SAS/OR® 13.2 User's Guide
Mathematical Programming

Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

§sas

THE POWER TO KNOW®

Chapter 15
The Decomposition Algorithm

Contents

Overview: Decomposition Algorithm

The SAS/OR decomposition algorithm (DECOMP) provides an alternative method of solving linear programs (LPs) and mixed integer linear programs (MILPs) by exploiting the ability to efficiently solve a relaxation of the original problem. The algorithm is available as an option in the OPTMODEL, OPTLP, and OPTMILP procedures and is based on the methodology described in Galati (2009).

A standard linear or mixed integer linear program has the formulation

$$
\begin{array}{lllll}
\text{minimize} & \mathbf{c}^\top \mathbf{x} & + & \mathbf{f}^\top \mathbf{y} \\
\text{subject to} & \mathbf{Dx} & + & \mathbf{By} & \{\geq, =, \leq\} \quad \mathbf{d} \quad \text{(master)} \\
& \mathbf{Ax} & & & \{\geq, =, \leq\} \quad \mathbf{b} \quad \text{(subproblem)} \\
& \underline{x}_i \leq x_i \leq \overline{x}_i, & x_i \in \mathbb{Z} \quad i \in \mathcal{S}_x \\
& \underline{y}_i \leq y_i \leq \overline{y}_i, & y_i \in \mathbb{Z} \quad i \in \mathcal{S}_y
\end{array}
$$

where

$\mathbf{x} \in \mathbb{R}^n$	is the vector of structural variables
$\mathbf{y} \in \mathbb{R}^s$	is the vector of master-only structural variables
$\mathbf{c} \in \mathbb{R}^n$	is the vector of objective function coefficients that are associated with variables $\mathbf{x}$
$\mathbf{f} \in \mathbb{R}^s$	is the vector of objective function coefficients that are associated with variables $\mathbf{y}$
$\mathbf{D} \in \mathbb{R}^{t \times n}$	is the matrix of master constraint coefficients that are associated with variables $\mathbf{x}$
$\mathbf{B} \in \mathbb{R}^{t \times s}$	is the matrix of master constraint coefficients that are associated with variables $\mathbf{y}$
$\mathbf{A} \in \mathbb{R}^{m \times n}$	is the matrix of subproblem constraint coefficients
$\mathbf{d} \in \mathbb{R}^t$	is the vector of master constraints' right-hand sides
$\mathbf{b} \in \mathbb{R}^m$	is the vector of subproblem constraints' right-hand sides
$\underline{\mathbf{x}} \in \mathbb{R}^n$	is the vector of lower bounds on variables $\mathbf{x}$
$\overline{\mathbf{x}} \in \mathbb{R}^n$	is the vector of upper bounds on variables $\mathbf{x}$
$\underline{\mathbf{y}} \in \mathbb{R}^s$	is the vector of lower bounds on variables $\mathbf{y}$
$\overline{\mathbf{y}} \in \mathbb{R}^s$	is the vector of upper bounds on variables $\mathbf{y}$
$\mathcal{S}_x$	is a subset of the set $\{1, \ldots, n\}$ of indices on variables $\mathbf{x}$
$\mathcal{S}_y$	is a subset of the set $\{1, \ldots, s\}$ of indices on variables $\mathbf{y}$

You can form a relaxation of the preceding mathematical program by removing the master constraints, which are defined by the matrices $\mathbf{D}$ and $\mathbf{B}$. The resulting constraint system, defined by the matrix $\mathbf{A}$, forms the subproblem, which can often be solved much more efficiently than the entire original problem. This is one of the key motivators for using the decomposition algorithm.

The decomposition algorithm works by finding convex combinations of extreme points of the subproblem polyhedron that satisfy the constraints defined in the master. For MILP subproblems, the strength of the relaxation is another important motivator for using this method. If the subproblem polyhedron defines feasible solutions that are close to the original feasible space, the chance of success for the algorithm increases.

The region that defines the subproblem space is often separable. That is, the formulation of the preceding mathematical program can be written in *block-angular* form as

$$
\begin{array}{llllllllll}
\text{minimize} & \mathbf{c}^1\mathbf{x}^1 & + & \mathbf{c}^2\mathbf{x}^2 & + & \cdots & + & \mathbf{c}^\kappa\mathbf{x}^\kappa & + & \mathbf{f}^\top y \\
\text{subject to} & \mathbf{D}^1\mathbf{x}^1 & + & \mathbf{D}^2\mathbf{x}^2 & + & \cdots & + & \mathbf{D}^\kappa\mathbf{x}^\kappa & + & \mathbf{B}y \quad \{\geq,=,\leq\} \ \ \mathbf{d} \\
& \mathbf{A}^1\mathbf{x}^1 & & & & & & & & \quad\quad\quad\ \ \{\geq,=,\leq\} \ \ \mathbf{b}^1 \\
& & & \mathbf{A}^2\mathbf{x}^2 & & & & & & \quad\quad\quad\ \ \{\geq,=,\leq\} \ \ \mathbf{b}^2 \\
& & & & & \ddots & & & & \quad\quad\quad\ \ \{\geq,=,\leq\} \ \ \vdots \\
& & & & & & & \mathbf{A}^\kappa\mathbf{x}^\kappa & & \quad\quad\quad\ \ \{\geq,=,\leq\} \ \ \mathbf{b}^\kappa
\end{array}
$$

$$
\underline{x}_i \leq x_i \leq \overline{x}_i, \quad x_i \in \mathbb{Z} \ \ i \in \mathcal{S}_x
$$
$$
\underline{y}_i \leq y_i \leq \overline{y}_i, \quad y_i \in \mathbb{Z} \ \ i \in \mathcal{S}_y
$$

where $K = \{1, \ldots, \kappa\}$ defines a partition of the constraints (and variables) into independent subproblems (blocks) such that $\mathbf{A} = [\mathbf{A}^1 \ldots \mathbf{A}^\kappa]$, $\mathbf{D} = [\mathbf{D}^1 \ldots \mathbf{D}^\kappa]$, $\mathbf{c} = [\mathbf{c}^1 \ldots \mathbf{c}^\kappa]$, $\mathbf{b} = [\mathbf{b}^1 \ldots \mathbf{b}^\kappa]$, $\underline{\mathbf{x}} = [\underline{\mathbf{x}}^1 \ldots \underline{\mathbf{x}}^\kappa]$, $\overline{\mathbf{x}} = [\overline{\mathbf{x}}^1 \ldots \overline{\mathbf{x}}^\kappa]$, and $\mathbf{x} = [\mathbf{x}^1 \ldots \mathbf{x}^\kappa]$. This type of structure is relatively common in modeling mathematical programs. For example, consider a model that defines a workplace that has separate departmental restrictions (defined as the subproblem constraints), which are coupled together by a company-wide budget across departments (defined as the master constraint). By relaxing the budget (master) constraint, the decomposition algorithm can take advantage of the fact that the decoupled subproblems are separable, and it can process them in parallel. A special case of block-angular form, called *block-diagonal* form, occurs when the set of master constraints is empty. In this special case, the subproblem matrices define the entire original problem.

An important indicator of a problem that is well suited for decomposition is the amount by which the subproblems cover the original problem with respect to both variables and constraints in the original presolved model. This value, which is expressed as a percentage of the original model, is known as the *coverage*. For LPs, the decomposition algorithm usually performs better than standard approaches only if the subproblems cover a significant amount of the original problem. For MILPs, the correlation between performance and coverage is more difficult to determine, because the strength of the subproblem with respect to integrality is not always proportional to the size of the system. Regardless, it is unlikely that the decomposition algorithm will outperform more standard methods (such as branch-and-cut) in problems that have small coverage.

The primary input and output for the decomposition algorithm are identical to those that are needed and produced by the OPTLP, OPTMILP, and OPTMODEL procedures. For more information, see the following sections:

- "Data Input and Output" on page 570 in Chapter 12, "The OPTLP Procedure"

- "Data Input and Output" on page 629 in Chapter 13, "The OPTMILP Procedure"

- "Details: LP Solver" on page 266 in Chapter 7, "The Linear Programming Solver"

- "Details: MILP Solver" on page 337 in Chapter 8, "The Mixed Integer Linear Programming Solver"

The only additional input that can be provided for the decomposition algorithm is an explicit definition of the partition of the subproblem constraints. The following section gives a simple example of providing this input for both PROC OPTMILP and PROC OPTMODEL.

Getting Started: Decomposition Algorithm

This example illustrates how you can use the decomposition algorithm to solve a simple mixed integer linear program. Suppose you want to solve the following problem:

$$
\begin{array}{rllllllll}
\max & x_{11} & + & 2x_{21} & + & x_{31} & + & & + & x_{22} & + & x_{32} \\
\text{subject to} & x_{11} & & & & & & x_{12} & & & & & \geq & 1 & \text{(m)} \\
& 5x_{11} & + & 7x_{21} & + & 4x_{31} & & & & & & & \leq & 11 & \text{(s1)} \\
& & & & & & & x_{12} & + & 2x_{22} & + & x_{32} & \leq & 2 & \text{(s2)} \\
\end{array}
$$

$$x_{ij} \in \{0,1\} \quad i \in \{1,\ldots,3\}, \; j \in \{1,\ldots,2\}$$

It is obvious from the structure of the problem that if constraint m is removed, then the remaining constraints s1 and s2 decompose into two independent subproblems. The next two sections describe how to solve this MILP by using the decomposition algorithm in the OPTMODEL procedure and OPTMILP procedure, respectively.

Solving a MILP with DECOMP and PROC OPTMODEL

The following statements use the OPTMODEL procedure and the decomposition algorithm to solve the MILP:

```
proc optmodel;
   var x{i in 1..3, j in 1..2} binary;

   max f =    x[1,1] + 2*x[2,1] +   x[3,1]
                     +   x[2,2] +   x[3,2];

   con m :    x[1,1] +   x[1,2]                >=   1;
   con s1:  5*x[1,1] + 7*x[2,1] + 4*x[3,1] <=  11;
   con s2:    x[1,2] + 2*x[2,2] +   x[3,2] <=   2;

   s1.block = 0;
   s2.block = 1;

   solve with milp / presolver=none decomp=(logfreq=1);
   print x;
quit;
```

Here, the PRESOLVER=NONE option is used, because otherwise the presolver solves this small instance without invoking any solver. The solution summary and optimal solution are displayed in Figure 15.1.

Figure 15.1 Solution Summary and Optimal Solution

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	f
Solution Status	Optimal
Objective Value	4
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	0
Bound Infeasibility	0
Integer Infeasibility	0
Best Bound	4
Nodes	1
Iterations	2
Presolve Time	0.00
Solution Time	0.01

	x	
	1	2
1	0	1
2	1	0
3	1	1

The iteration log, which displays the problem statistics, the progress of the solution, and the optimal objective value, is shown in Figure 15.2.

Figure 15.2 Log

```
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 6 variables (0 free, 0 fixed).
NOTE: The problem has 6 binary and 0 integer variables.
NOTE: The problem has 3 linear constraints (2 LE, 0 EQ, 1 GE, 0 range).
NOTE: The problem has 8 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value NONE is applied.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: The problem has a decomposable structure with 2 blocks. The largest block covers 33.33%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 6 (100.00%) variables and 2 (66.67%) constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter          Best        Master         Best       LP       IP  CPU Real
                   Bound     Objective      Integer      Gap      Gap Time Time
NOTE: Starting phase 1.
         1       0.0000        0.0000            .    0.00%        .   0    0
NOTE: Starting phase 2.
         2       4.0000        4.0000       4.0000    0.00%    0.00%   0    0
        Node   Active   Sols         Best         Best      Gap      CPU     Real
                                  Integer        Bound             Time     Time
         0        0      1       4.0000       4.0000    0.00%        0        0
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 0.01 seconds.
NOTE: Optimal.
NOTE: Objective = 4.
```

Solving a MILP with DECOMP and PROC OPTMILP

Alternatively, to solve the MILP with the OPTMILP procedure, create a corresponding SAS data set that uses the mathematical programming system (MPS) format as follows:

```
data mpsdata;
   input field1 $ field2 $ field3 $ field4 field5 $ field6;
   datalines;
NAME      .          mpsdata  .       .       .
ROWS      .          .        .       .       .
MAX       f          .        .       .       .
G         m          .        .       .       .
L         s1         .        .       .       .
L         s2         .        .       .       .
COLUMNS   .          .        .       .       .
```

```
     .        .MRK0000 'MARKER' .          'INTORG' .
     .        x[1,1]   f       1           m        1
     .        x[1,1]   s1      5           .        .
     .        x[2,1]   f       2           s1       7
     .        x[3,1]   f       1           s1       4
     .        x[1,2]   m       1           s2       1
     .        x[2,2]   f       1           s2       2
     .        x[3,2]   f       1           s2       1
     .        .MRK0001 'MARKER' .          'INTEND' .
RHS      .        .        .          .        .
     .        .RHS.    m       1           .        .
     .        .RHS.    s1      11          .        .
     .        .RHS.    s2      2           .        .
BOUNDS   .        .        .          .        .
UP       .BOUNDS. x[1,1]   1           .        .
UP       .BOUNDS. x[2,1]   1           .        .
UP       .BOUNDS. x[3,1]   1           .        .
UP       .BOUNDS. x[1,2]   1           .        .
UP       .BOUNDS. x[2,2]   1           .        .
UP       .BOUNDS. x[3,2]   1           .        .
ENDATA   .        .        .          .        .
;
```

Next, use the following SAS data set to define the subproblem blocks:

```
data blocks;
   input _row_ $ _block_;
   datalines;
s1 0
s2 1
;
```

Now, you can use the following OPTMILP statements to solve this MILP:

```
proc optmilp
   data      = mpsdata
   presolver  = none;
   decomp
      logfreq = 1
      blocks  = blocks;
run;
```

Syntax: Decomposition Algorithm

You can specify the decomposition algorithm either by using options in a SOLVE statement in the OPT-MODEL procedure or by using statements in the OPTLP and OPTMILP procedures. Except for the fact that you use SOLVE statement options in PROC OPTMODEL or you use statements in PROC OPTLP and PROC OPTMILP, the syntax is identical.

The following decomposition algorithm options are available in the SOLVE statement in the OPTMODEL procedure:

SOLVE WITH LP / *< options >*
        < **DECOMP***< =(decomp-options) > >*
        < **DECOMP_MASTER=(***< decomp-master-options >***)** *>*
        < **DECOMP_SUBPROB=(***< decomp-subprob-options >***)** *>* **;**

SOLVE WITH MILP / *< options >*
        < **DECOMP***< =(decomp-options) > >*
        < **DECOMP_MASTER=(***< decomp-master-options >***)** *>*
        < **DECOMP_MASTER_IP=(***< decomp-master-ip-options >***)** *>*
        < **DECOMP_SUBPROB=(***< decomp-subprob-options >***)** *>* **;**

The following statements are available in the OPTLP procedure:

PROC OPTLP *< options >* **;**
    DECOMP *< decomp-options >* **;**
    DECOMP_MASTER *< decomp-master-options >* **;**
    DECOMP_SUBPROB *< decomp-subprob-options >* **;**

The following statements are available in the OPTMILP procedure:

PROC OPTMILP *< options >* **;**
    DECOMP *< decomp-options >* **;**
    DECOMP_MASTER *< decomp-master-options >* **;**
    DECOMP_MASTER_IP *< decomp-master-ip-options >* **;**
    DECOMP_SUBPROB *< decomp-subprob-options >* **;**

Decomposition Algorithm Options in the PROC OPTLP Statement or the SOLVE WITH LP Statement in PROC OPTMODEL

To solve a linear program, you can specify the decomposition algorithm in a SOLVE WITH LP statement in the OPTMODEL procedure or in a PROC OPTLP statement in the OPTLP procedure. To control the overall decomposition algorithm, you can specify one or more of the LP solver options that are shown in Table 15.1. (As indicated, you can specify some options only in the PROC OPTLP statement.)

The options in Table 15.1 control the overall process flow for solving a linear program, and they are equivalent to the options that are used in PROC OPTLP and PROC OPTMODEL with standard methods. These options are called main solver options in this chapter. They are described in detail in the section "Syntax: LP Solver" on page 259 in Chapter 7, "The Linear Programming Solver" and the section "Syntax: OPTLP Procedure" on page 562 in Chapter 12, "The OPTLP Procedure." The DUALIZE= option has a different default when you use the decomposition algorithm, as shown in Table 15.1.

