	87160.00
13	f1_apr_2	f1_mar_2	38.40	30	0	30.000	1152.00
14	f2_mar_2	f1_mar_2	20.00	25	0	25.000	500.00
15	fact1_1	f1_may_1	90.10	400	50	128.333	11562.83
16	f1_apr_1	f1_may_1	12.00	50	0	0.000	0.00
17	f2_may_1	f1_may_1	13.00	40	0	0.000	0.00
18	fact1_2	f1_may_2	113.30	350	40	350.000	39655.00
19	f1_apr_2	f1_may_2	18.00	40	0	0.000	0.00
20	f2_may_2	f1_may_2	13.00	25	0	0.000	0.00
21	f1_apr_1	f2_apr_1	11.00	99999999	0	13.333	146.67
22	fact2_1	f2_apr_1	62.40	480	35	480.000	29952.00
23	f2_mar_1	f2_apr_1	18.00	30	0	0.000	0.00
24	f2_may_1	f2_apr_1	30.00	15	0	0.000	0.00
25	f1_apr_2	f2_apr_2	23.00	99999999	0	0.000	0.00
26	fact2_2	f2_apr_2	196.70	680	35	577.500	113594.25
27	f2_mar_2	f2_apr_2	28.00	50	0	0.000	0.00
28	f2_may_2	f2_apr_2	64.80	15	0	0.000	0.00
29	f1_mar_1	f2_mar_1	11.00	99999999	0	0.000	0.00
30	fact2_1	f2_mar_1	88.00	450	35	290.000	25520.00
31	f2_apr_1	f2_mar_1	20.40	15	0	0.000	0.00
32	f1_mar_2	f2_mar_2	23.00	99999999	0	0.000	0.00
33	fact2_2	f2_mar_2	182.00	650	35	650.000	118300.00
34	f2_apr_2	f2_mar_2	37.20	15	0	0.000	0.00
35	f1_may_1	f2_may_1	16.00	99999999	0	115.000	1840.00
36	fact2_1	f2_may_1	128.80	250	35	35.000	4508.00
37	f2_apr_1	f2_may_1	20.00	30	0	0.000	0.00
38	f1_may_2	f2_may_2	26.00	99999999	0	350.000	9100.00
39	fact2_2	f2_may_2	181.40	550	35	122.500	22221.50
40	f2_apr_2	f2_may_2	38.00	50	0	0.000	0.00
41	f1_mar_1	shop1_1	-327.65	250	0	143.333	-46963.17
42	f1_apr_1	shop1_1	-300.00	250	0	250.000	-75000.00
43	f1_may_1	shop1_1	-285.00	250	0	13.333	-3800.00
44	f2_mar_1	shop1_1	-297.40	250	0	250.000	-74350.00
45	f2_apr_1	shop1_1	-290.00	250	0	243.333	-70566.67
46	f2_may_1	shop1_1	-292.00	250	0	0.000	0.00
47	f1_mar_2	shop1_2	-559.76	99999999	0	0.000	0.00
48	f1_apr_2	shop1_2	-524.28	99999999	0	0.000	0.00
49	f1_may_2	shop1_2	-475.02	99999999	0	0.000	0.00
50	f2_mar_2	shop1_2	-567.83	500	0	500.000	-283915.00
51	f2_apr_2	shop1_2	-542.19	500	0	400.000	-216876.00
52	f2_may_2	shop1_2	-491.56	500	0	0.000	0.00
53	f1_mar_1	shop2_1	-362.74	250	0	250.000	-90685.00
54	f1_apr_1	shop2_1	-300.00	250	0	250.000	-75000.00
55	f1_may_1	shop2_1	-245.00	250	0	0.000	0.00

Adding Side Constraints

Obs	_tail_	_head_	_cost_	_capac_	_lo_	_FLOW_	_FCOST_
56	f2_mar_1	shop2_1	-272.70	250	0	0.0	0.00
57	f2_apr_1	shop2_1	-312.00	250	0	250.0	-78000.00
58	f2_may_1	shop2_1	-299.00	250	0	150.0	-44850.00
59	f1_mar_2	shop2_2	-623.89	99999999	0	455.0	-283869.95
60	f1_apr_2	shop2_2	-549.68	99999999	0	220.0	-120929.60
61	f1_may_2	shop2_2	-460.00	99999999	0	0.0	0.00
62	f2_mar_2	shop2_2	-542.83	500	0	125.0	-67853.75
63	f2_apr_2	shop2_2	-559.19	500	0	177.5	-99256.23
64	f2_may_2	shop2_2	-519.06	500	0	472.5	-245255.85
							==========
							-1282708.63

Example 4.4. Using Constraints and More Alteration to Arc Data

Suppose the 25-inch screen TVs produced at factory 1 in May can be sold at either shop with an increased profit of 40 dollars each. What is the new optimal solution?

```
title2 'Using Constraints and Altering arc data';
data new_arc4;
   set arc4;
      oldcost=_cost_;
      oldflow=_flow_;
      oldfc=_fcost_;
      if _tail_='f1_may_2'
         & (_head_='shop1_2' | _head_='shop2_2')
         then _cost_=_cost_-40;

proc intpoint
   bytes=1000000
   printlevel2=2
   arcdata=new_arc4 nodedata=node0
   condata=con3 sparsecondata rhsobs='CHIP/BO LIMIT'
   conout=arc5;
run;
proc print data=arc5 (drop = _status_ _rcost_);
   var _tail_ _head_ _cost_ _capac_ _lo_
       _supply_ _demand_ _name_
       _flow_ _fcost_ oldflow oldfc;
          /* to get this variable order */
   sum oldfc _fcost_;
run;
```