Table 15.1 LP Options in the PROC OPTLP Statement or
SOLVE WITH LP Statement

Description	option	Different Default
Data Set Options (OPTLP procedure only)		
Specifies the input data set	DATA=	
Specifies the dual solution output data set	DUALOUT=	

Table 15.1 (continued)

Description	*option*	Different Default
Specifies whether the model is a maximization or minimization problem	OBJSENSE=	
Specifies the primal solution output data set	PRIMALOUT=	
Saves output data sets only if optimal	SAVE_ONLY_IF_OPTIMAL	
Presolve Options		
Controls the dualization of the problem	DUALIZE=	OFF
Specifies the type of presolve	PRESOLVER=	
Control Options		
Specifies the feasibility tolerance	FEASTOL=	
Specifies how frequently to print the solution progress	LOGFREQ=	
Specifies the level of detail of solution progress to print in the log	LOGLEVEL=	
Specifies the maximum number of iterations	MAXITER=	
Specifies the time limit for the optimization process	MAXTIME=	
Specifies the optimality tolerance	OPTTOL=	
Enables or disables printing summary (OPTLP procedure only)	PRINTLEVEL=	
Specifies whether time units are CPU time or real time	TIMETYPE=	
Algorithm Options		
Enables or disables scaling of the problem	SCALE=	

Decomposition Algorithm Options in the PROC OPTMILP Statement or the SOLVE WITH MILP Statement in PROC OPTMODEL

To solve a mixed integer linear program, you can specify the decomposition algorithm in a SOLVE WITH MILP statement in the OPTMODEL procedure or in a PROC OPTMILP statement in the OPTMILP procedure. To control the overall decomposition algorithm, you can specify one or more of the MILP solver options shown in Table 15.2. (As indicated, you can specify some options only in the PROC OPTMILP statement.)

The options in Table 15.2 control the overall process flow for solving a mixed integer linear program, and they are equivalent to the options that are used in the OPTMILP and OPTMODEL procedures with standard methods. These options are called main solver options in this chapter. They are described in detail in the section "Syntax: MILP Solver" on page 325 and the section "Syntax: OPTMILP Procedure" on page 617.

The HYBRID= option in the DECOMP statement indicates the processing mode for the root node of the branch-and-bound search tree. When HYBRID=ON, the root node is first processed using standard MILP techniques, as described in the section "Details: MILP Solver" on page 337. The default setting for the decomposition algorithm is HYBRID=OFF. In this case, the root processing is done solely by the decomposition algorithm, and several of the direct MILP options are ignored. These options are indicated in Table 15.2 in the column Ignored HYBRID=OFF.

Table 15.2 MILP Options in the PROC OPTMILP Statement or
SOLVE WITH MILP Statement

Description	*option*	Ignored HYBRID=OFF
Data Set Options (OPTMILP procedure only)		
Specifies the input data set	DATA=	
Specifies the constraint activities output data set	DUALOUT=	
Specifies whether the model is a maximization or minimization problem	OBJSENSE=	
Specifies the primal solution input data set (warm start)	PRIMALIN=	
Specifies the primal solution output data set	PRIMALOUT=	
Presolve Option		
Specifies the type of presolve	PRESOLVER=	
Control Options		
Specifies the stopping criterion based on an absolute objective gap	ABSOBJGAP=	
Emphasizes feasibility or optimality	EMPHASIS=	X
Specifies the maximum violation of variables and constraints	FEASTOL=	
Specifies the maximum allowed difference between an integer variable's value and an integer	INTTOL=	
Specifies how frequently to print the node log	LOGFREQ=	
Specifies the level of detail of solution progress to print in the log	LOGLEVEL=	
Specifies the maximum number of nodes to be processed	MAXNODES=	
Specifies the maximum number of solutions to be found	MAXSOLS=	
Specifies the time limit for the optimization process	MAXTIME=	
Specifies the tolerance used in determining the optimality of nodes in the branch-and-bound tree	OPTTOL=	
Uses the input primal solution (warm start) (OPTMODEL procedure only)	PRIMALIN	
Enables or disables printing summary (OPTMILP procedure only)	PRINTLEVEL=	
Specifies the probing level	PROBE=	
Specifies the stopping criterion based on a relative objective gap	RELOBJGAP=	
Specifies the scale of the problem matrix	SCALE=	
Specifies the initial seed for the random number generator	SEED=	X
Specifies the stopping criterion based on target objective value	TARGET=	X
Specifies whether time units are CPU time or real time	TIMETYPE=	
Heuristics Option		
Specifies the primal heuristics level	HEURISTICS=	
Search Options		
Specifies the level of symmetry detection	SYMMETRY=	X
Specifies the rule for selecting the branching variable	VARSEL=	
Cut Options		
Specifies the cut level for all cuts	ALLCUTS=	X
Specifies the clique cut level	CUTCLIQUE=	X
Specifies the flow cover cut level	CUTFLOWCOVER=	X
Specifies the flow path cut level	CUTFLOWPATH=	X
Specifies the Gomory cut level	CUTGOMORY=	X
Specifies the generalized upper bound (GUB) cover cut level	CUTGUB=	X

Table 15.2 (continued)

Description	*option*	Ignored HYBRID=OFF
Specifies the implied bounds cut level	CUTIMPLIED=	X
Specifies the knapsack cover cut level	CUTKNAPSACK=	X
Specifies the lift-and-project cut level	CUTLAP=	X
Specifies the mixed lifted 0-1 cut level	CUTMILIFTED=	X
Specifies the mixed integer rounding (MIR) cut level	CUTMIR=	X
Specifies the row multiplier factor for cuts	CUTSFACTOR=	X
Specifies the overall cut aggressiveness	CUTSTRATEGY=	X
Specifies the zero-half cut level	CUTZEROHALF=	X

The following search option, listed in Table 15.2, has a different set of options from what is described in the MILP solver sections.

VARSEL=*number* | *string*

specifies the rule for selecting the branching variable. The values of *string* and the corresponding values of *number* are listed in Table 15.3.

Table 15.3 Values for VARSEL= Option

number	*string*	Description
–1	AUTOMATIC	Uses automatic branching variable selection.
0	MAXINFEAS	Chooses the variable in the original compact formulation with maximum infeasibility.
4	RYANFOSTER	When appropriate, uses a specialized branching rule known as *Ryan-Foster branching*.

The default value is AUTOMATIC. For details about variable selection, see the sections "Variable Selection" on page 633 and "Special Case: Identical Blocks and Ryan-Foster Branching" on page 728.

DECOMP Statement

> **DECOMP** < *decomp-options* > **;**

The DECOMP statement controls the overall decomposition algorithm.

Table 15.4 summarizes the *decomp-options* available in the DECOMP statement. These options control the overall decomposition algorithm process flow during the solution of an LP or a MILP. (As indicated, you can specify the data set options only in the OPTLP or OPTMILP procedure, and you can specify some control options only for a MILP.)

Table 15.4 Options in the DECOMP Statement

Description	*decomp-option*
Data Set Options (OPTLP and OPTMILP procedures only)	
Specifies the blocks input data set	BLOCKS=

Table 15.4 (continued)

Description	decomp-option
Control Options	
Specifies the stopping criterion based on an absolute objective gap	ABSOBJGAP=
Specifies the frequency of removing ineffective columns from the master LP	COMPRESSFREQ=
Specifies whether or not to first process the root node by using standard MILP techniques	HYBRID=
Specifies whether to initialize the columns by solving each block with the original cost vector	INITVARS=
Specifies the level of detail of solution progress to print in the log	LOGLEVEL=
Specifies the maximum number of blocks to allow	MAXBLOCKS=
Specifies the maximum number of new columns to allow into the master each pass	MAXCOLSPASS=
Specifies the maximum amount of time spent in the decomposition algorithm	MAXTIME=
Specifies the decomposition algorithm method	METHOD=
Specifies the number of blocks to search for by using METHOD=AUTO	NBLOCKS=
Specifies the number of threads to use in the overall decomposition algorithm	NTHREADS=
Specifies the stopping criterion based on relative objective gap	RELOBJGAP=
Control Options (MILP only)	
Specifies how frequently to print the continuous iteration log	LOGFREQ=
Specifies whether the master problem is solved as a MILP with the current set of columns at the beginning of phase II	MASTER_IP_BEG=
Specifies whether the master problem is solved as a MILP with the current set of columns at the end of phase II	MASTER_IP_END=
Specifies the frequency of solving the master problem as a MILP with the current set of columns	MASTER_IP_FREQ=
Specifies the maximum number of outer iterations for the decomposition algorithm	MAXITER=

The following list describes the *decomp-options* in detail.

ABSOBJGAP=*number*

> specifies a stopping criterion for the continuous bound of the decomposition. When the absolute difference between the master objective and the best dual bound falls below the value of *number*, the decomposition algorithm stops adding columns. The value of *number* can be any nonnegative number. The default value is the value of the OPTTOL= main solver option.

BLOCKS=*SAS-data-set*

> specifies (for OPTLP and OPTMILP procedures only) the input data set that contains block definitions to be used by the decomposition algorithm if METHOD=USER. See the section "The BLOCKS= Data Set in PROC OPTMILP and PROC OPTLP" on page 726 for more information. To specify blocks in PROC OPTMODEL, use the `.block` constraint suffix instead (see the section "The `.block` Constraint Suffix in PROC OPTMODEL" on page 726).

COMPRESSFREQ=*number*

removes ineffective columns from the master LP after every *number* of iterations. The frequency, *number*, is an integer between 0 and the largest four-byte signed integer, which is $2^{31} - 1$. The default value is 0.

HYBRID=*number | string*

specifies whether to first process the root node by using standard MILP techniques, as described in the section "Details: MILP Solver" on page 337.

Table 15.5 describes the valid values of the HYBRID= option.

Table 15.5 Values for HYBRID= Option

number	string	Description
0	OFF	Disables root processing by standard MILP techniques.
1	ON	Enables root processing by standard MILP techniques.

The default is OFF.

INITVARS=*number | string*

specifies whether to initialize the columns by using the original cost vector to solve each block.

Table 15.6 describes the valid values of the INITVARS= option.

Table 15.6 Values for INITVARS= Option

number	string	Description
0	OFF	Disables initializing the columns by using the original cost vector to solve each block.
1	ON	Enables initializing the columns by using the original cost vector to solve each block.

This option must be set to ON when used with METHOD=CONCOMP. The default is ON.

LOGFREQ=*number*

specifies (for MILP problems only) how often to print information in the continuous iteration log. The value of *number* can be any nonnegative number up to the largest four-byte signed integer, which is $2^{31} - 1$. The default value of *number* is 10. If *number* is set to 0, then the iteration log is disabled. If *number* is positive, then an entry is made in the log at the first iteration, at the last iteration, and at intervals that are dictated by the value of *number*. An entry is also made each time a better integer solution or improved bound is found.

LOGLEVEL=*number | string*

controls the amount of information that is displayed in the SAS log by the decomposition algorithm. Table 15.7 and Table 15.8 provide the valid values for this option and a description of what is displayed in the log when an LP and a MILP, respectively, is solved.

Table 15.7 Values for LOGLEVEL= Option for an LP

number	string	Description
−1	AUTOMATIC	Prints the continuous iteration log at the interval dictated by the LOGFREQ= main solver option.
0	NONE	Turns off printing of all of the decomposition algorithm messages to the SAS log.
1	BASIC	Prints the continuous iteration log at the interval dictated by the LOGFREQ= main solver option.
2	MODERATE	Prints the continuous iteration log and summary information for each iteration at the interval dictated by the LOGFREQ= main solver option.
3	AGGRESSIVE	Prints the continuous iteration log and detailed information for each iteration at the interval dictated by the LOGFREQ= main solver option.

Table 15.8 Values for LOGLEVEL= Option for a MILP

number	string	Description
−1	AUTOMATIC	Prints the continuous iteration log for the root node at the interval dictated by the LOGFREQ= option in the DECOMP statement. Prints the branch-and-bound node log at the interval dictated by the LOGFREQ= main solver option.
0	NONE	Turns off printing of all of the decomposition algorithm messages to the SAS log.
1	BASIC	Prints the continuous iteration log for each branch-and-bound node at the interval dictated by the LOGFREQ= option in the DECOMP statement.
2	MODERATE	Prints the continuous iteration log and summary information for each iteration of each branch-and-bound node at the interval dictated by the LOGFREQ= option in the DECOMP statement.
3	AGGRESSIVE	Prints the continuous iteration log and detailed information for each iteration of each branch-and-bound node at the interval dictated by the LOGFREQ= option in the DECOMP statement.

The default is AUTOMATIC for both LPs and MILPs.

MASTER_IP_BEG=number | string

specifies (for MILP problems only) whether the master problem is solved as a MILP with the current set of columns at the beginning of phase II. Table 15.9 describes the valid values of the MASTER_IP_BEG= option.

Table 15.9 Values for MASTER_IP_BEG= Option

number	string	Description
0	OFF	Disables solving the master as a MILP at the beginning of phase II.
1	ON	Enables solving the master as a MILP at the beginning of phase II.

The default is ON in the root node and 0 elsewhere.

MASTER_IP_END=_number | string_

specifies (for MILP problems only) whether the master problem is solved as a MILP with the current set of columns at the end of phase II. Table 15.10 describes the valid values of the MASTER_IP_END= option.

Table 15.10 Values for MASTER_IP_END= Option

number	string	Description
0	OFF	Disables solving the master as a MILP at the end of phase II.
1	ON	Enables solving the master as a MILP at the end of phase II.

The default is ON in the root node and 0 elsewhere.

MASTER_IP_FREQ=_number_

solves the master problem (for MILP problems only) as a MILP with the current set of columns after every *number* iterations. The frequency, *number*, is an integer between 0 and the largest four-byte signed integer, which is $2^{31} - 1$. The default is 10 in the root node and 0 elsewhere.

MAXBLOCKS=_number_

specifies the maximum number of blocks to allow. If the defined number of blocks exceeds *number*, the algorithm creates superblocks using a very simple round-robin scheme. The value of *number* can be any positive number; the default value is the positive number that has the largest absolute value that can be represented in your operating environment.

MAXCOLSPASS=_number_

specifies the maximum number of new columns to allow into the master at each pass. This option is disabled on the initial pass if INITVARS=1. The value of *number* can be any positive number; the default value is the positive number that has the largest absolute value that can be represented in your operating environment.

MAXITER=_number_

specifies (for MILP problems only) the maximum number of outer iterations for the decomposition algorithm. The value *number* can be any integer between 1 and the largest four-byte signed integer, which is $2^{31} - 1$. If you do not specify this option, the procedure does not stop based on the number of iterations performed.

MAXTIME=_number_

specifies an upper limit of *number* seconds of time for the decomposition algorithm. The value of the TIMETYPE= main solver option determines the type of units used. If you do not specify this option, the procedure does not stop based on the amount of time elapsed. The value of *number* can be any positive number; the default value is the positive number that has the largest absolute value that can be represented in your operating environment.

METHOD=_string_

specifies the decomposition algorithm method as shown in Table 15.11.

Table 15.11 Values for METHOD= Option

string	Description
AUTO	The algorithm attempts to find a block-angular structure in the constraint matrix by using matrix-stretching techniques similar to what is described in Grcar (1990) and Aykanat, Pinar, and Çatalyürek (2004). The NBLOCKS= option specifies the number of blocks into which the algorithm attempts to decompose the constraint matrix. If the algorithm fails to find a decomposition, the MILP solver is called directly.
CONCOMP	The algorithm attempts to find a block-diagonal (not block-angular) structure in the constraint matrix. Unless your problem separates into completely independent problems with no linking constraints, this method finds only one block and hence is equivalent to calling the MILP solver directly.
NETWORK	The algorithm attempts to find an embedded network similar to what is described in the section "The Network Simplex Algorithm" on page 266. The weakly connected components of this network are used as the blocks.
USER	The user defines which rows belong to which blocks (sub-problems). In PROC OPTMODEL, use the **.block** constraint suffix. In PROC OPTLP and PROC OPTMILP, use the BLOCKS= data set instead.

The default is USER if blocks are defined and AUTO otherwise.

NBLOCKS=_number_

specifies the initial number of blocks to search for when you specify METHOD=AUTO. If the algorithm is unable to find a block-angular structure that contains this number of blocks, it repeatedly attempts to find an appropriate structure that contains half the previously attempted number of blocks. If the algorithm fails to find a decomposition that contains at least two blocks, then the standard MILP solver is called directly. The value of *number* can be any positive number less than or equal to the number of rows in the presolved model; the default value is the number of threads that are used for processing. In single-machine mode, this is equivalent to the setting of the NTHREADS= option in the PERFORMANCE statement or the DECOMP statement. In distributed mode, this is equivalent to the number of compute nodes that are set using the NODES= option in the PERFORMANCE statement times the number of threads set for each compute node. For more information about parallel execution, see the section "Parallel Processing" on page 727.

NTHREADS=_number_

specifies the number of threads to be used by the overall block decomposition algorithm. This overrides the NTHREADS= option in the PERFORMANCE statement, which is described in the section "PERFORMANCE Statement" on page 23.

RELOBJGAP=_number_

specifies the relative objective gap as a stopping criterion. The relative objective gap is based on the master objective (MasterObjective) and the best dual bound (BestBound); it is equal to

$$| \text{MasterObjective} - \text{BestBound} | \, / \, (1\text{E}{-}10 + | \text{BestBound} |)$$

When this value becomes smaller than the specified gap size _number_, the decomposition algorithm stops adding columns. The value of _number_ can be any nonnegative number. For LP, the default value is 0; for MILP, the default value is 1e-4.

DECOMP_MASTER Statement

DECOMP_MASTER < _decomp-master-options_ > ;

MASTER < _decomp-master-options_ > ;

The DECOMP_MASTER statement controls the master problem.

Table 15.12 summarizes the options available in the DECOMP_MASTER statement. These options control the master LP solver in the decomposition algorithm during the solution of an LP or a MILP. (As indicated, you can specify the PRINTLEVEL= option only in the OPTLP procedure.) For descriptions of these options, see the section "LP Solver Options" on page 260 in Chapter 7, "The Linear Programming Solver" and the section "PROC OPTLP Statement" on page 564 in Chapter 12, "The OPTLP Procedure." Some options have different defaults when you use the decomposition algorithm, as indicated in Table 15.12.

Table 15.12 Options in the DECOMP_MASTER Statement

Description	_decomp-master-option_	Different Default
Algorithm Option		
Specifies the master algorithm	ALGORITHM=	PS[†]
Presolve Option		
Controls the dualization of the problem	DUALIZE=	OFF
Specifies, for the first master solve only, the type of presolve	INITPRESOLVER=	
Specifies the type of presolve	PRESOLVER=	NONE (ALGORITHM=PS) [†]
Control Options		
Specifies the feasibility tolerance	FEASTOL=	1E–7
Specifies how frequently to print the solution progress	LOGFREQ=	
Specifies the level of detail of solution progress to print in the log	LOGLEVEL=	
Specifies the maximum number of iterations	MAXITER=	
Specifies the time limit for the optimization process	MAXTIME=	
Specifies the number of threads to use in the master solver	NTHREADS=	
Specifies the optimality tolerance	OPTTOL=	1E–7
Enables or disables printing summary (OPTLP procedure only)	PRINTLEVEL=	
Specifies whether time units are CPU time or real time	TIMETYPE=	

Table 15.12 (continued)

Description	*decomp-master-option*	Different Default
Specifies the type of initial basis	BASIS=	WARMSTART (ALGORITHM=PS)[†]
Specifies the type of pricing strategy	PRICETYPE=	
Specifies the queue size for determining the entering variable	QUEUESIZE=	
Enables or disables scaling of the problem	SCALE=	
Specifies the initial seed for the random number generator	SEED=	
Interior Point Algorithm Options		
Enables or disables interior crossover	CROSSOVER=	
Specifies the stopping criterion based on a duality gap	STOP_DG=	
Specifies the stopping criterion based on dual infeasibility	STOP_DI=	
Specifies the stopping criterion based on primal infeasibility	STOP_PI=	

† The different defaults (ALGORITHM=PS, PRESOLVER=NONE, and BASIS=WARMSTART) are motivated by the fact that primal feasibility of the master problem is preserved when columns are added, so a warm start from the previous optimal basis tends to be more efficient than solving the master from scratch in each iteration.

The following options, listed in Table 15.12, are specific to the DECOMP_MASTER statement and are not described in the LP solver sections.

INITPRESOLVER=*number | string*

INITPRESOL=*number | string*

   specifies, for the first master solve only, presolve conditions as shown in Table 15.13.

Table 15.13 Values for INITPRESOLVER= Option

number	string	Description
−1	AUTOMATIC	Applies the default level of presolve processing.
0	NONE	Disables presolver.
1	BASIC	Performs minimal presolve processing.
2	MODERATE	Applies a higher level of presolve processing.
3	AGGRESSIVE	Applies the highest level of presolve processing.

   The default is AUTOMATIC.

NTHREADS=*number*

   specifies the number of threads to use in the master solver (if the chosen solver method supports multithreading). By default, the number of threads is the same as the setting that is used for the NTHREADS= option in the DECOMP statement.

DECOMP_MASTER_IP Statement

> **DECOMP_MASTER_IP** < *decomp-master-ip-options* > **;**

> **MASTER_IP** < *decomp-master-ip-options* > **;**

For mixed integer linear programming problems, the DECOMP_MASTER_IP statement controls the (restricted) master problem, which is solved as a MILP with the current set of columns in an effort to obtain an integer-feasible solution.

Table 15.14 summarizes the options available in the DECOMP_MASTER_IP statement. These options control the MILP solver that is used to solve the integer version of the master problem. For descriptions of these options, see the section "MILP Solver Options" on page 327 in Chapter 8, "The Mixed Integer Linear Programming Solver" and the section "PROC OPTMILP Statement" on page 618 in Chapter 13, "The OPTMILP Procedure." Some options have different defaults when you use the decomposition algorithm, as shown in Table 15.14.

Table 15.14   Options in the DECOMP_MASTER_IP Statement

Description	*decomp-master-ip-option*	Different Default
Presolve Option		
Specifies the type of presolve	PRESOLVER=	
Control Options		
Specifies the stopping criterion based on an absolute objective gap	ABSOBJGAP=	
Specifies the cutoff value for node removal	CUTOFF=	
Emphasizes feasibility or optimality	EMPHASIS=	
Specifies the maximum violation on variables and constraints	FEASTOL=	1E–7
Specifies the maximum allowed difference between an integer variable's value and an integer	INTTOL=	
Specifies how frequently to print the node log	LOGFREQ=	
Specifies the level of detail of solution progress to print in the log	LOGLEVEL=	
Specifies the maximum number of nodes to be processed	MAXNODES=[†]	
Specifies the maximum number of solutions to be found	MAXSOLS=	
Specifies the time limit for the optimization process	MAXTIME=	
Specifies the number of threads to use in the master integer solver	NTHREADS=	
Specifies the tolerance used when deciding on the optimality of nodes in the branch-and-bound tree	OPTTOL=	1E–7
Specifies whether to use the previous best primal solution as a warm start	PRIMALIN=	
Specifies the probing level	PROBE=	
Specifies the stopping criterion based on a relative objective gap	RELOBJGAP=	0.01
Specifies the scale of the problem matrix	SCALE=	
Specifies the stopping criterion based on the target objective value	TARGET=	

Table 15.14 (continued)

Description	*decomp-master-ip-option*	Different Default
Specifies whether time units are CPU time or real time	TIMETYPE=	
Heuristics Option		
Specifies the primal heuristics level	HEURISTICS=	
Search Options		
Specifies the level of conflict search	CONFLICTSEARCH=	
Specifies the node selection strategy	NODESEL=	
Specifies the restarting strategy	RESTARTS=	
Specifies the initial seed for the random number generator	SEED=	
Specifies the number of simplex iterations performed on each variable in strong branching strategy	STRONGITER=	
Specifies the number of candidates for strong branching	STRONGLEN=	
Specifies the level of symmetry detection	SYMMETRY=	
Specifies the rule for selecting branching variable	VARSEL=	
Cut Options		
Specifies the cut level for all cuts	ALLCUTS=	
Specifies the clique cut level	CUTCLIQUE=	
Specifies the flow cover cut level	CUTFLOWCOVER=	
Specifies the flow path cut level	CUTFLOWPATH=	
Specifies the Gomory cut level	CUTGOMORY=	
Specifies the generalized upper bound (GUB) cover cut level	CUTGUB=	
Specifies the implied bounds cut level	CUTIMPLIED=	
Specifies the knapsack cover cut level	CUTKNAPSACK=	
Specifies the lift-and-project cut level	CUTLAP=	
Specifies the mixed lifted 0-1 cut level	CUTMILIFTED=	
Specifies the mixed integer rounding (MIR) cut level	CUTMIR=	
Specifies the row multiplier factor for cuts	CUTSFACTOR=	
Specifies the overall cut aggressiveness	CUTSTRATEGY=	
Specifies the zero-half cut level	CUTZEROHALF=	

† MAXNODES=100000 in the root node, and MAXNODES=10000 in nodes that are not the root.

The following options are listed in Table 15.14 but are not described in the MILP solver sections. These options are specific to the DECOMP_MASTER_IP statement.

NTHREADS=*number*

    specifies the number of threads to use in the master integer solver (if the chosen solver method supports multithreading). By default, the number of threads is the same as the setting that is used for the NTHREADS= option in the DECOMP statement.

PRIMALIN=*number* | *string*

PIN=*number* | *string*

    specifies whether the MILP solver is to use the previous best solution's variables values as a starting solution (warm start). If the MILP solver finds that the input solution is feasible, then the input solution provides an incumbent solution and a bound for the branch-and-bound algorithm. If the solution is not

feasible, the MILP solver tries to repair it. When it is difficult to find a good integer-feasible solution for a problem, warm start can reduce solution time significantly. Table 15.15 describes the valid values of the PRIMALIN= option.

Table 15.15   Values for PRIMALIN= Option

number	string	Description
0	OFF	Ignores the previous solution.
1	ON	Starts from the previous solution.

The default is ON.

DECOMP_SUBPROB Statement

DECOMP_SUBPROB < *decomp-subprob-options* > **;**

SUBPROB < *decomp-subprob-options* > **;**

The DECOMP_SUBPROB statement controls the subproblem.

Table 15.16 summarizes the options available for the decomposition algorithm in the DECOMP_SUBPROB statement when the subproblem algorithm chosen is an LP algorithm. (As indicated, you can specify the PRINTLEVEL= option only in the OPTLP procedure.) For descriptions of these options, see the section "LP Solver Options" on page 260 in Chapter 7, "The Linear Programming Solver" and the section "PROC OPTLP Statement" on page 564 in Chapter 12, "The OPTLP Procedure." Some options have different defaults when you use the decomposition algorithm, as shown in Table 15.16.

Table 15.16   Options in the DECOMP_SUBPROB Statement Used with an LP Algorithm

Description	*decomp-subprob-option*	Different Default
Algorithm Option		
Specifies the subproblem algorithm	ALGORITHM=	PS (METHOD=USER) NETWORK_PURE (METHOD=NETWORK)[†]
Presolve Option		
Controls the dualization of the problem	DUALIZE=	OFF
Specifies, for the first subproblem solve only, the type of presolve	INITPRESOLVER=	
Specifies the type of presolve	PRESOLVER=	NONE (ALGORITHM=PS)[†]
Control Options		
Specifies the feasibility tolerance	FEASTOL=	1E–7
Specifies how frequently to print the solution progress	LOGFREQ=	
Specifies the level of detail of solution progress to print in the log	LOGLEVEL=	
Specifies the maximum number of iterations	MAXITER=	

Table 15.16 (continued)

Description	decomp-subprob-option	Different Default
Specifies the time limit for the optimization process	MAXTIME=	
Specifies the number of threads to use in the subproblem solver	NTHREADS=	
Specifies the optimality tolerance	OPTTOL=	1E–7
Enables or disables printing summary (OPTLP procedure only)	PRINTLEVEL=	
Specifies the initial seed for the random number generator	SEED=	
Specifies whether time units are CPU time or real time	TIMETYPE=	
Simplex Algorithm Options		
Specifies the type of initial basis	BASIS=	WARMSTART (ALGORITHM=PS)†
Specifies the type of pricing strategy	PRICETYPE=	
Specifies the queue size for determining entering variable	QUEUESIZE=	
Enables or disables scaling of the problem	SCALE=	
Interior Point Algorithm Options		
Enables or disables interior crossover	CROSSOVER=	
Specifies the stopping criterion based on duality gap	STOP_DG=	
Specifies the stopping criterion based on dual infeasibility	STOP_DI=	
Specifies the stopping criterion based on primal infeasibility	STOP_PI=	

† When METHOD=USER is specified in the DECOMP statement, ALGORITHM=PS, PRESOLVER=NONE, and BASIS=WARMSTART by default. These defaults are motivated by the fact that primal feasibility of the subproblem is preserved when the objective is changed, so a warm start from the previous optimal basis tends to be more efficient than solving the subproblem from scratch in each iteration. When METHOD=NETWORK, ALGORITHM=NETWORK_PURE by default because each subproblem is a pure network, causing the specialized pure network solver to usually be the most efficient choice.

Table 15.17 summarizes the options available in the DECOMP_SUBPROB statement when the subproblem algorithm chosen is a MILP algorithm. When the subproblem consists of multiple blocks (a block-diagonal structure), these settings apply to all subproblems. For descriptions of these options, see the section "MILP Solver Options" on page 327 in Chapter 8, "The Mixed Integer Linear Programming Solver" and the section "PROC OPTMILP Statement" on page 618 in Chapter 13, "The OPTMILP Procedure."

Table 15.17 Options in the DECOMP_SUBPROB Statement
Used with a MILP Algorithm

Description	*decomp-subprob-option*	Different Default
Algorithm Option		
Specifies the subproblem algorithm	ALGORITHM=	
Presolve Option		
Specifies, for the first subproblem solve only, the type of presolve	INITPRESOLVER=	
Specifies the type of presolve	PRESOLVER=	
Control Options		
Specifies the stopping criterion based on absolute objective gap	ABSOBJGAP=	
Emphasizes feasibility or optimality	EMPHASIS=	
Specifies the maximum violation on variables and constraints	FEASTOL=	1E–7
Specifies the maximum allowed difference between an integer variable's value and an integer	INTTOL=	
Specifies how frequently to print the node log	LOGFREQ=	
Specifies the level of detail of solution progress to print in the log	LOGLEVEL=	
Specifies the maximum number of nodes to be processed	MAXNODES=	
Specifies the maximum number of solutions to be found	MAXSOLS=	
Specifies the time limit for the optimization process	MAXTIME=	
Specifies the number of threads to use in the subproblem solver	NTHREADS=	
Specifies the tolerance used when deciding on the optimality of nodes in the branch-and-bound tree	OPTTOL=	1E–7
Specifies whether to use the previous best primal solution as a warm start	PRIMALIN=	
Specifies the probing level	PROBE=	
Specifies the stopping criterion based on relative objective gap	RELOBJGAP=	
Specifies the scale of the problem matrix	SCALE=	
Specifies the stopping criterion based on target objective value	TARGET=	
Specifies whether time units are CPU time or real time	TIMETYPE=	
Heuristics Option		
Specifies the primal heuristics level	HEURISTICS=	
Search Options		
Specifies the level of conflict search	CONFLICTSEARCH=	
Specifies the node selection strategy	NODESEL=	
Specifies the restarting strategy	RESTARTS=	

Table 15.17 (continued)

Description	*decomp-subprob-option*	Different Default
Specifies the initial seed for the random number generator	SEED=	
Specifies the number of simplex iterations performed on each variable in strong branching strategy	STRONGITER=	
Specifies the number of candidates for strong branching	STRONGLEN=	
Specifies the level of symmetry detection	SYMMETRY=	
Specifies the rule for selecting branching variable	VARSEL=	
Cut Options		
Specifies the cut level for all cuts	ALLCUTS=	
Specifies the clique cut level	CUTCLIQUE=	
Specifies the flow cover cut level	CUTFLOWCOVER=	
Specifies the flow path cut level	CUTFLOWPATH=	
Specifies the Gomory cut level	CUTGOMORY=	
Specifies the generalized upper bound (GUB) cover cut level	CUTGUB=	
Specifies the implied bounds cut level	CUTIMPLIED=	
Specifies the knapsack cover cut level	CUTKNAPSACK=	
Specifies the lift-and-project cut level	CUTLAP=	
Specifies the mixed lifted 0-1 cut level	CUTMILIFTED=	
Specifies the mixed integer rounding (MIR) cut level	CUTMIR=	
Specifies the row multiplier factor for cuts	CUTSFACTOR=	
Specifies the overall cut aggressiveness	CUTSTRATEGY=	
Specifies the zero-half cut level	CUTZEROHALF=	

The following options, listed in Table 15.16 and Table 15.17, are specific to the DECOMP_SUBPROB statement and are not described in the LP or MILP solver sections.

ALGORITHM=*string*

SOLVER=*string*

SOL=*string*

> specifies one of the algorithms shown in Table 15.18 (the valid abbreviated value for each *string* is shown in parentheses).

Table 15.18 Values for ALGORITHM= Option

string	Description
PRIMAL (PS)	Uses the primal simplex algorithm.
DUAL (DS)	Uses the dual simplex algorithm.
NETWORK (NS)	Uses the network simplex algorithm.
NETWORK_PURE (NSPURE)	Uses the network simplex algorithm for pure networks.
INTERIORPOINT (IP)	Uses the interior point algorithm.
MILP	Uses the mixed integer linear solver.

The default is NETWORK_PURE if METHOD=NETWORK, MILP for mixed integer linear programming subproblems, or PS for linear programming subproblems.

INITPRESOLVER=*number | string*

INITPRESOL=*number | string*

specifies, for the first subproblem solve only, presolve conditions as listed in Table 15.19.

Table 15.19 Values for INITPRESOLVER= Option

number	string	Description
–1	AUTOMATIC	Applies the default level of presolve processing
0	NONE	Disables presolver
1	BASIC	Performs minimal presolve processing
2	MODERATE	Applies a higher level of presolve processing
3	AGGRESSIVE	Applies the highest level of presolve processing

The default is AUTOMATIC.

NTHREADS=*number*

specifies the number of threads to use in the subproblem solver (if the chosen solver method supports multithreading). By default, the number of threads is either 1 or the setting that is used for the NTHREADS= option in the DECOMP statement divided by the number of blocks in the decomposition (rounded down), whichever is greater.