The following messages appear on the SAS log:

```
NOTE: Number of nodes= 20 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 4350 , total demand= 4150 .
NOTE: Number of arcs= 64 .
NOTE: Number of <= side constraints= 5 .
NOTE: Number of == side constraints= 0 .
NOTE: Number of >= side constraints= 0 .
NOTE: Number of side constraint coefficients= 16 .
NOTE: The following messages relate to the equivalent
      Linear Program solved by the Interior Point
      algorithm.
NOTE: Number of <= constraints= 5 .
NOTE: Number of == constraints= 21 .
NOTE: Number of >= constraints= 0 .
NOTE: Number of constraint coefficients= 152 .
NOTE: Number of variables= 68 .
NOTE: 5 columns, 0 rows and 5 coefficients were added to
      the problem to handle unrestricted variables,
      variables that are split, and constraint slack or
      surplus variables.
NOTE: There are 82 nonzero elements in A * A transpose.
NOTE: Of the 26 rows and columns, 14 are sparse.
NOTE: There are 80 nonzero superdiagonal elements in the
```

```
       sparse rows of the factored A * A transpose. This
       includes fill-in.
  NOTE: There are 68 operations of the form
       u[i,j]=u[i,j]-u[q,j]*u[q,i]/u[q,q] to factorize the
       sparse rows of A * A transpose.
Iter Complem_aff Complem-ity  Duality_gap Tot_infeasb  Tot_infeasc Tot_infeasd
   0   -1.000000   176072547     0.833846       52835        40217       46430
   1    49323199    21763015     0.910646 2971.887954  2262.182150 2506.228441
   2     4256602     1419262     0.526045           0 6.291145E-11   51.536868
   3      371793      247491     0.159553           0            0    8.939364
   4      119362       60394     0.043112           0            0    1.565229
   5       27925       21537     0.015622           0            0    0.576834
   6       10494 6617.424035     0.004839           0            0    0.124639
   7 3367.128699 1357.529806     0.000996           0            0    0.001406
   8  498.362201  156.077166     0.000115           0            0    0.000163
   9   28.159576    1.000647  0.000000735           0            0           0
  10    0.000549 0.000050153 -8.10256E-11           0            0           0
  NOTE: Primal-Dual Predictor-Corrector Interior point algorithm
       performed 10 iterations.
  NOTE: Objective = -1295661.8.
  NOTE: The data set WORK.ARC5 has 64 observations and 17
       variables.
  NOTE: There were 64 observations read from the data set
       WORK.NEW_ARC4.
  NOTE: There were 8 observations read from the data set
       WORK.NODE0.
  NOTE: There were 21 observations read from the data set
       WORK.CON3.
  NOTE: The data set WORK.ARC5 has 64 observations and 17
       variables.
```

Output 4.4.1. CONOUT=ARC5

tail	_head_	_cost_	_capac_	_lo_	_SUPPLY_	_DEMAND_
fact1_1	f1_apr_1	78.6	600	50	1000	.
f1_mar_1	f1_apr_1	15.0	50	0	.	.
f1_may_1	f1_apr_1	33.6	20	0	.	.
f2_apr_1	f1_apr_1	11.0	40	0	.	.
fact1_2	f1_apr_2	174.5	550	50	1000	.
f1_mar_2	f1_apr_2	20.0	40	0	.	.
f1_may_2	f1_apr_2	49.2	15	0	.	.
f2_apr_2	f1_apr_2	21.0	25	0	.	.
fact1_1	f1_mar_1	127.9	500	50	1000	.
f1_apr_1	f1_mar_1	33.6	20	0	.	.
f2_mar_1	f1_mar_1	10.0	40	0	.	.
fact1_2	f1_mar_2	217.9	400	40	1000	.
f1_apr_2	f1_mar_2	38.4	30	0	.	.
f2_mar_2	f1_mar_2	20.0	25	0	.	.
fact1_1	f1_may_1	90.1	400	50	1000	.
f1_apr_1	f1_may_1	12.0	50	0	.	.
f2_may_1	f1_may_1	13.0	40	0	.	.
fact1_2	f1_may_2	113.3	350	40	1000	.
f1_apr_2	f1_may_2	18.0	40	0	.	.
f2_may_2	f1_may_2	13.0	25	0	.	.
f1_apr_1	f2_apr_1	11.0	99999999	0	.	.
fact2_1	f2_apr_1	62.4	480	35	850	.
f2_mar_1	f2_apr_1	18.0	30	0	.	.
f2_may_1	f2_apr_1	30.0	15	0	.	.
f1_apr_2	f2_apr_2	23.0	99999999	0	.	.
fact2_2	f2_apr_2	196.7	680	35	1500	.
f2_mar_2	f2_apr_2	28.0	50	0	.	.