PRIMALIN=*number | string*

PIN=*number | string*

specifies (for MILP problems only) whether the MILP solver is to use the values of the previous best solution's variables as a starting solution (warm start). If the MILP solver finds that the input solution is feasible, then the input solution provides an incumbent solution and a bound for the branch-and-bound algorithm. If the solution is not feasible, the MILP solver tries to repair it. When it is difficult to find a good integer-feasible solution for a problem, warm start can reduce solution time significantly. Table 15.20 describes the valid values of the PRIMALIN= option.

Table 15.20 Values for PRIMALIN= Option

number	string	Description
0	OFF	Ignores the previous solution.
1	ON	Starts from the previous solution.

The default is ON.

Details: Decomposition Algorithm

Data Input

This subsection describes the format for describing the partition of the constraint system that defines the subproblem blocks. In the OPTLP and OPTMILP procedures, partitioning is done by using a data set specified in the BLOCKS= data option in the DECOMP statement. In PROC OPTMODEL, partitioning is done by using the `.block` suffix on constraints.

The blocks must be disjoint with respect to variables. If two blocks contain a nonzero coefficient for the same variable, the decomposition algorithm produces an error that contains information about where the blocks overlap.

The BLOCKS= Data Set in PROC OPTMILP and PROC OPTLP

The BLOCKS= data set has two required variables:

ROW
> specifies the constraint (row) names of the problem. The values should be a subset of the row names in the DATA= data set for the current problem.

BLOCK
> specifies the numeric block identifier for each constraint in the problem. A missing observation or missing value indicates a master (linking) constraint that does not appear in any block. Listing the linking constraints is optional. The block identifiers must start from 0 and be consecutive.

See the section "Solving a MILP with DECOMP and PROC OPTMILP" on page 706 for an example of using this BLOCKS= data set with PROC OPTMILP.

The `.block` Constraint Suffix in PROC OPTMODEL

The `.block` constraint suffix specifies the numeric block identifier for each constraint in the problem. The block identifiers do not need to start from 0, nor do they need to be consecutive. Master (linking) constraints can be identified by using a missing value. Listing the linking constraints is optional.

See the section "Solving a MILP with DECOMP and PROC OPTMODEL" on page 704 for an example of using the `.block` constraint suffix with PROC OPTMODEL.

Decomposition Algorithm

The decomposition algorithm for LPs is based on the original Dantzig-Wolfe method (Dantzig and Wolfe 1960). Embedding this method in the context of a branch-and-bound algorithm for MILPs is described in Barnhart et al. (1998) and is often referred to as *branch-and-price*. The design of a framework that allows for building a generic branch-and-price solver that requires only the original (compact) formulation and the constraint partition was first proposed independently by Ralphs and Galati (2006) and Vanderbeck and Savelsbergh (2006). This method is also commonly referred to as *column generation*, although the algorithm implemented here is only one specific variant of this wider class of algorithms.

The algorithm setup starts by forming various components that are used iteratively during the solver process. These components include the master problem (controlled by options in the DECOMP_MASTER statement), one subproblem for each block (controlled by options in the DECOMP_SUBPROB statement) and, for MILPs, the integer version of the master problem (controlled by options in the DECOMP_MASTER_IP statement).

The master problem is a linear program that is defined over a potentially large number of variables that represent the weights of a convex combination. The points in the convex combination satisfy the constraints that are defined in the subproblem. The master constraints of the original problem are enforced in this reformulated space. In this sense, the decomposition algorithm takes the intersection of two polyhedra: one defined by original master constraints and one defined by the subproblem constraints. Since the set of variables needed to define the intersection of the polyhedra can be large, the algorithm works on a restricted subset and generates only those variables (columns) that have good potential with respect to feasibility and optimality. This generation is done by using the dual information that is obtained by solving the master problem to *price out* new variables. These new variables are generated by solving the subproblems over the appropriate cost vector (the reduced cost in the original space). This generation is similar to the revised simplex method, except that the variable space is exponentially large and therefore is generated implicitly by solving an optimization problem. This idea of generating variables as needed is the reason why this method is often referred to as *column generation*.

Similar to the two-phase simplex algorithm, the algorithm first introduces slack variables and solves a phase I problem to find a feasible solution. After the algorithm finds a feasible solution, it switches to a phase II problem to search for an optimal solution. The process of solving the master to generate pricing information and then solving one or more subproblems to generate candidate variables is repeated until there are no longer any improving variables and the method has converged.

For MILPs, this process is then used as a bounding method in a branch-and-bound algorithm, as described in the section "Branch-and-Bound Algorithm" on page 631. The strength of the subproblem polyhedron is one of the key reasons why decomposition can often solve problems that the standard branch-and-cut algorithm cannot solve in a reasonable amount of time. Since the points used in the convex combination are solutions (extreme points) of the subproblem (typically a MILP itself), then the bounds obtained can often be much stronger than the bounds obtained from standard branch-and-bound methods that are based on the LP relaxation. The subproblem polyhedron intersected with the continuous master polyhedron can provide a very good approximation of the true convex hull of the original integer program.

For more information about the algorithm process flow and the framework design, see Galati (2009).

Parallel Processing

At each iteration of the decomposition method, the subproblem is solved to minimize the reduced cost that is derived from the dual information that solving the master problem provides. As discussed in the section "Overview: Decomposition Algorithm" on page 702, the subproblem often has a block-diagonal structure that enables the solver to process each block independently.

You can run the decomposition algorithm in either a single-machine or a distributed computing environment. In single-machine mode, the computation is executed by multiple threads on a single computer. You can specify options for parallel execution in the PERFORMANCE statement, which is documented in the section "PERFORMANCE Statement" on page 23 of Chapter 4, "Shared Concepts and Topics." You can control the number of threads that are used by specifying the NTHREADS= option in the PERFORMANCE statement.

In distributed mode, the computation is executed in a distributed computing environment. You can control the number of grid nodes (machines) that are used by specifying the NODES= option in the PERFORMANCE statement. The decomposition algorithm supports only the deterministic mode of the PARALLELMODE= option in the PERFORMANCE statement. The default mode of operation is single-machine mode, in which the number of concurrent threads is based on the number of CPUs (cores) on the machine (subject to any configuration limitations of the system).

The specified number of threads is used at each iteration to determine the number of blocks to be processed simultaneously. This same value is also used to determine the number of threads to be used for solving the master (continuous and integer) problem if the chosen solver method supports multithreading. To avoid contention, the number of threads that are allocated to each subproblem solve is 1 (unless the number of blocks to process is less than the number of threads).

In addition, in each subcomponent statement you can use the NTHREADS= option to specify the number of threads to use for that solver.

NOTE: Distributed mode requires SAS High-Performance Optimization.

Special Case: Identical Blocks and Ryan-Foster Branching

In the special case of a set partitioning master problem and identical blocks, the underlying algorithm is automatically adjusted to reduce symmetry and improve overall performance. Identical blocks are subproblems (see the section "Overview: Decomposition Algorithm" on page 702) that have equivalent feasible regions (and optima) when they are projected. Algebraically, this means that

$$
\begin{aligned}
\mathbf{A}^1 &= \mathbf{A}^2 = \ldots = \mathbf{A}^\kappa \\
\mathbf{D}^1 &= \mathbf{D}^2 = \ldots = \mathbf{D}^\kappa \\
\mathbf{c}^1 &= \mathbf{c}^2 = \ldots = \mathbf{c}^\kappa \\
\mathbf{b}^1 &= \mathbf{b}^2 = \ldots = \mathbf{b}^\kappa \\
\overline{\mathbf{x}}^1 &= \overline{\mathbf{x}}^2 = \ldots = \overline{\mathbf{x}}^\kappa \\
\underline{\mathbf{x}}^1 &= \underline{\mathbf{x}}^2 = \ldots = \underline{\mathbf{x}}^\kappa
\end{aligned}
$$

A *set partitioning* problem is a specific type of integer programming model in which each constraint represents choosing exactly one member of a set. These constraints are often referred to as *assignment constraints*. The linear relaxation of a set partitioning problem enables an algorithm to choose fractional parts of several members of some set such that they sum to 1. Algebraically, this means $\mathbf{Ax} = \mathbf{1}$, where all the coefficients in $\mathbf{A}$ are 0 or 1.

The performance of algorithms that use a branch-and-bound method can suffer when the formulation contains substructures that are symmetric. In this context, *symmetric* means that an assignment of solutions can be arbitrarily permuted for some component without affecting the optimality of that solution. For example, if

$$x_{11} = 1 \quad x_{12} = 0 \quad x_{21} = 0 \quad x_{22} = 1$$

and

$$x_{11} = 0 \quad x_{12} = 1 \quad x_{21} = 1 \quad x_{22} = 0$$

are both optimal, then these solutions, x_{ij}, are considered symmetric on index j. That is, you can interchange $j = 1$ and $j = 2$ without affecting the optimality of the solution. The presence of identical blocks in a

mathematical program is an obvious case in which symmetry can hurt performance. In order to overcome this handicap, the decomposition algorithm aggregates the identical blocks into one block when it forms the Dantzig-Wolfe master problem. If the Dantzig-Wolfe master problem is a set partitioning model, the algorithm uses a specialized branching rule known as *Ryan-Foster branching*. If the original master model (after aggregation) is equivalent to the identity matrix, this guarantees that the Dantzig-Wolfe master problem is of the appropriate form. For more information about the aggregate formulation and Ryan-Foster branching, see Barnhart et al. (1998).

Suppose you want to solve the following problem:

$$
\begin{array}{rlcccccccccl}
\max & x_{11} & + & 2x_{21} & + & x_{31} & + & x_{12} & + & 2x_{22} & + & x_{32} \\
\text{subject to} & x_{11} & & & & & & x_{12} & & & & & = & 1 & \text{(m1)} \\
& & & x_{21} & & & & & & x_{22} & & & = & 1 & \text{(m2)} \\
& 5x_{11} & + & 7x_{21} & + & 4x_{31} & & & & & & & \leq & 11 & \text{(s1)} \\
& & & & & & & 5x_{12} & + & 7x_{22} & + & 4x_{32} & \leq & 11 & \text{(s2)} \\
\end{array}
$$
$$x_{ij} \in \{0,1\} \ \ i \in \{1,\ldots,3\}, \ j \in \{1,\ldots,2\}$$

If constraints m1 and m2 are removed, then the remaining constraints s1 and s2 decompose into two independent and identical subproblems. In addition, constraints m1 and m2 form a set partitioning master problem.

The following statements use the OPTMODEL procedure and the decomposition algorithm to solve the preceding problem:

```
proc optmodel;
   var x{i in 1..3, j in 1..2} binary;

   max f =    x[1,1] + 2*x[2,1] +   x[3,1] +
              x[1,2] + 2*x[2,2] +   x[3,2];

   con m1:    x[1,1] +   x[1,2]                   =  1;
   con m2:    x[2,1] +   x[2,2]                   =  1;
   con s1:  5*x[1,1] + 7*x[2,1] + 4*x[3,1] <= 11;
   con s2:  5*x[1,2] + 7*x[2,2] + 4*x[3,2] <= 11;

   s1.block = 0;
   s2.block = 1;

   solve with milp / presolver=none decomp=(logfreq=1);
   print x;
quit;
```

Here, the PRESOLVER=NONE option is used again, because otherwise the presolver solves this small instance without invoking any solver. The solution summary and optimal solution are displayed in Figure 15.3.

Figure 15.3 Solution Summary and Optimal Solution

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	f
Solution Status	Optimal
Objective Value	4
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	0
Bound Infeasibility	0
Integer Infeasibility	0
Best Bound	4
Nodes	1
Iterations	2
Presolve Time	0.00
Solution Time	0.01

	x	
	1	2
1	0	1
2	1	0
3	1	1

The iteration log, which displays the problem statistics, the progress of the solution, and the optimal objective value, is shown in Figure 15.4.

Figure 15.4 Log

```
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 6 variables (0 free, 0 fixed).
NOTE: The problem has 6 binary and 0 integer variables.
NOTE: The problem has 4 linear constraints (2 LE, 2 EQ, 0 GE, 0 range).
NOTE: The problem has 10 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value NONE is applied.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: All blocks are identical and the master model is set partitioning.
NOTE: The Decomposition algorithm is using an aggregate formulation and Ryan-Foster branching.
NOTE: The problem has a decomposable structure with 2 blocks. The largest block covers 25.00%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 6 (100.00%) variables and 2 (50.00%) constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter          Best         Master         Best        LP         IP   CPU Real
                    Bound      Objective       Integer      Gap        Gap  Time Time
NOTE: Starting phase 1.
         1        0.0000        0.0000            .        0.00%        .     0    0
NOTE: Starting phase 2.
         .        6.0000        5.0000        5.0000     16.67%     16.67%    0    0
         2        6.0000        5.0000        5.0000     16.67%     16.67%    0    0
         3        5.0000        5.0000        5.0000      0.00%      0.00%    0    0
      Node   Active   Sols          Best           Best        Gap     CPU    Real
                                   Integer         Bound               Time    Time
         0        0      1        5.0000         5.0000      0.00%       0       0
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 0.01 seconds.
NOTE: Optimal.
NOTE: Objective = 5.
```

The decomposition solver recognizes that the original master model is of the appropriate form and that each block is identical. It formulates the aggregate master and uses Ryan-Foster branching to solve the model.

In the presence of identical blocks, under certain circumstances, the aggregate formulation can also be used with a set covering master formulation. A *set covering* problem is an integer programming model in which each constraint represents choosing at least one member of a set. Algebraically, this means $\mathbf{Ax} \geq \mathbf{1}$, where all the coefficients in $\mathbf{A}$ are 0 or 1. Aggregate formulation and Ryan-Foster branching can be used if there exists an optimal solution, $\mathbf{x}^*$, that is binding at equality ($\mathbf{Ax}^* = \mathbf{1}$). If you can guarantee such a condition, you can greatly improve performance by explicitly using VARSEL=RYANFOSTER as a MILP main solver option. The decomposition algorithm usually performs better when it uses a set covering formulation as opposed to a set partitioning formulation, because it is usually easier to find integer feasible solutions. If the models are equivalent, using the set covering formulation is recommended. For two examples, see Example 15.6, which shows the bin packing problem, and Example 15.8, which shows the vehicle routing problem.

Similarly, a *set packing* problem is an integer programming model in which each constraint represents choosing at most one member of a set. Algebraically, this means $\mathbf{Ax} \leq \mathbf{1}$, where all the coefficients in $\mathbf{A}$ are 0 or 1. Aggregate formulation and Ryan-Foster branching can be used if there exists an optimal solution, $\mathbf{x}^*$, that is binding at equality ($\mathbf{Ax}^* = \mathbf{1}$). In this case, using VARSEL=RYANFOSTER can improve performance. Alternatively, you can transform any set packing model into a set partitioning model by introducing a zero-cost slack variable for each packing constraint. See Example 15.11, which shows an application that optimizes a kidney donor exchange.

The decomposition algorithm automatically searches for identical blocks and the appropriate set partitioning master formulation. If it finds this structure, the algorithm automatically generates the aggregate formulation and uses Ryan-Foster branching. The aggregate model needs to process only one block at each subproblem iteration. Therefore, parallel processing (in which multiple blocks are processed simultaneously), as described in the section "Parallel Processing" on page 727, cannot improve performance. For this reason, when the decomposition algorithm runs in distributed mode, it does not create the aggregate formulation, nor does it use Ryan-Foster branching, even if the blocks are found to be identical.

Log for the Decomposition Algorithm

The following subsections describe what to expect in the SAS log when you run the decomposition algorithm.

Setup Information in the SAS Log

In the setup phase of the algorithm, information about the method you choose and the structure of the model is written to the SAS log. One of the most important pieces of information displayed in the log is the number of disjoint blocks and the coverage of those blocks with respect to both variables and constraints in the original presolved model. As explained in the section "Overview: Decomposition Algorithm" on page 702, the decomposition algorithm usually performs better than standard approaches only if the subproblems cover a significant amount of the original problem. However, this is not always a straightforward indicator for MILPs, because the strength of the subproblem with respect to integrality is not always proportional to the size of the system.

After the structural information is written to the log, the algorithm begins and the iteration log is displayed.

Iteration Log for LPs

When the decomposition algorithm solves LPs, the iteration log shows the progress of convergence in finding the appropriate set of columns in the reformulated space.

The following information is written to the iteration log:

Iter	indicates the iteration number.
Best Bound	indicates the best dual bound found so far.
Master Objective	indicates the current amount of infeasibility in phase I and the primal objective value of the current solution in phase II.
Gap	indicates the relative difference between the master objective and the best known dual bound. This indicates how close the algorithm is to convergence. If the relative gap is greater than 1000%, then the absolute gap is written.

CPU Time   indicates the CPU time elapsed (in seconds).

Real Time   indicates the real time elapsed (in seconds).

Entries are made in the log at a frequency that is specified in the LOGFREQ= option. If LOGFREQ=0, then the iteration log is disabled. If the LOGFREQ= value is positive, then an entry is made in the log at the first iteration, at the last iteration, and at intervals that are specified by the LOGFREQ= value. An entry is also made each time an improved bound is found.