name	_FLOW_	_FCOST_	oldflow	oldfc
prod f1 19 apl	533.333	41920.00	533.333	41920.00
	0.000	0.00	0.000	0.00
back f1 19 may	0.000	0.00	0.000	0.00
	0.000	0.00	0.000	0.00
prod f1 25 apl	250.000	43625.00	250.000	43625.00
	0.000	0.00	0.000	0.00
back f1 25 may	0.000	0.00	0.000	0.00
	0.000	0.00	0.000	0.00
prod f1 19 mar	333.333	42633.33	333.333	42633.33
back f1 19 apl	20.000	672.00	20.000	672.00
	40.000	400.00	40.000	400.00
prod f1 25 mar	400.000	87160.00	400.000	87160.00
back f1 25 apl	30.000	1152.00	30.000	1152.00
	25.000	500.00	25.000	500.00
	128.333	11562.83	128.333	11562.83
	0.000	0.00	0.000	0.00
	0.000	0.00	0.000	0.00
	350.000	39655.00	350.000	39655.00
	0.000	0.00	0.000	0.00
	0.000	0.00	0.000	0.00
	13.333	146.67	13.333	146.67
prod f2 19 apl	480.000	29952.00	480.000	29952.00
	0.000	0.00	0.000	0.00
back f2 19 may	0.000	0.00	0.000	0.00
	0.000	0.00	0.000	0.00
prod f2 25 apl	550.000	108185.00	577.500	113594.25
	0.000	0.00	0.000	0.00

tail	_head_	_cost_	_capac_	_lo_	_SUPPLY_	_DEMAND_
f2_may_2	f2_apr_2	64.8	15	0	.	.
f1_mar_1	f2_mar_1	11.0	99999999	0	.	.
fact2_1	f2_mar_1	88.0	450	35	850	.
f2_apr_1	f2_mar_1	20.4	15	0	.	.
f1_mar_2	f2_mar_2	23.0	99999999	0	.	.
fact2_2	f2_mar_2	182.0	650	35	1500	.
f2_apr_2	f2_mar_2	37.2	15	0	.	.

name	_FLOW_	_FCOST_	oldflow	oldfc
back f2 25 may	0.000	0.00	0.000	0.00
	0.000	0.00	0.000	0.00
prod f2 19 mar	290.000	25520.00	290.000	25520.00
back f2 19 apl	0.000	0.00	0.000	0.00
	0.000	0.00	0.000	0.00
prod f2 25 mar	650.000	118300.00	650.000	118300.00
back f2 25 apl	0.000	0.00	0.000	0.00

tail	_head_	_cost_	_capac_	_lo_	_SUPPLY_	_DEMAND_
f1_may_1	f2_may_1	16.00	99999999	0	.	.
fact2_1	f2_may_1	128.80	250	35	850	.
f2_apr_1	f2_may_1	20.00	30	0	.	.
f1_may_2	f2_may_2	26.00	99999999	0	.	.
fact2_2	f2_may_2	181.40	550	35	1500	.
f2_apr_2	f2_may_2	38.00	50	0	.	.
f1_mar_1	shop1_1	-327.65	250	0	.	900
f1_apr_1	shop1_1	-300.00	250	0	.	900
f1_may_1	shop1_1	-285.00	250	0	.	900
f2_mar_1	shop1_1	-297.40	250	0	.	900
f2_apr_1	shop1_1	-290.00	250	0	.	900
f2_may_1	shop1_1	-292.00	250	0	.	900
f1_mar_2	shop1_2	-559.76	99999999	0	.	900
f1_apr_2	shop1_2	-524.28	99999999	0	.	900
f1_may_2	shop1_2	-515.02	99999999	0	.	900
f2_mar_2	shop1_2	-567.83	500	0	.	900
f2_apr_2	shop1_2	-542.19	500	0	.	900
f2_may_2	shop1_2	-491.56	500	0	.	900
f1_mar_1	shop2_1	-362.74	250	0	.	900
f1_apr_1	shop2_1	-300.00	250	0	.	900
f1_may_1	shop2_1	-245.00	250	0	.	900
f2_mar_1	shop2_1	-272.70	250	0	.	900
f2_apr_1	shop2_1	-312.00	250	0	.	900
f2_may_1	shop2_1	-299.00	250	0	.	900
f1_mar_2	shop2_2	-623.89	99999999	0	.	1450
f1_apr_2	shop2_2	-549.68	99999999	0	.	1450
f1_may_2	shop2_2	-500.00	99999999	0	.	1450

name	_FLOW_	_FCOST_	oldflow	oldfc
	115.000	1840.00	115.000	1840.00
	35.000	4508.00	35.000	4508.00
	0.000	0.00	0.000	0.00
	0.000	0.00	350.000	9100.00
	150.000	27210.00	122.500	22221.50
	0.000	0.00	0.000	0.00
	143.333	-46963.17	143.333	-46963.17
	250.000	-75000.00	250.000	-75000.00
	13.333	-3800.00	13.333	-3800.00
	250.000	-74350.00	250.000	-74350.00
	243.333	-70566.67	243.333	-70566.67
	0.000	0.00	0.000	0.00
	0.000	0.00	0.000	0.00
	0.000	0.00	0.000	0.00
	350.000	-180257.00	0.000	0.00
	500.000	-283915.00	500.000	-283915.00
	50.000	-27109.50	400.000	-216876.00
	0.000	0.00	0.000	0.00
	250.000	-90685.00	250.000	-90685.00
	250.000	-75000.00	250.000	-75000.00
	0.000	0.00	0.000	0.00
	0.000	0.00	0.000	0.00
	250.000	-78000.00	250.000	-78000.00
	150.000	-44850.00	150.000	-44850.00
	455.000	-283869.95	455.000	-283869.95
	220.000	-120929.60	220.000	-120929.60
	0.000	0.00	0.000	0.00

tail	_head_	_cost_	_capac_	_lo_	_SUPPLY_	_DEMAND_
f2_mar_2	shop2_2	-542.83	500	0	.	1450
f2_apr_2	shop2_2	-559.19	500	0	.	1450
f2_may_2	shop2_2	-519.06	500	0	.	1450

name	_FLOW_	_FCOST_	oldflow	oldfc
	125.000	-67853.75	125.000	-67853.75
	500.000	-279595.00	177.500	-99256.23
	150.000	-77859.00	472.500	-245255.85