The behavior of objective values in the iteration log depends on both the current phase and on which solver you choose. In phase I, the master formulation has an artificial objective value that decreases to 0 when a feasible solution is found. In phase II, the decomposition algorithm maintains a primal feasible solution, so a minimization problem has decreasing objective values in the iteration log.

When you specify LOGLEVEL=MODERATE or LOGLEVEL=AGGRESSIVE in the DECOMP statement, information about the subproblem solves is written before each iteration line.

Iteration Log for MILPs

When the decomposition algorithm solves MILPs, the iteration log shows the progress of convergence in finding the appropriate set of columns in the reformulated space, in addition to the global convergence of the branch-and-bound algorithm for finding an optimal integer solution.

You can control the amount of information at each node by using the LOGLEVEL= option in the DECOMP statement. By default, the continuous iteration log for the root node is written at the interval specified in the LOGFREQ= option in the DECOMP statement. Then the branch-and-bound node log is written at the interval specified in the LOGFREQ= main solver option.

When the algorithm solves MILPs, the continuous iteration log is similar to the iteration log described in the section "Iteration Log for LPs" on page 732 except that information about integer-feasible solutions is also displayed. The following information is printed in the continuous iteration log when the algorithm solves MILPs:

Iter   indicates the iteration number.

Best Bound   indicates the best dual bound found so far.

Master Objective   indicates the current amount of infeasibility in phase I and the primal objective value of the current solution in phase II.

Best Integer   indicates the objective of the best integer-feasible solution found so far.

LP Gap   indicates the relative difference between the master objective and the best known dual bound. This indicates how close the algorithm for this particular node is to convergence. If the relative gap is greater than 1000%, then the absolute gap is displayed.

IP Gap   indicates the relative difference between the best integer and the best known dual bound. This indicates how close the branch-and-bound algorithm is to convergence. If the relative gap is greater than 1000%, then the absolute gap is displayed.

CPU Time   indicates the CPU time elapsed (in seconds).

Real Time   indicates the real time elapsed (in seconds).

After the root node is complete, the algorithm then moves into the branch-and-bound phase. By default, it displays the branch-and-bound node log and suppresses the continuous iteration log.

The following information is printed in the branch-and-bound node log when the algorithm solves MILPs:

Node   indicates the sequence number of the current node in the search tree.

Active   indicates the current number of active nodes in the branch-and-bound tree.

Sols   indicates the number of feasible solutions found so far.

Best Integer   indicates the objective of the best integer-feasible solution found so far.

Best Bound   indicates the best dual bound found so far.

Gap   indicates the relative difference between the best integer and the best known dual bound. This indicates how close the branch-and-bound algorithm is to convergence. If the relative gap is greater than 1000%, then the absolute gap is displayed.

CPU Time   indicates the CPU time elapsed (in seconds).

Real Time   indicates the real time elapsed (in seconds).

If the LOGLEVEL= option in the DECOMP statement is set to BASIC, MODERATE or AGGRESSIVE, then the continuous iteration log is displayed for each branch-and-bound node at the interval specified in the LOGFREQ= option in the DECOMP statement.

Additional information can be displayed to the log by specifying the LOGLEVEL= option in each of the algorithmic component statements (DECOMP_MASTER, DECOMP_MASTER_IP, and DECOMP_SUBPROB). By default, the individual component log levels are all disabled.

Examples: Decomposition Algorithm

Example 15.1: Multicommodity Flow Problem

This example demonstrates how to use the decomposition algorithm to find a minimum-cost multicommodity flow (MMCF) in a directed network. This type of problem was motivation for the development of the original Dantzig-Wolfe decomposition method (Dantzig and Wolfe 1960).

Let $G = (N, A)$ be a directed graph, and let K be a set of commodities. For each link $(i, j) \in A$ and each commodity k, associate a cost per unit of flow, designated by c_{ij}^k. The demand (or supply) at each node $i \in N$ for commodity k is designated as b_i^k, where $b_i^k \geq 0$ denotes a supply node and $b_i^k < 0$ denotes a demand node. Define decision variables x_{ij}^k that denote the amount of commodity k sent from node i and node j. The amount of total flow, across all commodities, that can be sent across each link is bounded above by u_{ij}.

The problem can be modeled as a linear programming problem as follows:

minimize
$$\sum_{(i,j)\in A}\sum_{k\in K} c_{ij}^k x_{ij}^k$$

subject to

$$\sum_{k\in K} x_{ij}^k \le u_{ij} \qquad (i,j) \in A \qquad \text{(Capacity)}$$

$$\sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = b_i^k \qquad i \in N, \ k \in K \qquad \text{(Balance)}$$

$$x_{ij}^k \ge 0 \qquad (i,j) \in A, \ k \in K$$

In this formulation, The Capacity constraints limit the total flow across all commodities on each arc. The Balance constraints ensure that the flow of commodities leaving each supply node and entering each demand node are balanced.

Consider the directed graph in Figure 15.5 which appears in Ahuja, Magnanti, and Orlin (1993).

Figure 15.5 Example Network with Two Commodities
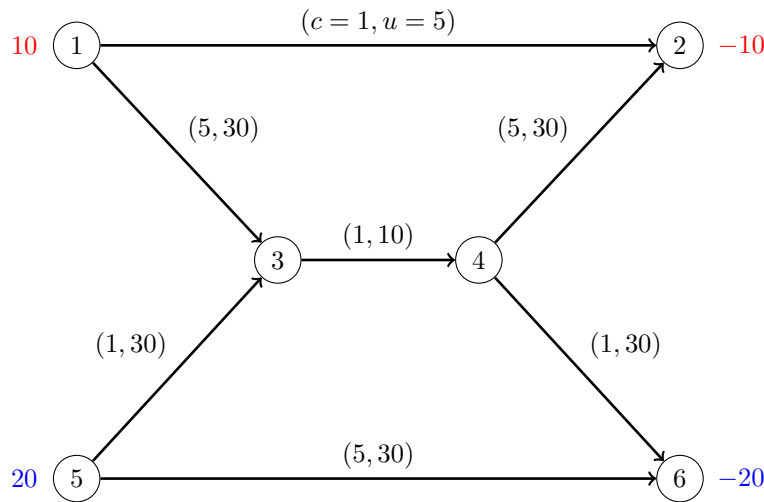
The goal in this example is to minimize the total cost of sending two commodities across the network while satisfying all supplies and demands and respecting arc capacities. If there were no arc capacities linking the two commodities, you could solve a separate minimum-cost network flow problem for each commodity one at a time.

The following data set arc_comm_data provides the cost c_{ij}^k of sending a unit of commodity k along arc (i,j):

```
data arc_comm_data;
   input k i j cost;
   datalines;
1 1 2 1
1 1 3 5
1 5 3 1
1 5 6 5
1 3 4 1
```

```
1 4 2 5
1 4 6 1
2 1 2 1
2 1 3 5
2 5 3 1
2 5 6 5
2 3 4 1
2 4 2 5
2 4 6 1
;
```

Next, the data set arc_data provides the capacity u_{ij} for each arc:

```
data arc_data;
   input i j capacity;
   datalines;
1 2  5
1 3 30
5 3 30
5 6 30
3 4 10
4 2 30
4 6 30
;

data supply_data;
   input k i supply;
   datalines;
1 1  10
1 2 -10
2 5  20
2 6 -20
;
```

The following PROC OPTMODEL statements find the minimum-cost multicommodity flow:

```
proc optmodel;
   set <num,num,num> ARC_COMM;
   num cost {ARC_COMM};
   read data arc_comm_data into ARC_COMM=[i j k] cost;

   set ARCS = setof {<i,j,k> in ARC_COMM} <i,j>;
   set COMMODITIES = setof {<i,j,k> in ARC_COMM} k;
   set NODES = union {<i,j> in ARCS} {i,j};

   num arcCapacity {ARCS};
   read data arc_data into [i j] arcCapacity=capacity;

   num supply {NODES, COMMODITIES} init 0;
   read data supply_data into [i k] supply;

   var Flow {<i,j,k> in ARC_COMM} >= 0;
   min TotalCost =
      sum {<i,j,k> in ARC_COMM} cost[i,j,k] * Flow[i,j,k];
   con Balance {i in NODES, k in COMMODITIES}:
```

```
        sum {<(i),j,(k)> in ARC_COMM} Flow[i,j,k]
          - sum {<j,(i),(k)> in ARC_COMM} Flow[j,i,k] = supply[i,k];
    con Capacity {<i,j> in ARCS}:
        sum {<(i),(j),k> in ARC_COMM} Flow[i,j,k] <= arcCapacity[i,j];
```

Because each Balance constraint involves variables for only one commodity, a decomposition by commodity is a natural choice. In both the OPTLP and OPTMILP procedures, the block identifiers must be consecutive integers starting from 0. In PROC OPTMODEL, the block identifiers only need to be numeric. The following **FOR** loop populates the **.block** constraint suffix with block identifier *k* for commodity *k*:

```
    for{i in NODES, k in COMMODITIES}
        Balance[i,k].block = k;
```

The **.block** constraint suffix for the linking Capacity constraints is left missing, so these constraints become part of the master problem.

The following SOLVE statement uses the DECOMP= option to invoke the decomposition algorithm:

```
    solve with LP / presolver=none decomp subprob=(algorithm=nspure);
    print Flow;
 quit;
```

Here, the PRESOLVER=NONE option is used, because otherwise the presolver solves this small instance without invoking any solver. Because each subproblem is a pure network flow problem, you can use the ALGORITHM=NSPURE option in the SUBPROB= option to request that a network simplex algorithm for pure networks be used instead of the default algorithm, which for linear programming subproblems is primal simplex.

It turns out for this example that if you specify METHOD=NETWORK (instead of the default METHOD=USER) in the DECOMP= option, the network extractor finds the same blocks, one per commodity. To invoke the METHOD=NETWORK option, simply change the SOLVE statement as follows:

```
    solve with LP / presolver=none decomp=(method=network);
```

In this case, the default subproblem solver is NSPURE.

The optimal solution and solution summary are displayed in Output 15.1.1.

Output 15.1.1 Solution Summary and Optimal Solution

The OPTMODEL Procedure

Solution Summary	
Solver	LP
Algorithm	Decomposition
Objective Function	TotalCost
Solution Status	Optimal
Objective Value	150
Primal Infeasibility	0
Dual Infeasibility	0
Bound Infeasibility	0
Iterations	4
Presolve Time	0.00
Solution Time	0.01

[1]	[2]	[3]	Flow
1	2	1	5
1	2	2	0
1	3	1	5
1	3	2	0
3	4	1	5
3	4	2	5
4	2	1	5
4	2	2	0
4	6	1	0
4	6	2	5
5	3	1	0
5	3	2	5
5	6	1	0
5	6	2	15

The optimal solution is shown on the network in Figure 15.6.

Figure 15.6 Optimal Flow on Network with Two Commodities
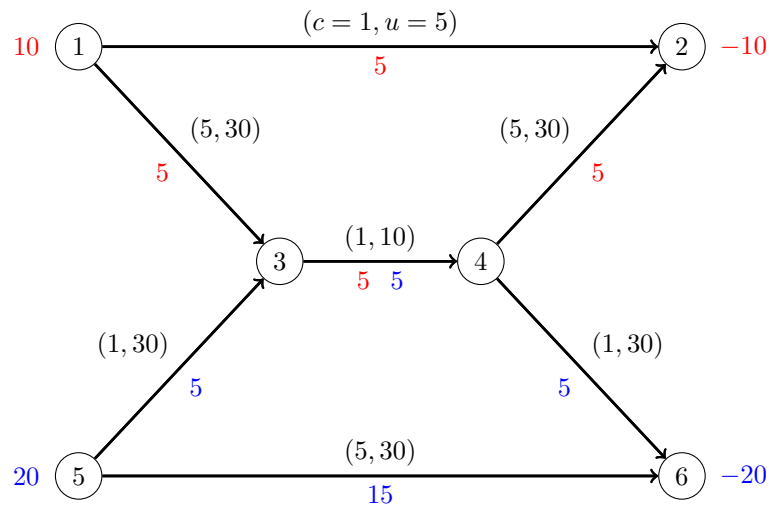
The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.1.2.

Output 15.1.2 Log

```
NOTE: There were 14 observations read from the data set WORK.ARC_COMM_DATA.
NOTE: There were 7 observations read from the data set WORK.ARC_DATA.
NOTE: There were 4 observations read from the data set WORK.SUPPLY_DATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 14 variables (0 free, 0 fixed).
NOTE: The problem has 19 linear constraints (7 LE, 12 EQ, 0 GE, 0 range).
NOTE: The problem has 42 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The LP presolver value NONE is applied.
NOTE: The LP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: The problem has a decomposable structure with 2 blocks. The largest block covers 31.58%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 14 (100.00%) variables and 12 (63.16%) constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter          Best        Master        Gap       CPU        Real
                    Bound     Objective                  Time       Time
NOTE: Starting phase 1.
        1         0.0000       15.0000    1.50e+01       0.0        0.0
        3         0.0000        0.0000       0.00%       0.0        0.0
NOTE: Starting phase 2.
        4       150.0000      150.0000       0.00%       0.0        0.0
NOTE: The Decomposition algorithm used 2 threads.
NOTE: The Decomposition algorithm time is 0.01 seconds.
NOTE: Optimal.
NOTE: Objective = 150.
```

Example 15.2: Generalized Assignment Problem

The generalized assignment problem (GAP) is that of finding a maximum profit assignment from n tasks to m machines such that each task is assigned to precisely one machine subject to capacity restrictions on the machines. With each possible assignment, associate a binary variable x_{ij}, which, if set to 1, indicates that machine i is assigned to task j. For ease of notation, define two index sets $M = \{1, \ldots, m\}$ and

$N = \{1, \ldots, n\}$. A GAP can be formulated as a MILP as follows:

$$\text{maximize} \quad \sum_{i \in M} \sum_{j \in N} p_{ij} x_{ij}$$

$$\text{subject to} \quad \sum_{i \in M} x_{ij} = 1 \qquad j \in N \qquad \text{(Assignment)}$$

$$\sum_{j \in N} w_{ij} x_{ij} \leq b_i \qquad i \in M \qquad \text{(Knapsack)}$$

$$x_{ij} \in \{0, 1\} \qquad i \in M, \ j \in N$$

In this formulation, Assignment constraints ensure that each task is assigned to exactly one machine. Knapsack constraints ensure that for each machine, the capacity restrictions are met.

Consider the following example taken from Koch et al. (2011) with $n = 24$ tasks to be assigned to $m = 8$ machines. The data set profit_data provides the profit for assigning a particular task to a particular machine:

```
%let NumTasks    = 24;
%let NumMachines = 8;

data profit_data;
   input p1-p&NumTasks;
   datalines;
25 23 20 16 19 22 20 16 15 22 15 21 20 23 20 22 19 25 25 24 21 17 23 17
16 19 22 22 19 23 17 24 15 24 18 19 20 24 25 25 19 24 18 21 16 25 15 20
20 18 23 23 23 17 19 16 24 24 17 23 19 22 23 25 23 18 19 24 20 17 23 23
16 16 15 23 15 15 25 22 17 20 19 16 17 17 20 17 17 18 16 18 15 25 22 17
17 23 21 20 24 22 25 17 22 20 16 22 21 23 24 15 22 25 18 19 19 17 22 23
24 21 23 17 21 19 19 17 18 24 15 15 17 18 15 24 19 21 23 24 17 20 16 21
18 21 22 23 22 15 18 15 21 22 15 23 21 25 25 23 20 16 25 17 15 15 18 16
19 24 18 17 21 18 24 25 18 23 21 15 24 23 18 18 23 23 16 20 20 19 25 21
;
```

The data set weight_data provides the amount of resources used by a particular task when assigned to a particular machine:

```
data weight_data;
   input w1-w&NumTasks;
   datalines;
 8 18 22  5 11 11 22 11 17 22 11 20 13 13  7 22 15 22 24  8  8 24 18  8
24 14 11 15 24  8 10 15 19 25  6 13 10 25 19 24 13 12  5 18 10 24  8  5
22 22 21 22 13 16 21  5 25 13 12  9 24  6 22 24 11 21 11 14 12 10 20  6
13  8 19 12 19 18 10 21  5  9 11  9 22  8 12 13  9 25 19 24 22  6 19 14
25 16 13  5 11  8  7  8 25 20 24 20 11  6 10 10  6 22 10 10 13 21  5 19
19 19  5 11 22 24 18 11  6 13 24 24 22  6 22  5 14  6 16 11  6  8 18 10
24 10  9 10  6 15  7 13 20  8  7  9 24  9 21  9 11 19 10  5 23 20  5 21
 6  9  9  5 12 10 16 15 19 18 20 18 16 21 11 12 22 16 21 25  7 14 16 10
;
```

Finally, the data set capacity_data provides the resource capacity for each machine:

```
data capacity_data;
   input b @@;
   datalines;
36 35 38 34 32 34 31 34
;
```

The following PROC OPTMODEL statements read in the data and define the necessary sets and parameters:

```
proc optmodel;
   /* declare index sets */
   set TASKS    = 1..&NumTasks;
   set MACHINES = 1..&NumMachines;

   /* declare parameters */
   num profit   {MACHINES, TASKS};
   num weight   {MACHINES, TASKS};
   num capacity {MACHINES};

   /* read data sets to populate data */
   read data profit_data   into [i=_n_] {j in TASKS} <profit[i,j]=col('p'||j)>;
   read data weight_data    into [i=_n_] {j in TASKS} <weight[i,j]=col('w'||j)>;
   read data capacity_data into [_n_] capacity=b;
```

The following statements declare the optimization model:

```
   /* declare decision variables */
   var Assign {MACHINES, TASKS} binary;

   /* declare objective */
   max TotalProfit =
      sum {i in MACHINES, j in TASKS} profit[i,j] * Assign[i,j];

   /* declare constraints */
   con Assignment {j in TASKS}:
      sum {i in MACHINES} Assign[i,j] = 1;

   con Knapsack {i in MACHINES}:
      sum {j in TASKS} weight[i,j] * Assign[i,j] <= capacity[i];
```

The following statements use two different decompositions to solve the problem. The first decomposition defines each Assignment constraint as a block and uses the pure network simplex solver for the subproblem. The second decomposition defines each Knapsack constraint as a block and uses the MILP solver for the subproblem.

```
   /* each Assignment constraint defines a block */
   for{j in TASKS}
      Assignment[j].block = j;

   solve with milp / logfreq=1000
      decomp        =()
      decomp_subprob=(algorithm=nspure);
```

```
    /* each Knapsack constraint defines a block */
    for{j in TASKS}
        Assignment[j].block = .;
    for{i in MACHINES}
        Knapsack[i].block = i;


    solve with milp / decomp;
quit;
```

The solution summaries are displayed in Output 15.2.1.