Example 4.5. Nonarc Variables in the Side Constraints

You can verify that the FACT2 MAR GIZMO constraint has a left-hand-side activity of 3,470, which is not equal to the _RHS_ of this constraint. Not all of the 3,750 chips that can be supplied to factory 2 for March production are used. It is suggested that all the possible chips be obtained in March and those not used be saved for April production. Because chips must be kept in an air-controlled environment, it costs one dollar to store each chip purchased in March until April. The maximum number of chips that can be stored in this environment at each factory is 150. In addition, a search of the parts inventory at factory 1 turned up 15 chips available for their March production.

Nonarc variables are used in the side constraints that handle the limitations of supply of Gizmo chips. A nonarc variable called f1 unused mar has as a value the number of chips that are not used at factory 1 in March. Another nonarc variable, f2 unused mar, has as a value the number of chips that are not used at factory 2 in March. f1 chips from mar has as a value the number of chips left over from March used for production at factory 1 in April. Similarly, f2 chips from mar has as a value the number of chips left over from March used for April production at factory 2 in April. The last two nonarc variables have objective function coefficients of 1 and upper bounds of 150. The Gizmo side constraints are

```
3*prod f1 19 mar + 4*prod f1 25 mar + f1 unused chips = 2615
3*prod f2 19 apl + 4*prod f2 25 apl + f2 unused chips = 3750
3*prod f1 19 apl + 4*prod f1 25 apl - f1 chips from mar = 2600
3*prod f2 19 apl + 4*prod f2 25 apl - f2 chips from mar = 3750
f1 unused chips + f2 unused chips -
f1 chips from mar - f2 chips from mar >= 0
```

The last side constraint states that the number of chips not used in March is not less than the number of chips left over from March and used in April. Here, this constraint is called CHIP LEFTOVER.

The following SAS code creates a new data set containing constraint data. It seems that most of the constraints are now equalities, so you specify DEFCONTYPE=EQ in the PROC INTPOINT statements from now on and provide constraint type data for constraints that are not "equal to" type, using the default TYPEOBS value _TYPE_ as the _COLUMN_ variable value to indicate observations that contain constraint type data. Also, from now on, the default RHSOBS value is used:

```
title2 'Nonarc Variables in the Side Constraints';
   data con6;
      input _column_ &$17. _row_ &$15. _coef_ ;
      datalines;
   prod f1 19 mar     FACT1 MAR GIZMO  3
   prod f1 25 mar     FACT1 MAR GIZMO  4
   f1 unused chips    FACT1 MAR GIZMO  1
   _RHS_              FACT1 MAR GIZMO  2615
   prod f2 19 mar     FACT2 MAR GIZMO  3
   prod f2 25 mar     FACT2 MAR GIZMO  4
   f2 unused chips    FACT2 MAR GIZMO  1
   _RHS_              FACT2 MAR GIZMO  3750
   prod f1 19 apl     FACT1 APL GIZMO  3
   prod f1 25 apl     FACT1 APL GIZMO  4
   f1 chips from mar  FACT1 APL GIZMO  -1
   _RHS_              FACT1 APL GIZMO  2600
   prod f2 19 apl     FACT2 APL GIZMO  3
   prod f2 25 apl     FACT2 APL GIZMO  4
   f2 chips from mar  FACT2 APL GIZMO  -1
   _RHS_              FACT2 APL GIZMO  3750
   f1 unused chips    CHIP LEFTOVER    1
   f2 unused chips    CHIP LEFTOVER    1
   f1 chips from mar  CHIP LEFTOVER    -1
   f2 chips from mar  CHIP LEFTOVER    -1
   _TYPE_             CHIP LEFTOVER    1
   back f1 19 apl     TOTAL BACKORDER  1
   back f1 25 apl     TOTAL BACKORDER  1
   back f2 19 apl     TOTAL BACKORDER  1
   back f2 25 apl     TOTAL BACKORDER  1
   back f1 19 may     TOTAL BACKORDER  1
   back f1 25 may     TOTAL BACKORDER  1
   back f2 19 may     TOTAL BACKORDER  1
   back f2 25 may     TOTAL BACKORDER  1
   _TYPE_             TOTAL BACKORDER  -1
   _RHS_              TOTAL BACKORDER  50
   ;
```

The nonarc variables f1 chips from mar and f2 chips from mar have objective function coefficients of 1 and upper bounds of 150. There are various ways in which this information can be furnished to PROC INTPOINT. If there were a TYPE list variable in the CONDATA= data set, observations could be in the form:

```
   _COLUMN_           _TYPE_  _ROW_ _COEF_
   f1 chips from mar  objfn     .       1
   f1 chips from mar  upperbd   .     150
   f2 chips from mar  objfn     .       1
   f2 chips from mar  upperbd   .     150
```

It is desirable to assign ID list variable values to all the nonarc variables:

```
data arc6;
   set arc5;
   drop oldcost oldfc oldflow _flow_ _fcost_ _status_ _rcost_;
data arc6_b;
   input _name_ &$17. _cost_ _capac_ factory key_id $ ;
   datalines;
f1 unused chips    .   . 1 chips
f2 unused chips    .   . 2 chips
f1 chips from mar  1 150 1 chips
f2 chips from mar  1 150 2 chips
;
proc append force
   base=arc6 data=arc6_b;
proc intpoint
   bytes=1000000
   printlevel2=2
   nodedata=node0 arcdata=arc6
   condata=con6 defcontype=eq sparsecondata
   conout=arc7;
run;
```

The following messages appear on the SAS log:

```
NOTE: Number of nodes= 20 .
NOTE: Number of supply nodes= 4 .
NOTE: Number of demand nodes= 4 .
NOTE: Total supply= 4350 , total demand= 4150 .
NOTE: Number of arcs= 64 .
NOTE: Number of nonarc variables= 4 .
NOTE: Number of <= side constraints= 1 .
NOTE: Number of == side constraints= 4 .
NOTE: Number of >= side constraints= 1 .
NOTE: Number of side constraint coefficients= 24 .
NOTE: The following messages relate to the equivalent
      Linear Program solved by the Interior Point
      algorithm.
NOTE: Number of <= constraints= 1 .
NOTE: Number of == constraints= 25 .
NOTE: Number of >= constraints= 1 .
NOTE: Number of constraint coefficients= 160 .
NOTE: Number of variables= 72 .
NOTE: 2 columns, 0 rows and 2 coefficients were added to
      the problem to handle unrestricted variables,
      variables that are split, and constraint slack or
      surplus variables.
NOTE: There are 86 nonzero elements in A * A transpose.
NOTE: Of the 27 rows and columns, 16 are sparse.
NOTE: There are 102 nonzero superdiagonal elements in the
      sparse rows of the factored A * A transpose. This
      includes fill-in.
NOTE: There are 160 operations of the form
      u[i,j]=u[i,j]-u[q,j]*u[q,i]/u[q,q] to factorize the
      sparse rows of A * A transpose.
Iter Complem_aff Complem-ity  Duality_gap Tot_infeasb  Tot_infeasc Tot_infeasd
   0   -1.000000   180211988     0.837584       55030        38331       44219
```

```
 1    54358286    27998179     0.909216 4949.974964  3447.853782 6457.752523
 2     9372834     2493303     0.657738           0 8.594938E-10  196.155157
 3      360076      311567     0.192309           0            0   24.426434
 4      133734       91250     0.064112           0            0    5.827344
 5       66155       36320     0.026230           0            0           0
 6       18053 8903.419999     0.006517           0            0           0
 7 3897.577387 1910.615008     0.001402           0            0           0
 8  847.313077  362.303033     0.000266           0            0           0
 9  146.082127   45.122412  0.000033136           0            0           0
10    6.217057    0.574649  0.000000422           0            0           0
11    0.002810 0.000029061 2.152507E-11           0            0           0
NOTE: Primal-Dual Predictor-Corrector Interior point algorithm
      performed 11 iterations.
NOTE: Objective = -1295542.742.
NOTE: The data set WORK.ARC7 has 68 observations and 14
      variables.
NOTE: There were 68 observations read from the data set
      WORK.ARC6.
NOTE: There were 8 observations read from the data set
      WORK.NODE0.
NOTE: There were 31 observations read from the data set
      WORK.CON6.
NOTE: The data set WORK.ARC7 has 68 observations and 14
      variables.
```

The optimal solution data set, CONOUT=ARC7 in Output 4.5.1 follow:

```
proc print data=arc7;
   var _tail_ _head_ _name_ _cost_ _capac_ _lo_
       _flow_ _fcost_;
   sum _fcost_;
run;
```