Output 15.2.1 Solution Summaries

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalProfit
Solution Status	Optimal within Relative Gap
Objective Value	563
Relative Gap	0.0000925018
Absolute Gap	0.0520833333
Primal Infeasibility	6.661338E-16
Bound Infeasibility	2.220446E-16
Integer Infeasibility	6.661338E-16
Best Bound	563.05208333
Nodes	1763
Iterations	1802
Presolve Time	0.00
Solution Time	4.46

The iteration log for both decompositions is shown in Output 15.2.2. This example is interesting because it shows the tradeoff between the strength of the relaxation and the difficulty of its resolution. In the first decomposition, the subproblems are totally unimodular and can be solved trivially. Consequently, each iteration of the decomposition algorithm is very fast. However, the bound obtained is as weak as the bound found in direct methods (the LP bound). The weaker bound leads to the need to enumerate more nodes overall. Alternatively, in the second decomposition, the subproblem is the knapsack problem, which is solved using MILP. In this case, the bound is much tighter and the problem solves in very few nodes. The tradeoff, of course, is that each iteration takes longer because solving the knapsack problem is not trivial. Another interesting aspect of this problem is that the subproblem coverage in the second decomposition is much smaller than that of the first decomposition. However, when dealing with MILP, it is not always the size of the coverage that determines the overall effectiveness of a particular choice of decomposition.

Output 15.2.2 Log

```
NOTE: There were 8 observations read from the data set WORK.PROFIT_DATA.
NOTE: There were 8 observations read from the data set WORK.WEIGHT_DATA.
NOTE: There were 8 observations read from the data set WORK.CAPACITY_DATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 192 variables (0 free, 0 fixed).
NOTE: The problem has 192 binary and 0 integer variables.
NOTE: The problem has 32 linear constraints (8 LE, 24 EQ, 0 GE, 0 range).
NOTE: The problem has 384 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 0 variables and 0 constraints.
NOTE: The MILP presolver removed 0 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 192 variables, 32 constraints, and 384 constraint coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: The subproblem solver chosen is an LP solver but at least one block has integer variables.
NOTE: The problem has a decomposable structure with 24 blocks. The largest block covers 3.13%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 192 (100.00%) variables and 24 (75.00%) constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter          Best        Master         Best        LP      IP  CPU Real
                    Bound     Objective      Integer       Gap     Gap Time Time
NOTE: Starting phase 1.
         1        0.0000        8.9248           .  8.92e+00        .   0    0
         4        0.0000        0.0000           .     0.00%        .   0    0
NOTE: Starting phase 2.
         5      574.0000      561.1587           .     2.24%        .   0    0
         6      568.8833      568.5610           .     0.06%        .   0    0
         8      568.6464      568.6464     560.0000     0.00%    1.52%   0    0
NOTE: Starting branch and bound.
        Node  Active    Sols          Best          Best       Gap     CPU    Real
                                   Integer         Bound              Time    Time
           0       1       1     560.0000      568.6464     1.52%       0       0
           5       7       2     563.0000      568.4782     0.96%       0       0
        1000     432       2     563.0000      564.6212     0.29%       2       2
        1762       0       2     563.0000      563.0521     0.01%       4       4
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 4.46 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 563.
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 0 variables and 0 constraints.
NOTE: The MILP presolver removed 0 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 192 variables, 32 constraints, and 384 constraint coefficients.
```

Output 15.2.2 *continued*

```
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: The problem has a decomposable structure with 8 blocks. The largest block covers 3.13% of
      the constraints in the problem.
NOTE: The decomposition subproblems cover 192 (100.00%) variables and 8 (25.00%) constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
```

Iter	Best Bound	Master Objective	Best Integer	LP Gap	IP Gap	CPU Time	Real Time
NOTE: Starting phase 1.							
1	0.0000	10.0000	.	1.00e+01	.	0	0
8	0.0000	0.0000	.	0.00%	.	0	0
NOTE: Starting phase 2.							
11	717.5556	540.0000	540.0000	24.74%	24.74%	0	0
13	670.3333	548.0000	548.0000	18.25%	18.25%	0	0
14	627.9557	548.0000	548.0000	12.73%	12.73%	0	0
16	592.2500	549.8750	548.0000	7.15%	7.47%	0	0
19	592.2500	558.0000	558.0000	5.78%	5.78%	0	0
.	592.2500	558.0000	558.0000	5.78%	5.78%	0	0
20	577.6667	558.0000	558.0000	3.40%	3.40%	0	0
23	574.6667	560.6667	560.0000	2.44%	2.55%	0	0
24	574.6667	563.0000	563.0000	2.03%	2.03%	0	0
25	569.5000	563.5000	563.0000	1.05%	1.14%	0	0
26	566.1905	563.7143	563.0000	0.44%	0.56%	0	0
28	564.5000	564.0000	563.0000	0.09%	0.27%	0	0
29	564.0000	564.0000	563.0000	0.00%	0.18%	0	0

```
NOTE: Starting branch and bound.
```

Node	Active	Sols	Best Integer	Best Bound	Gap	CPU Time	Real Time
0	1	8	563.0000	564.0000	0.18%	0	0
4	0	8	563.0000	563.0000	0.00%	0	0

```
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 0.15 seconds.
NOTE: Optimal.
NOTE: Objective = 563.
```

Example 15.3: Block-Diagonal Structure and METHOD=CONCOMP in Single-Machine Mode

This example demonstrates how you can use the METHOD=CONCOMP option in the DECOMP statement to execute the decomposition algorithm in single-machine mode.

Consider a mixed integer linear program that is defined by the MPS data set mpsdata. In this case, the structure of the model is unknown and only the MPS data set is provided to you.

The following PROC OPTMILP statements solve the problem by using standard methods:

```
proc optmilp
    data = mpsdata;
run;
```

The solution summary is shown in .

Output 15.3.1 Solution Summary

The OPTMILP Procedure

Solution Summary	
Solver	MILP
Algorithm	Branch and Cut
Objective Function	R0001298
Solution Status	Optimal
Objective Value	120
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	2.842171E-14
Bound Infeasibility	1.421085E-14
Integer Infeasibility	1.776357E-15
Best Bound	120
Nodes	1
Iterations	53384
Presolve Time	0.03
Solution Time	3.00

The iteration log, which contains the problem statistics and the progress of the solution, is shown in Output 15.3.2.

Output 15.3.2 Log

```
NOTE: The problem MPSDATA has 388 variables (36 binary, 0 integer, 1 free, 0 fixed).
NOTE: The problem has 1297 constraints (630 LE, 37 EQ, 630 GE, 0 range).
NOTE: The problem has 4204 constraint coefficients.
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 37 variables and 37 constraints.
NOTE: The MILP presolver removed 424 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 351 variables, 1260 constraints, and 3780 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The problem has a decomposable structure with 4 blocks. The largest block covers 25.08%
      of the constraints in the problem. The DECOMP option with METHOD=CONCOMP is recommended
      for solving problems with this structure.
         Node  Active    Sols    BestInteger      BestBound      Gap    Time
            0       1       1    231.0000000     91.4479396  152.60%       0
            0       1       3    120.0000000    120.0000000    0.00%       2
            0       0       3    120.0000000    120.0000000    0.00%       2
NOTE: The MILP solver added 6 cuts with 33 cut coefficients at the root.
NOTE: Optimal.
NOTE: Objective = 120.
```

A note in the log suggests that you can use the decomposition algorithm because of the structure of the problem. The following PROC OPTMILP statements use the METHOD=CONCOMP option in the DECOMP statement in single-machine mode. The PERFORMANCE statement specifies the number of threads to be used.

```
proc optmilp
   data       = mpsdata;
   decomp
     loglevel = 2
     method   = concomp;
   performance
     nthreads = 4;
run;
```

The performance information and solution summary are displayed in Output 15.3.3.

Output 15.3.3 Performance Information and Solution Summary

The OPTMILP Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	R0001298
Solution Status	Optimal
Objective Value	120
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	9.381385E-14
Bound Infeasibility	4.263256E-14
Integer Infeasibility	0
Best Bound	120
Nodes	1
Iterations	1
Presolve Time	0.04
Solution Time	1.14

The iteration log, which contains the problem statistics and the progress of the solution, is shown in Output 15.3.4. When you specify NTHREADS=4 in the PERFORMANCE statement in single-machine mode, each block is processed simultaneously on each of four threads.

Output 15.3.4 Log

```
NOTE: The OPTMILP procedure is executing in single-machine mode.
NOTE: The problem MPSDATA has 388 variables (36 binary, 0 integer, 1 free, 0 fixed).
NOTE: The problem has 1297 constraints (630 LE, 37 EQ, 630 GE, 0 range).
NOTE: The problem has 4204 constraint coefficients.
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 37 variables and 37 constraints.
NOTE: The MILP presolver removed 424 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 351 variables, 1260 constraints, and 3780 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The DECOMP method value CONCOMP is applied.
NOTE: The problem has a decomposable structure with 4 blocks. The largest block covers 25.08%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 351 (100.00%) variables and 1260 (100.00%)
      constraints.
NOTE: Block 1 has 88 (25.07%) variables and 316 (25.08%) constraints.
NOTE: Block 2 has 88 (25.07%) variables and 316 (25.08%) constraints.
NOTE: Block 3 has 88 (25.07%) variables and 316 (25.08%) constraints.
NOTE: Block 4 has 87 (24.79%) variables and 312 (24.76%) constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
NOTE: ------------------------------------------------------------------------------------------
NOTE: Starting to process node 0.
NOTE: ------------------------------------------------------------------------------------------
NOTE: Using a starting solution with objective value 231 to provide initial columns.
NOTE: The initial column pool using the starting solution contains 4 columns.
NOTE: The subproblem solver for 4 blocks at iteration 0 is starting.
NOTE: The subproblem solver for 4 blocks used 1.09 (cpu: 3.28) seconds.
NOTE: The initial column pool after generating initial variables contains 8 columns.
NOTE: The Decomposition algorithm stopped on the integer RELOBJGAP= option.
NOTE: The number of active nodes is 0.
NOTE: The objective value of the best integer feasible solution is 120.0000 and the best bound
      is 120.0000.
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 1.09 seconds.
NOTE: Optimal.
NOTE: Objective = 120.
```

In this case, the solver finds that after presolve, the constraint matrix decomposes into block-diagonal form. That is, all the constraints are covered by subproblem blocks, leaving the set of master constraints empty. Because there are no coupling constraints, the problem decomposes into four completely independent problems. If you specify LOGLEVEL=2 in the DECOMP statement, the log displays the size of each block. The blocks in this case are nicely balanced, allowing parallel execution to be efficient.

Example 15.4: Block-Diagonal Structure and METHOD=CONCOMP in Distributed Mode

This example demonstrates how you can use the METHOD=CONCOMP option in the DECOMP statement to execute the decomposition algorithm in distributed mode.

As in Example 15.3, consider a mixed integer linear program that is defined by the MPS data set mpsdata. In this case, the structure of the model is unknown and only the MPS data set is provided to you.

The following PROC OPTMILP statements use the METHOD=CONCOMP option in distributed mode. The PERFORMANCE statement specifies the numbers of threads and nodes to be used.

```
proc optmilp
   data       = mpsdata;
   decomp
     loglevel = 2
     method   = concomp;
   performance
     details
     nthreads = 1
     nodes    = 4;
run;
```

The performance information is displayed in Output 15.4.1.

Output 15.4.1 Performance Information

Performance Information	
Host Node	<< your grid host >>
Execution Mode	Distributed
Number of Compute Nodes	4
Number of Threads per Node	1

The solution summary is displayed in Output 15.4.2.

Output 15.4.2 Solution Summary

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	R0001298
Solution Status	Optimal
Objective Value	120
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	9.492407E-14
Bound Infeasibility	1.998401E-14
Integer Infeasibility	0
Best Bound	120
Nodes	1
Iterations	1
Presolve Time	0.05
Solution Time	0.76

The iteration log, which contains the problem statistics and the progress of the solution, is shown in Output 15.4.3. When you specify NODES=4 and NTHREADS=1 in the PERFORMANCE statement in distributed mode, each block is processed simultaneously on each of four grid nodes.

Output 15.4.3 Log

```
NOTE: The problem MPSDATA has 388 variables (36 binary, 0 integer, 1 free, 0 fixed).
NOTE: The problem has 1297 constraints (630 LE, 37 EQ, 630 GE, 0 range).
NOTE: The problem has 4204 constraint coefficients.
NOTE: The OPTMILP procedure is executing in the distributed computing environment with 4 worker
      nodes.
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 37 variables and 37 constraints.
NOTE: The MILP presolver removed 424 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 351 variables, 1260 constraints, and 3780 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The DECOMP method value CONCOMP is applied.
NOTE: The problem has a decomposable structure with 4 blocks. The largest block covers 25.08%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 351 (100.00%) variables and 1260 (100.00%)
      constraints.
NOTE: Block 1 has 88 (25.07%) variables and 316 (25.08%) constraints.
NOTE: Block 2 has 88 (25.07%) variables and 316 (25.08%) constraints.
NOTE: Block 3 has 88 (25.07%) variables and 316 (25.08%) constraints.
NOTE: Block 4 has 87 (24.79%) variables and 312 (24.76%) constraints.
NOTE: ----------------------------------------------------------------
NOTE: Starting to process node 0.
NOTE: ----------------------------------------------------------------
NOTE: Using a starting solution with objective value 231 to provide initial columns.
NOTE: The initial column pool using the starting solution contains 4 columns.
NOTE: The subproblem solver for 4 blocks at iteration 0 is starting.
NOTE: The subproblem solver for 4 blocks used 0.68 (cpu: 0.00) seconds.
NOTE: The initial column pool after generating initial variables contains 8 columns.
NOTE: The Decomposition algorithm stopped on the integer RELOBJGAP= option.
NOTE: The number of active nodes is 0.
NOTE: The objective value of the best integer feasible solution is 120.0000 and the best bound
      is 120.0000.
NOTE: The Decomposition algorithm time is 0.68 seconds.
NOTE: Optimal.
NOTE: Objective = 120.
NOTE: The data set WORK.PERFINFO has 4 observations and 3 variables.
```

Example 15.5: Block-Angular Structure and METHOD=AUTO

This example demonstrates how you can use the METHOD=AUTO option in the DECOMP statement to execute the decomposition algorithm.

As in Example 15.3, consider a mixed integer linear program that is defined by the MPS data set mpsdata. In this case, the structure of the model is unknown and only the MPS data set is provided to you.

The following PROC OPTMILP statements attempt to solve the problem by using standard methods and a 60-second time limit:

```
proc optmilp
   maxtime = 60
   data    = mpsdata;
run;
```

The solution summary is shown in Output 15.5.1.

Output 15.5.1 Solution Summary

The OPTMILP Procedure

Solution Summary	
Solver	MILP
Algorithm	Branch and Cut
Objective Function	Total_Profit
Solution Status	Time Limit Reached
Best Bound	7291.4925915
Nodes	1
Iterations	38627
Presolve Time	0.45
Solution Time	60.01

The iteration log, which contains the problem statistics and the progress of the solution, is shown in Output 15.5.2.