Output 4.5.1. CONOUT=ARC7

Nonarc Variables in the Side Constraints

Obs	_tail_	_head_	_name_	_cost_	_capac_	_lo_	_FLOW_	_FCOST_
1	fact1_1	f1_apr_1	prod f1 19 apl	78.60	600	50	540.000	42444.00
2	f1_mar_1	f1_apr_1		15.00	50	0	0.000	0.00
3	f1_may_1	f1_apr_1	back f1 19 may	33.60	20	0	0.000	0.00
4	f2_apr_1	f1_apr_1		11.00	40	0	0.000	0.00
5	fact1_2	f1_apr_2	prod f1 25 apl	174.50	550	50	250.000	43625.00
6	f1_mar_2	f1_apr_2		20.00	40	0	0.000	0.00
7	f1_may_2	f1_apr_2	back f1 25 may	49.20	15	0	0.000	0.00
8	f2_apr_2	f1_apr_2		21.00	25	0	25.000	525.00
9	fact1_1	f1_mar_1	prod f1 19 mar	127.90	500	50	338.333	43272.83
10	f1_apr_1	f1_mar_1	back f1 19 apl	33.60	20	0	20.000	672.00
11	f2_mar_1	f1_mar_1		10.00	40	0	40.000	400.00
12	fact1_2	f1_mar_2	prod f1 25 mar	217.90	400	40	400.000	87160.00
13	f1_apr_2	f1_mar_2	back f1 25 apl	38.40	30	0	30.000	1152.00
14	f2_mar_2	f1_mar_2		20.00	25	0	25.000	500.00
15	fact1_1	f1_may_1		90.10	400	50	116.667	10511.67
16	f1_apr_1	f1_may_1		12.00	50	0	0.000	0.00
17	f2_may_1	f1_may_1		13.00	40	0	0.000	0.00
18	fact1_2	f1_may_2		113.30	350	40	350.000	39655.00
19	f1_apr_2	f1_may_2		18.00	40	0	0.000	0.00
20	f2_may_2	f1_may_2		13.00	25	0	0.000	0.00
21	f1_apr_1	f2_apr_1		11.00	99999999	0	20.000	220.00
22	fact2_1	f2_apr_1	prod f2 19 apl	62.40	480	35	480.000	29952.00
23	f2_mar_1	f2_apr_1		18.00	30	0	0.000	0.00
24	f2_may_1	f2_apr_1	back f2 19 may	30.00	15	0	0.000	0.00
25	f1_apr_2	f2_apr_2		23.00	99999999	0	0.000	0.00
26	fact2_2	f2_apr_2	prod f2 25 apl	196.70	680	35	577.500	113594.25
27	f2_mar_2	f2_apr_2		28.00	50	0	0.000	0.00
28	f2_may_2	f2_apr_2	back f2 25 may	64.80	15	0	0.000	0.00
29	f1_mar_1	f2_mar_1		11.00	99999999	0	0.000	0.00
30	fact2_1	f2_mar_1	prod f2 19 mar	88.00	450	35	290.000	25520.00
31	f2_apr_1	f2_mar_1	back f2 19 apl	20.40	15	0	0.000	0.00
32	f1_mar_2	f2_mar_2		23.00	99999999	0	0.000	0.00
33	fact2_2	f2_mar_2	prod f2 25 mar	182.00	650	35	650.000	118300.00
34	f2_apr_2	f2_mar_2	back f2 25 apl	37.20	15	0	0.000	0.00
35	f1_may_1	f2_may_1		16.00	99999999	0	115.000	1840.00
36	fact2_1	f2_may_1		128.80	250	35	35.000	4508.00
37	f2_apr_1	f2_may_1		20.00	30	0	0.000	0.00
38	f1_may_2	f2_may_2		26.00	99999999	0	0.000	0.00
39	fact2_2	f2_may_2		181.40	550	35	122.500	22221.50
40	f2_apr_2	f2_may_2		38.00	50	0	0.000	0.00
41	f1_mar_1	shop1_1		-327.65	250	0	148.333	-48601.42
42	f1_apr_1	shop1_1		-300.00	250	0	250.000	-75000.00
43	f1_may_1	shop1_1		-285.00	250	0	1.667	-475.00
44	f2_mar_1	shop1_1		-297.40	250	0	250.000	-74350.00
45	f2_apr_1	shop1_1		-290.00	250	0	250.000	-72500.00
46	f2_may_1	shop1_1		-292.00	250	0	0.000	0.00
47	f1_mar_2	shop1_2		-559.76	99999999	0	0.000	0.00
48	f1_apr_2	shop1_2		-524.28	99999999	0	0.000	0.00
49	f1_may_2	shop1_2		-515.02	99999999	0	347.500	-178969.45
50	f2_mar_2	shop1_2		-567.83	500	0	500.000	-283915.00
51	f2_apr_2	shop1_2		-542.19	500	0	52.500	-28464.98
52	f2_may_2	shop1_2		-491.56	500	0	0.000	0.00
53	f1_mar_1	shop2_1		-362.74	250	0	250.000	-90685.00
54	f1_apr_1	shop2_1		-300.00	250	0	250.000	-75000.00
55	f1_may_1	shop2_1		-245.00	250	0	0.000	0.00

Nonarc Variables in the Side Constraints

Obs	_tail_	_head_	_name_	_cost_	_capac_	_lo_	_FLOW_	_FCOST_
56	f2_mar_1	shop2_1		-272.70	250	0	0.0	0.00
57	f2_apr_1	shop2_1		-312.00	250	0	250.0	-78000.00
58	f2_may_1	shop2_1		-299.00	250	0	150.0	-44850.00
59	f1_mar_2	shop2_2		-623.89	99999999	0	455.0	-283869.95
60	f1_apr_2	shop2_2		-549.68	99999999	0	245.0	-134671.60
61	f1_may_2	shop2_2		-500.00	99999999	0	2.5	-1250.00
62	f2_mar_2	shop2_2		-542.83	500	0	125.0	-67853.75
63	f2_apr_2	shop2_2		-559.19	500	0	500.0	-279595.00
64	f2_may_2	shop2_2		-519.06	500	0	122.5	-63584.85
65			f1 chips from mar	1.00	150	0	20.0	20.00
66			f1 unused chips	0.00	99999999	0	0.0	0.00
67			f2 chips from mar	1.00	150	0	0.0	0.00
68			f2 unused chips	0.00	99999999	0	280.0	0.00
								-1295542.74

The optimal value of the nonarc variable f2 unused chips is 280. This means that although there are 3,750 chips that can be used at factory 2 in March, only 3,470 are used. As the optimal value of f1 unused chips is zero, all chips available for production in March at factory 1 are used. The nonarc variable f2 chips from mar also has zero optimal value. This means that the April production at factory 2 does not need any chips that could have been held in inventory since March. However, the nonarc variable f1 chips from mar has value of 20. Thus, 3,490 chips should be ordered for factory 2 in March. Twenty of these chips should be held in inventory until April, then sent to factory 1.