Output 15.5.2 Log

```
NOTE: The problem MPSDATA has 52638 variables (16038 binary, 0 integer, 0 free, 0 fixed).
NOTE: The problem has 3949 constraints (3339 LE, 0 EQ, 610 GE, 0 range).
NOTE: The problem has 148866 constraint coefficients.
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 0 variables and 367 constraints.
NOTE: The MILP presolver removed 6606 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 52638 variables, 3582 constraints, and 142260 constraint
      coefficients.
NOTE: The MILP solver is called.
         Node   Active    Sols    BestInteger      BestBound      Gap    Time
            0        1       0              .      7342.1209242      .       0
            0        1       0              .      7339.5605463      .       7
            0        1       0              .      7329.6821306      .      14
            0        1       0              .      7321.1115391      .      22
            0        1       0              .      7315.4737113      .      29
            0        1       0              .      7310.5357458      .      38
            0        1       0              .      7306.4611529      .      45
            0        1       0              .      7300.6129841      .      53
            0        1       0              .      7295.3817577      .      55
            0        1       0              .      7291.4925915      .      58
NOTE: The MILP solver added 4944 cuts with 118150 cut coefficients at the root.
NOTE: CPU time limit reached.
NOTE: No integer solutions found.
```

Standard MILP techniques struggle to find a feasible solution within the specified time limit. The default decomposition method (METHOD=AUTO) attempts to find a block-angular structure by using the matrix-stretching techniques that are described in Grcar (1990) and Aykanat, Pinar, and Çatalyürek (2004).

```
proc optmilp
   data    = mpsdata;
   decomp
     method = auto;
run;
```

The solution summary is displayed in Output 15.5.3.

Output 15.5.3 Solution Summary

The OPTMILP Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	Total_Profit
Solution Status	Optimal within Relative Gap
Objective Value	6972.3309347
Relative Gap	2.884077E-9
Absolute Gap	0.0000201087
Primal Infeasibility	1.000244E-11
Bound Infeasibility	8.097545E-11
Integer Infeasibility	7.771561E-15
Best Bound	6972.3309548
Nodes	1
Iterations	11
Presolve Time	0.46
Solution Time	34.55

The iteration log, which contains the problem statistics and the progress of the solution, is shown in Output 15.5.4.

Output 15.5.4 Log

```
NOTE: The OPTMILP procedure is executing in single-machine mode.
NOTE: The problem MPSDATA has 52638 variables (16038 binary, 0 integer, 0 free, 0 fixed).
NOTE: The problem has 3949 constraints (3339 LE, 0 EQ, 610 GE, 0 range).
NOTE: The problem has 148866 constraint coefficients.
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 0 variables and 367 constraints.
NOTE: The MILP presolver removed 6606 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 52638 variables, 3582 constraints, and 142260 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The DECOMP method value AUTO is applied.
NOTE: The automated method will attempt to find block-angular form with 4 blocks.
NOTE: The problem has a decomposable structure with 610 blocks. The largest block covers 0.22%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 52638 (100.00%) variables and 3574 (99.78%)
      constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter         Best       Master        Best        LP       IP  CPU Real
                  Bound    Objective     Integer       Gap      Gap Time Time
NOTE: Starting phase 1.
         1      0.0000     302.6667               . 3.03e+02        .   12    3
         4      0.0000       0.0000               .    0.00%        .   13    3
NOTE: Starting phase 2.
         6   7586.8795    6393.7089   6387.7547   15.73%   15.81%   33   10
         7   7119.4327    6751.1789   6387.7547    5.17%   10.28%   52   15
         8   6976.4120    6932.9565   6387.7547    0.62%    8.44%   77   22
         9   6976.4120    6963.3497   6963.3497    0.19%    0.19%  102   29
         .   6976.4120    6968.0658   6968.0658    0.12%    0.12%  103   30
        10   6976.4120    6968.0658   6968.0658    0.12%    0.12%  108   31
        11   6972.3310    6972.3309   6972.3309    0.00%    0.00%  113   33
      Node  Active   Sols        Best        Best      Gap    CPU   Real
                             Integer       Bound            Time   Time
         0       0      4   6972.3309   6972.3310    0.00%    113     33
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 33.97 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 6972.3309347.
```

As stated in the log, the automated method attempts to find a balanced block-angular form that contains four blocks (the same setting is used by default in the NTHREADS= option). The algorithm successfully finds such a decomposition and then further decomposes each block into its weakly connected components, resulting in 610 blocks and 99.78% subproblem coverage.

Example 15.6: Bin Packing Problem

The bin packing problem (BPP) finds the minimum number of capacitated bins that are needed to store a set of products of varying size. Define a set P of products, their sizes s_p, and a set $B = \{1, \ldots, |P|\}$ of candidate bins, each having capacity C. Let x_{pb} be a binary variable that, if set to 1, indicates that product p is assigned to bin b. In addition, let y_b be a binary variable that, if set to 1, indicates that bin b is used.

A BPP can be formulated as a MILP as follows:

$$
\begin{array}{llll}
\text{minimize} & \displaystyle\sum_{b \in B} y_b & & \\[2em]
\text{subject to} & \displaystyle\sum_{b \in B} x_{pb} = 1 & p \in P & \text{(Assignment)} \\[2em]
& \displaystyle\sum_{p \in P} s_p x_{pb} \leq C y_b & b \in B & \text{(Capacity)} \\[1.5em]
& x_{pb} \in \{0, 1\} & p \in P, \; b \in B & \\[0.5em]
& y_b \in \{0, 1\} & b \in B &
\end{array}
$$

In this formulation, the Assignment constraints ensure that each product is assigned to exactly one bin. The Capacity constraints ensure that the capacity restrictions are met for each bin. In addition, these constraints enforce the condition that if any product is assigned to bin b, then y_b must be positive.

In this formulation, the bin identifier is arbitrary. For example, in any solution, the assignments to bin 1 can be swapped with the assignments to bin 2 without affecting feasibility or the objective value. Consider a decomposition by bin, where the Assignment constraints form the master problem and the Capacity constraints form identical subproblems. As described in the section "Special Case: Identical Blocks and Ryan-Foster Branching" on page 728, this is a situation in which an aggregate formulation and Ryan-Foster branching can greatly improve performance by reducing symmetry.

Consider a series of University of North Carolina basketball games that are recorded on a DVR. The following data set, dvr, provides the name of each game in the column opponent and the size of that game in gigabytes (GB) as it resides on the DVR in the column size:

```
/* game, size (in GBs) */
data dvr;
   input opponent $ size;
   datalines;
Clemson  1.36
Clemson2 1.97
Duke     2.76
Duke2    2.52
FSU      2.56
FSU2     2.34
GT       1.49
GT2      1.12
IN       1.45
KY       1.42
Loyola   1.42
MD       1.33
MD2      2.71
```

```
    Miami     1.22
    NCSU      2.52
    NCSU2     2.54
    UConn     1.25
    VA        2.33
    VA2       2.48
    VT        1.41
    Vermont   1.28
    WM        1.25
    WM2       1.23
    Wake      1.61
    ;
```

The goal is to use the fewest DVDs on which to store the games for safekeeping. Each DVD can hold 4.38GB reforded data. The problem can be formulated as a bin packing problem and solved by using PROC OPTMODEL and the decomposition algorithm. The following PROC OPTMODEL statements read in the data, declare the optimization model, and use the decomposition algorithm to solve it:

```
proc optmodel;
   /* read the product and size data */
   set <str> PRODUCTS;
   num size {PRODUCTS};
   read data dvr into PRODUCTS=[opponent] size;

   /* 4.38 GBs per DVD */
   num binsize = 4.38;

   /* the number of products is a trivial upper bound on the
      number of bins needed */
   num upperbound init card(PRODUCTS);
   set BINS = 1..upperbound;

   /* Assign[p,b] = 1, if product p is assigned to bin b */
   var Assign {PRODUCTS, BINS} binary;
   /* UseBin[b] = 1, if bin b is used */
   var UseBin {BINS} binary;

   /* minimize number of bins used */
   min Objective = sum {b in BINS} UseBin[b];

   /* assign each product to exactly one bin */
   con Assignment {p in PRODUCTS}:
      sum {b in BINS} Assign[p,b] = 1;

   /* Capacity constraint on each bin (and definition of UseBin) */
   con Capacity {b in BINS}:
      sum {p in PRODUCTS} size[p] * Assign[p,b] <= binsize * UseBin[b];

   /* decompose by bin (subproblem is a knapsack problem) */
   for {b in BINS} Capacity[b].block = b;

   /* solve using decomp (aggregate formulation) */
   solve with milp / decomp;
```

The following OPTMODEL statements create a sequential numbering of the bins and then output to the data set dvd the optimal assignments of games to bins:

```
/* create a map from arbitrary bin number to sequential bin number */
num binId init 1;
num binMap {BINS};
for {b in BINS: UseBin[b].sol > 0.5} do;
   binMap[b] = binId;
   binId     = binId + 1;
end;

/* create map of product to bin from solution */
num bin {PRODUCTS};
for {p in PRODUCTS} do;
   for {b in BINS: Assign[p,b].sol > 0.5} do;
      bin[p] = binMap[b];
      leave;
   end;
end;

/* create solution data */
create data dvd from [product] bin size;
quit;
```

The solution summary is displayed in Output 15.6.1.

Output 15.6.1 Solution Summary

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	Objective
Solution Status	Optimal within Relative Gap
Objective Value	11
Relative Gap	1.356491E-14
Absolute Gap	1.49214E-13
Primal Infeasibility	3.330669E-16
Bound Infeasibility	2.220446E-16
Integer Infeasibility	3.330669E-16
Best Bound	11
Nodes	1
Iterations	15
Presolve Time	0.01
Solution Time	0.08

The iteration log is displayed in Output 15.6.2.

Output 15.6.2 Log

```
NOTE: There were 24 observations read from the data set WORK.DVR.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 600 variables (0 free, 0 fixed).
NOTE: The problem has 600 binary and 0 integer variables.
NOTE: The problem has 48 linear constraints (24 LE, 24 EQ, 0 GE, 0 range).
NOTE: The problem has 1176 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 0 variables and 0 constraints.
NOTE: The MILP presolver removed 0 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 600 variables, 48 constraints, and 1176 constraint coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: All blocks are identical and the master model is set partitioning.
NOTE: The Decomposition algorithm is using an aggregate formulation and Ryan-Foster branching.
NOTE: The problem has a decomposable structure with 24 blocks. The largest block covers 2.08%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 600 (100.00%) variables and 24 (50.00%) constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter          Best        Master         Best        LP       IP  CPU Real
                   Bound     Objective      Integer       Gap      Gap Time Time
NOTE: Starting phase 1.
         1        0.0000        0.0000            .      0.00%        .    0    0
NOTE: Starting phase 2.
         3        0.0000       11.0000      11.0000 1.10e+01 1.10e+01    0    0
         .        0.0000       11.0000      11.0000 1.10e+01 1.10e+01    0    0
        10        0.0000       11.0000      11.0000 1.10e+01 1.10e+01    0    0
        12        3.0000       11.0000      11.0000   266.67%   266.67%   0    0
        15       11.0000       11.0000      11.0000     0.00%     0.00%   0    0
      Node   Active   Sols         Best         Best      Gap    CPU   Real
                             Integer        Bound           Time   Time
         0        0      2      11.0000      11.0000    0.00%      0      0
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 0.07 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 11.
NOTE: The data set WORK.DVD has 24 observations and 3 variables.
```

The following call to PROC SORT sorts the assignments by bin:

```
proc sort data=dvd;
   by bin;
run;
```

The optimal assignments from the output data set dvd are displayed in Figure 15.7.

Figure 15.7 Optimal Assignment of Games to DVDs

bin=1

product	size
Clemson	1.36
WM	1.25
Wake	1.61
bin	**4.22**

bin=2

product	size
Duke2	2.52
UConn	1.25
bin	**3.77**

bin=3

product	size
FSU	2.56
Miami	1.22
bin	**3.78**

bin=4

product	size
MD	1.33
VA	2.33
bin	**3.66**

bin=5

product	size
Loyola	1.42
NCSU2	2.54
bin	**3.96**

bin=6

product	size
KY	1.42
NCSU	2.52
bin	**3.94**

Figure 15.7 *continued*

bin=7

product	size
IN	1.45
MD2	2.71
bin	**4.16**

bin=8

product	size
GT	1.49
GT2	1.12
WM2	1.23
bin	**3.84**

bin=9

product	size
Clemson2	1.97
FSU2	2.34
bin	**4.31**

bin=10

product	size
Duke	2.76
Vermont	1.28
bin	**4.04**

bin=11

product	size
VA2	2.48
VT	1.41
bin	**3.89**
	43.57

In this example, the objective function ensures that there exists an optimal solution that never assigns a product to more than one bin. Therefore, you could instead model the Assignment constraint as an inequality rather than an equality. In this case, the best performance would come from forcing the use of an aggregate formulation and Ryan-Foster branching by specifying the option VARSEL=RYANFOSTER. An example of doing this is shown in Example 15.8.

Example 15.7: Resource Allocation Problem

This example describes a model for selecting tasks to be run on a shared resource (Gamrath 2010). Consider a set I of tasks and a resource capacity C. Each item $i \in I$ has a profit p_i, a resource utilization level w_i, a starting period s_i, and an ending period e_i. The time horizon considered is from the earliest starting time to the latest ending time of all tasks. With each task, associate a binary variable x_i, which, if set to 1, indicates that the task is running from its start time until just before its end time. A task consumes capacity if it is running. The goal is to select which tasks to run in order to maximize profit while not exceeding the shared resource

capacity. Let $S = \{s_i \mid i \in I\}$ define the set of start times for all tasks, and let $L_s = \{i \in I \mid s_i \leq s < e_i\}$ define the set of tasks that are running at each start time $s \in S$. The problem can be modeled as a mixed integer linear programming problem as follows:

$$\text{maximize} \quad \sum_{i \in I} p_i x_i$$

$$\text{subject to} \quad \sum_{i \in L_s} w_i x_i \leq C \qquad s \in S \qquad \text{(CapacityCon)}$$

$$x_i \in \{0, 1\} \qquad i \in I$$

In this formulation, CapacityCon constraints ensure that the running tasks do not exceed the resource capacity. To illustrate, consider the following five-task example with data: $p_i = (6, 8, 5, 9, 8)$, $w_i = (8, 5, 3, 4, 3)$, $s_i = (1, 3, 5, 7, 8)$, $e_i = (5, 8, 9, 17, 10)$, and $C = 10$. The formulation leads to a constraint matrix that has a *staircase structure* that is determined by tasks coming on and offline:

$$
\begin{array}{rrrrrrrrrrr}
\text{maximize} & 6x_1 & + & 8x_2 & + & 5x_3 & + & 9x_4 & + & 8x_5 & \\
\text{subject to} & 8x_1 & & & & & & & & & \leq \quad 10 \\
& 8x_1 & + & 5x_2 & & & & & & & \leq \quad 10 \\
& & & 5x_2 & + & 3x_3 & & & & & \leq \quad 10 \\
& & & 5x_2 & + & 3x_3 & + & 4x_4 & & & \leq \quad 10 \\
& & & & & 3x_3 & + & 4x_4 & + & 3x_5 & \leq \quad 10 \\
& x_i \in \{0, 1\} & i \in I & & & & & & & &
\end{array}
$$

Lagrangian Decomposition

This formulation clearly has no decomposable structure. However, you can use a common modeling technique known as *Lagrangian decomposition* to bring the model into block-angular form. Lagrangian decomposition works by first partitioning the constraints into blocks. Then, each original variable is split into multiple copies of itself, one copy for each block in which the variable has a nonzero coefficient in the constraint matrix. Constraints are added to enforce the equality of each copy of the original variable. Then, the original constraints can be written in block-angular form by using the duplicate variables.