Example 4.6. Solving an LP Problem with Data in MPS Format

In this example, PROC INTPOINT is ultimately used to solve an LP. But prior to that, there is SAS code that is used to read a MPS format file and initialize an input SAS data set. MPS was an optimization package developed for IBM computers many years ago and the format by which data had to be supplied to that system became the industry standard for other optimization software packages, including those developed recently. The MPS format is described in Murtagh (1981). If you have an LP with has data in MPS format in a file /your-directorys/your-filename.dat, then the following SAS code should be run:

```
filename w '/your-directorys/your-filename.dat';
data raw;
  infile w lrecl=80 pad;
  input field1 $ 2-3 field2 $ 5-12 field3 $ 15-22
        field4  25-36 field5 $ 40-47 field6 50-61;
run;
%sasmpsxs;
data lp;
  set;
  if _type_="FREE" then _type_="MIN";
  if lag(_type_)="*HS" then _type_="RHS";
run;
proc sort data=lp;by _col_;run;

proc intpoint
   arcdata=lp
   condata=lp sparsecondata rhsobs=rhs grouped=condata
   conout=solutn  /* SAS data set for the optimal solution */
   bytes=20000000
   nnas=1700 ncoefs=4000 ncons=700
   printlevel2=2 memrep;
run;

proc lp
   data=lp sparsedata
   endpause time=3600 maxit1=100000 maxit2=100000;
run;
show status;
quit;
```

You will have to specify the appropriate path and file name in which your MPS format data resides.

%sasmpsxs; is a SAS macro provided within SAS/OR software. The MPS format resembles the sparse format of the CONDATA= data set for PROC INTPOINT. The SAS macro %sasmpsxs; examines the MPS data and transfers it into a SAS data set while automatically taking into account how the MPS format differs slightly from PROC INTPOINT's sparse format.

The parameters NNAS==1700, NCOEFS=4000, and NCONS=700 indicate the approximate (over-estimated) number of variables, coefficients and constraints this model has. You must change these to your problems dimensions. Knowing these, PROC INTPOINT is able to utilize memory better and read the data faster. These parameters are optional.

The PROC SORT preceding PROC INTPOINT is not necessary, but sorting the SAS data set can speed up PROC INTPOINT when it reads the data. After the sort, data for each column is grouped together. GROUPED=condata can be specified.

For small problems, presorting and specifying those additional options is not going to greatly influence PROC INTPOINT's run time. However, when problems are large, presorting and specifying those additional options can be very worthwhile.

If you generate the model yourself, you will be familiar enough with it to know what to specify for the RHSOBS= parameter. If the value of the SAS variable in the COLUMN list is equal to the character string specified as the RHSOBS= option, the data in that observation is interpreted as right-hand-side data as opposed to coefficient data. If you do not know what to specify for the RHSOBS=rhs-charstr, you should first run PROC LP and optionally set maxit1=1 and maxit2=1. PROC LP will output a Problem Summary that includes the line

```
Rhs Variable    rhs-charstr.
```

BYTES=20000000 is the size of working memory PROC INTPOINT is allowed.

The options PRINTLEVEL2=2 and MEMREP indicate that you want to see an iteration log and messages about memory usage. Specifying these options are optional.

References

Lustig, I.J., Marsten, R.E., and Shanno, D.F. (1992), "On Implementing Mehrotra's Predictor-Corrector Interior-Point Method for Linear Programming," *SIAM Journal of Optimization* 2 (3), 435-449

Murtagh, B.A. (1981), "Advanced Linear Programming," McGraw-Hill International Book Company.

Reid, J.K. (1975), "A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases," *Harwell Report CSS 20*, Atomic Energy Research Establishment, Didcot, Oxfordshire, England.

Roos, C., Terlaky, T., and Vial, J.-Ph. (1997), "Theory and Algorithms for Linear Optimization," Chichester, England: John Wiley & Sons.

Wright, S.J. (1996), "Primal-Dual Interior Point Algorithms," Philadelphia: SIAM.

Ye, Y. (1996), "Interior Point Algorithms: Theory and Analysis," Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons.

Chapter 5
The LP Procedure

Chapter Table of Contents

Chapter 5
The LP Procedure

Overview

A new option has been added to the LP procedure that enables you to choose a generally more efficient LU decomposition algorithm. While the current default algorithm for PROC LP, Reid's algorithm, is very reliable, you can now choose a relatively new and very efficient algorithm by Gilbert and Peierls. It is several times faster than Reid's algorithm on many large linear or integer programming problems.

Syntax

PROC LP Statement

The following new option is available in the LP procedure statement:

LU=*i*

specifies an LU decomposition algorithm.

LU=1	selects the LU decomposition algorithm by Reid, 1976
LU=2	selects the LU decomposition algorithm by Gilbert and Peierls, 1988

The default value is 1.

Chapter 6
The PM Procedure

Chapter Table of Contents

Chapter 6
The PM Procedure

Overview

MP2KTOPM is a new SAS macro that converts Microsoft® Project 2000 data saved in MDB format to a form that is readable by the PM procedure. The converted data can also be read by the CPM procedure.

The new options added to the CPM procedure are also available with PROC PM.

Details

Microsoft Project Data Conversion

There are two SAS macros, MDBTOPM and MP2KTOPM, that are available for converting Microsoft® Project data to a form that is readable by the PM procedure. MDBTOPM converts Microsoft Project 98 data and MP2KTOPM converts Microsoft Project 2000 data. In the event of a succesful conversion, each of these macros proceeds to invoke an instance of the PM procedure using values for the relevant options that were determined during the course of the data conversion. Execution of these macros requires SAS/ACCESS software.

MDBTOPM is a SAS macro that converts Microsoft Project 98 data saved in MDB format to a form that is readable by the PM procedure. The MDBTOPM macro has two arguments: one that specifies the location of the .MDB file and another that specifies the location of the directory for storing the SAS data sets.

MP2KTOPM is a SAS macro that converts Microsoft Project 2000 data saved in MDB format to a form that is readable by the PM procedure. The MP2KTOPM macro has three arguments: one that specifies the location of the .MDB file, one that specifies the location of the directory for storing the SAS data sets, and an optional numeric third argument that controls the mode of the PM procedure invocation. A nonzero value invokes the PM procedure in the default interactive mode and a value of zero invokes the PM procedure in noninteractive mode.

The following SAS data sets are generated by the conversion macros:

- Activity data set
- Calendar data set
- Holiday data set
- Workday data set
- Resource data set

- Preferences data set

The MDBTOPM and MP2KTOPM macros convert the hierarchical relationships, precedence relationships, time constraints, resource availabilities, resource requirements, project calendars, resource calendars, holiday information, workshift information, actual start and finish times, and baseline start and finish times. In addition, the notes and cost fields are also extracted and stored in the Activity data set.

Note: In order to retain the values of the options used to invoke the PM procedure, you will need to save the preferences by choosing Preferences->Save from the "Project" pull-down menu.

Subject Index

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Syntax Index

U

V

W

Z

Your Turn

If you have comments or suggestions about *SAS/OR® Software: Changes and Enhancements, Release 8.2,* please send them to us on a photocopy of this page or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Institute Inc.
SAS Publishing
SAS Campus Drive
Cary, NC 27513
E-mail: yourturn@sas.com

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
E-mail: suggest@sas.com

*Welcome * Bienvenue * Willkommen * Yohkoso * Bienvenido*

SAS® Institute Publishing Is Easy to Reach

Visit our Web page located at www.sas.com/pubs

You will find product and service details, including

- **sample chapters**
- **tables of contents**
- **author biographies**
- **book reviews**

Learn about

- **regional user-group conferences**
- **trade-show sites and dates**
- **authoring opportunities**
- **custom textbooks**

Explore all the services that SAS Institute Publishing has to offer!

Your Listserv Subscription Automatically Brings the News to You

Do you want to be among the first to learn about the latest books and services available from SAS Institute Publishing? Subscribe to our listserv **newdocnews-l** and, once each month, you will automatically receive a description of the newest books and which environments or operating systems and SAS release(s) that each book addresses.

To subscribe,

1. Send an e-mail message to **listserv@vm.sas.com**.
2. Leave the "Subject" line blank.
3. Use the following text for your message:

   subscribe NEWDOCNEWS-L *your-first-name your-last-name*

   For example: subscribe NEWDOCNEWS-L John Doe

Create Customized Textbooks Quickly, Easily, and Affordably

SelecText® offers instructors at U.S. colleges and universities a way to create custom textbooks for courses that teach students how to use SAS software.

For more information, see our Web page at **www.sas.com/selectext**, or contact our SelecText coordinators by sending e-mail to **selectext@sas.com**.

You're Invited to Publish with SAS Institute's User Publishing Program

If you enjoy writing about SAS software and how to use it, the User Publishing Program at SAS Institute offers a variety of publishing options. We are actively recruiting authors to publish books, articles, and sample code. Do you find the idea of writing a book or an article by yourself a little intimidating? Consider writing with a co-author. Keep in mind that you will receive complete editorial and publishing support, access to our users, technical advice and assistance, and competitive royalties. Please contact us for an author packet. E-mail us at **sasbbu@sas.com** or call 919-677-8000, then press 1-6479. See the SAS Institute Publishing Web page at **www.sas.com/pubs** for complete information.

See *Observations*®, Our Online Technical Journal

Feature articles from *Observations*®: *The Technical Journal for SAS® Software Users* are now available online at **www.sas.com/obs**. Take a look at what your fellow SAS software users and SAS Institute experts have to tell you. You may decide that you, too, have information to share. If you are interested in writing for *Observations*, send e-mail to **sasbbu@sas.com** or call 919-677-8000, then press 1-6479.

Book Discount Offered at SAS Public Training Courses!

When you attend one of our SAS Public Training Courses at any of our regional Training Centers in the U.S., you will receive a 15% discount on book orders that you place during the course. Take advantage of this offer at the next course you attend!

SAS Institute
SAS Campus Drive
Cary, NC 27513-2414
Fax 919-677-4444

E-mail: sasbook@sas.com
Web page: www.sas.com/pubs
To order books, call Fulfillment Services at 800-727-3228*
For other SAS Institute business, call 919-677-8000*

*** Note:** Customers outside the U.S. should contact their local SAS office.