To apply Lagrangian decomposition to the resource allocation problem, define a set B of blocks and let S_b define the set of start times for a given block b, such that $S = \cup_{b \in B} S_b$. Given this partition of start times, let B_i define the set of blocks in which task $i \in I$ is scheduled to be running. Now, for each task $i \in I$, define duplicate variables x_i^b for each $b \in B_i$. Let m_i define the minimum block index for each class of variable that represents task i. The problem can now be modeled in block-angular form as follows:

$$\text{maximize} \quad \sum_{i \in I} p_i x_i^{m_i}$$

$$\text{subject to} \quad x_i^b = x_i^{m_i} \qquad i \in I,\ b \in B_i \setminus \{m_i\} \qquad \text{(LinkDupVarsCon)}$$

$$\sum_{i \in L_s} w_i x_i^b \leq C \qquad b \in B,\ s \in S_b \qquad \text{(CapacityCon)}$$

$$x_i^b \in \{0, 1\} \qquad i \in I,\ b \in B_i$$

In this formulation, the LinkDupVarsCon constraints ensure that the duplicate variables are equal to the original variables. Now, the five-task example has been transformed from a staircase structure to a block-angular structure:

$$
\begin{array}{rcrcrcrcrcrcrcl}
\text{maximize} & 6x_1^1 & + & 8x_2^1 & + & & & 5x_3^2 & + & 9x_4^2 & + & & & 8x_5^3 & \\
\text{subject to} & & & x_2^1 & - & x_2^2 & & & & & & & & & = & 0 \\
& & & & & x_3^2 & - & & & x_3^3 & & & & & = & 0 \\
& & & & & & & x_4^2 & & & - & x_4^3 & & & = & 0 \\
& 8x_1^1 & & & & & & & & & & & & & \leq & 10 \\
& 8x_1^1 & + & 5x_2^1 & & & & & & & & & & & \leq & 10 \\
& & & 5x_2^2 & + & 3x_3^2 & & & & & & & & & \leq & 10 \\
& & & 5x_2^2 & + & 3x_3^2 & + & 4x_4^2 & & & & & & & \leq & 10 \\
& & & & & 3x_3^3 & + & 4x_4^3 & + & 3x_5^3 & & & & & \leq & 10 \\
\end{array}
$$

$$x_i^b \in \{0, 1\} \quad i \in I, \ b \in B_i$$

To show how to apply Lagrangian decomposition in PROC OPTMODEL, consider the following data set TaskData from Caprara, Furini, and Malaguti (2010) which consists of $|I| = 2697$ tasks:

```
data TaskData;
   input profit weight start end;
   datalines;
100 74    1    12
 98 32    1     9
 73 27    1    22
 98 51    1    31
...
 23 40 2684 2689
 36 85 2685 2687
 65 44 2686 2689
 18 36 2687 2689
 88 57 2688 2689
;
```

Using the MILP Solver Directly in PROC OPTMODEL

The following PROC OPTMODEL statements read in the data and solve the original staircase formulation by calling the MILP solver directly:

```
%macro SetupData(task_data=, capacity=);
   set TASKS;
   num capacity=&capacity;
   num profit{TASKS}, weight{TASKS}, start{TASKS}, end{TASKS};

   read data &task_data into TASKS=[_n_] profit weight start end;
   /* the set of start times */

   set STARTS = setof{i in TASKS} start[i];
   /* the set of tasks i that are active at a given start time s */
   set TASKS_START{s in STARTS}
      = {i in TASKS: start[i] <= s < end[i]};
%mend SetupData;
```

```
%macro ResourceAllocation_Direct(task_data=, capacity=);
   proc optmodel;
      %SetupData(task_data=&task_data,capacity=&capacity);

      /* select task i to come online from period [start to end) */
      var x{TASKS} binary;

      /* maximize the total profit of running tasks */
      max TotalProfit = sum{i in TASKS} profit[i] * x[i];

      /* enforce that the shared resource capacity is not exceeded */
      con CapacityCon{s in STARTS}:
         sum{i in TASKS_START[s]} weight[i] * x[i] <= capacity;

      solve;
   quit;
%mend ResourceAllocation_Direct;

%ResourceAllocation_Direct(task_data=TaskData, capacity=100);
```

The problem summary and solution summary are displayed in Output 15.7.1.

Output 15.7.1 Problem Summary and Solution Summary

The OPTMODEL Procedure

Problem Summary	
Objective Sense	Maximization
Objective Function	TotalProfit
Objective Type	Linear
Number of Variables	2697
Bounded Above	0
Bounded Below	0
Bounded Below and Above	2697
Free	0
Fixed	0
Binary	2697
Integer	0
Number of Constraints	2688
Linear LE (<=)	2688
Linear EQ (=)	0
Linear GE (>=)	0
Linear Range	0
Constraint Coefficients	26880

Output 15.7.1 *continued*

Solution Summary	
Solver	MILP
Algorithm	Branch and Cut
Objective Function	TotalProfit
Solution Status	Optimal within Relative Gap
Objective Value	62519.000416
Relative Gap	0.0000801027
Absolute Gap	5.0083389432
Primal Infeasibility	1.421085E-14
Bound Infeasibility	5.6710247E-7
Integer Infeasibility	7.2750423E-6
Best Bound	62524.008755
Nodes	849
Iterations	34532
Presolve Time	0.05
Solution Time	14.02

The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.7.2.

Output 15.7.2 Log

```
NOTE: There were 2697 observations read from the data set WORK.TASKDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 2697 variables (0 free, 0 fixed).
NOTE: The problem has 2697 binary and 0 integer variables.
NOTE: The problem has 2688 linear constraints (2688 LE, 0 EQ, 0 GE, 0 range).
NOTE: The problem has 26880 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The OPTMODEL presolver is disabled for linear problems.
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 0 variables and 0 constraints.
NOTE: The MILP presolver removed 0 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 2697 variables, 2688 constraints, and 26880 constraint
      coefficients.
NOTE: The MILP solver is called.
         Node   Active   Sols    BestInteger       BestBound      Gap     Time
            0        1      3   54182.0000000          145710   62.82%        0
            0        1      3   54182.0000000   73230.2096818   26.01%        0
            0        1      6   60619.0000000   68709.3995483   11.77%        0
            0        1      7   61318.0000000   66335.8247833    7.56%        1
            0        1      7   61318.0000000   64887.3843006    5.50%        2
            0        1      7   61318.0000000   64027.1770833    4.23%        3
            0        1      7   61318.0000000   63508.8490549    3.45%        3
            0        1      7   61318.0000000   63201.3644716    2.98%        4
            0        1      7   61318.0000000   63008.6642092    2.68%        4
            0        1      7   61318.0000000   62915.6674025    2.54%        4
            0        1      7   61318.0000000   62862.9976970    2.46%        4
            0        1      7   61318.0000000   62835.8136672    2.42%        4
            0        1      7   61318.0000000   62817.9245085    2.39%        4
            0        1      7   61318.0000000   62801.4613187    2.36%        4
            0        1      7   61318.0000000   62791.6244128    2.35%        4
            0        1      7   61318.0000000   62787.2794961    2.34%        4
            0        1      8   61432.0000000   62787.2794961    2.16%        4
            0        1      8   61432.0000000   62787.2794961    2.16%        5
NOTE: The MILP solver added 1255 cuts with 8485 cut coefficients at the root.
          100       97      8   61432.0000000   62754.5429591    2.11%        6
          200      192      8   61432.0000000   62734.0925899    2.08%        8
          300      287      8   61432.0000000   62690.7887512    2.01%        9
          400      382      8   61432.0000000   62628.0076901    1.91%       10
          500      478      8   61432.0000000   62614.7974543    1.89%       11
          600      573      8   61432.0000000   62602.1443564    1.87%       12
          700      668      8   61432.0000000   62585.9237739    1.84%       13
          787       24      9   62503.0003956   62524.0686924    0.03%       13
          800       36      9   62503.0003956   62524.0390976    0.03%       13
          815       26     11   62514.0003173   62524.0390976    0.02%       13
          848        4     13   62519.0004163   62524.0087553    0.01%       14
NOTE: Optimal within relative gap.
NOTE: Objective = 62519.000416.
```

Using the Decomposition Algorithm in PROC OPTMODEL

To transform this data into block-angular form, first sort the task data to help reduce the number of duplicate variables needed in the reformulation as follows:

```
proc sort data=TaskData;
   by start end;
run;
```

Then, create the partition of constraints into blocks of size **block_size** as follows:

```
%macro ResourceAllocation_Decomp(task_data=, capacity=, block_size=);
   proc optmodel;
      %SetupData(task_data=&task_data,capacity=&capacity);
      /* partition into blocks of size blocks_size */
      num block_size = &block_size;
      num num_blocks = ceil( card(TASKS) / block_size );
      set BLOCKS      = 1..num_blocks;

      /* the set of starts s for which task i is active */
      set STARTS_TASK{i in TASKS} = {s in STARTS: start[i] <= s < end[i]};

      /* partition the start times into blocks of size block_size */
      set STARTS_BLOCK{BLOCKS} init {};
      num block_id     init 1;
      num block_count init 0;
      for{s in STARTS} do;
         STARTS_BLOCK[block_id] = STARTS_BLOCK[block_id] union {s};
         block_count = block_count + 1;
         if(mod(block_count, block_size) = 0) then
             block_id = block_id + 1;
      end;
```

Then, the following PROC OPTMODEL statements define the block-angular formulation and solve the problem by using the decomposition algorithm, the PRESOLVER=BASIC option, and **block_size=40**. Because this reformulation is equivalent to the original staircase formulation, disabling some of the advanced presolver techniques ensures that the model maintains block-angularity.

```
      /* blocks in which task i is online */
      set BLOCKS_TASK{i in TASKS} =
         {b in BLOCKS: card(STARTS_BLOCK[b] inter STARTS_TASK[i]) > 0};

      /* minimum block id in which task i is online */
      num min_block{i in TASKS} = min{b in BLOCKS_TASK[i]} b;

      /* select task i to come online from period [start to end)
         in each block */
      var x{i in TASKS, b in BLOCKS_TASK[i]} binary;

      /* maximize the total profit of running tasks */
      max TotalProfit = sum{i in TASKS} profit[i] * x[i,min_block[i]];

      /* enforce that task selection is consistent across blocks */
      con LinkDupVarsCon{i in TASKS, b in BLOCKS_TASK[i] diff {min_block[i]}}:
         x[i,b] = x[i,min_block[i]];
```

```
        /* enforce that the shared resource capacity is not exceeded */
        con CapacityCon{b in BLOCKS, s in STARTS_BLOCK[b]}:
            sum{i in TASKS_START[s]} weight[i] * x[i,b] <= capacity;

        /* define blocks for decomposition algorithm */
        for{b in BLOCKS, s in STARTS_BLOCK[b]} CapacityCon[b,s].block = b;

        solve with milp / presolver=basic decomp;
    quit;
%mend ResourceAllocation_Decomp;

%ResourceAllocation_Decomp(task_data=TaskData, capacity=100, block_size=40);
```

The problem summary and solution summary are displayed in Output 15.7.3. Compared to the original formulation, the number of variables and constraints is increased by the number of duplicate variables.

Output 15.7.3 Problem Summary and Solution Summary

The OPTMODEL Procedure

Problem Summary	
Objective Sense	Maximization
Objective Function	TotalProfit
Objective Type	Linear
Number of Variables	3300
Bounded Above	0
Bounded Below	0
Bounded Below and Above	3300
Free	0
Fixed	0
Binary	3300
Integer	0
Number of Constraints	3291
Linear LE (<=)	2688
Linear EQ (=)	603
Linear GE (>=)	0
Linear Range	0
Constraint Coefficients	28086

Output 15.7.3 *continued*

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalProfit
Solution Status	Optimal within Relative Gap
Objective Value	62524
Relative Gap	0.0000693019
Absolute Gap	4.3333333333
Primal Infeasibility	1.421085E-14
Bound Infeasibility	6.661338E-16
Integer Infeasibility	2.997602E-15
Best Bound	62528.333333
Nodes	14
Iterations	92
Presolve Time	0.07
Solution Time	15.97

The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.7.4.

Output 15.7.4 Log

```
NOTE: There were 2697 observations read from the data set WORK.TASKDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 3300 variables (0 free, 0 fixed).
NOTE: The problem has 3300 binary and 0 integer variables.
NOTE: The problem has 3291 linear constraints (2688 LE, 603 EQ, 0 GE, 0 range).
NOTE: The problem has 28086 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value BASIC is applied.
NOTE: The MILP presolver removed 0 variables and 0 constraints.
NOTE: The MILP presolver removed 0 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 3300 variables, 3291 constraints, and 28086 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: The problem has a decomposable structure with 68 blocks. The largest block covers 1.22%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 3300 (100.00%) variables and 2688 (81.68%)
      constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
```

Iter	Best Bound	Master Objective	Best Integer	LP Gap	IP Gap	CPU Time	Real Time
NOTE: Starting phase 1.							
1	0.0000	0.0000	.	0.00%	.	0	0
NOTE: Starting phase 2.							
4	65890.0022	54643.0000	54643.0000	17.07%	17.07%	1	0
6	65890.0022	55803.0000	55803.0000	15.31%	15.31%	1	0
8	65890.0022	57033.0000	57033.0000	13.44%	13.44%	2	1
12	65890.0022	58656.3333	58649.0000	10.98%	10.99%	3	1
15	65890.0022	59958.3333	59951.0000	9.00%	9.01%	4	1
18	65890.0022	61274.0000	61274.0000	7.01%	7.01%	5	2
21	65890.0022	61450.0000	61402.0000	6.74%	6.81%	6	2
23	64827.4672	61755.0000	61707.0000	4.74%	4.81%	7	3
26	64188.2500	62160.0000	62160.0000	3.16%	3.16%	8	3
28	63540.6667	62228.0000	62228.0000	2.07%	2.07%	8	3
.	63540.6667	62288.3333	62280.0000	1.97%	1.98%	9	4
30	63179.0000	62288.3333	62280.0000	1.41%	1.42%	9	4
31	63054.8333	62312.3333	62280.0000	1.18%	1.23%	10	4
32	62952.1728	62390.0833	62280.0000	0.89%	1.07%	11	4
33	62884.3374	62395.5833	62280.0000	0.78%	0.96%	11	5
37	62671.3333	62505.3333	62472.0000	0.26%	0.32%	12	5
38	62569.3333	62521.3333	62472.0000	0.08%	0.16%	13	5
39	62553.3333	62521.3333	62472.0000	0.05%	0.13%	13	6
.	62553.3333	62527.3333	62472.0000	0.04%	0.13%	13	6
40	62553.3333	62527.3333	62472.0000	0.04%	0.13%	13	6
41	62534.3333	62527.3333	62472.0000	0.01%	0.10%	14	6

Output 15.7.4 *continued*

```
      42   62528.3333   62527.3333   62472.0000    0.00%    0.09%   14    6
NOTE: The Decomposition algorithm stopped on the continuous RELOBJGAP= option.
NOTE: Starting branch and bound.
        Node   Active   Sols       Best         Best      Gap    CPU    Real
                                  Integer       Bound            Time   Time
          0       1     26    62472.0000   62528.3333   0.09%    14     6
          3       5     28    62509.0000   62528.3333   0.03%    22    10
          5       5     30    62515.0000   62528.3333   0.02%    26    11
          8       4     31    62522.0000   62528.3333   0.01%    32    13
         10       2     31    62522.0000   62528.3333   0.01%    36    15
         13       1     32    62524.0000   62528.3333   0.01%    37    15
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 15.77 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 62524.
```

Using a Hybrid Method in PROC OPTMODEL

The decomposition algorithm solves the problem in fewer nodes due to the stronger bound obtained by the reformulation. However, it takes longer than the direct method to find a good feasible solution. The fact that the direct method seems to quickly find good feasible solutions but has weaker bounds motivates the use of a hybrid algorithm. In the macro **%ResourceAllocation_Decomp**, replace the statement,

```
solve with milp / presolver=basic decomp;
```

with the following statements:

```
solve with milp / relobjgap=0.1;
solve with milp / presolver=basic primalin decomp;
```

These statements use the direct method with RELOBJGAP=0.1 to find a good starting solution and then use that result to seed the initial columns of the decomposition algorithm.

The solution summaries are displayed in Output 15.7.5.

Output 15.7.5 Solution Summaries

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Branch and Cut
Objective Function	TotalProfit
Solution Status	Optimal within Relative Gap
Objective Value	61318
Relative Gap	0.0756427586
Absolute Gap	5017.8247833
Primal Infeasibility	0
Bound Infeasibility	2.220446E-16
Integer Infeasibility	2.220446E-16
Best Bound	66335.824783
Nodes	1
Iterations	3092
Presolve Time	0.06
Solution Time	1.73

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalProfit
Solution Status	Optimal within Relative Gap
Objective Value	62524
Relative Gap	0.0000959539
Absolute Gap	6
Primal Infeasibility	2.220446E-16
Bound Infeasibility	6.661338E-16
Integer Infeasibility	6.661338E-16
Best Bound	62530
Nodes	3
Iterations	42
Presolve Time	0.07
Solution Time	7.66

The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.7.6.

Output 15.7.6 Log

```
NOTE: There were 2697 observations read from the data set WORK.TASKDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 3300 variables (0 free, 0 fixed).
NOTE: The problem has 3300 binary and 0 integer variables.
NOTE: The problem has 3291 linear constraints (2688 LE, 603 EQ, 0 GE, 0 range).
NOTE: The problem has 28086 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 603 variables and 603 constraints.
NOTE: The MILP presolver removed 1206 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 2697 variables, 2688 constraints, and 26880 constraint
      coefficients.
NOTE: The MILP solver is called.
         Node   Active    Sols    BestInteger       BestBound      Gap     Time
            0        1       3   54609.0000000          145710   62.52%        0
            0        1       3   54609.0000000   73230.2096818   25.43%        0
            0        1       6   60619.0000000   68709.3995483   11.77%        0
            0        1       7   61318.0000000   66335.8247833    7.56%        1
NOTE: The MILP solver added 734 cuts with 3910 cut coefficients at the root.
NOTE: Optimal within relative gap.
NOTE: Objective = 61318.
NOTE: The MILP presolver value BASIC is applied.
NOTE: The MILP presolver removed 0 variables and 0 constraints.
NOTE: The MILP presolver removed 0 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 3300 variables, 3291 constraints, and 28086 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: The problem has a decomposable structure with 68 blocks. The largest block covers 1.22%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 3300 (100.00%) variables and 2688 (81.68%)
      constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter         Best        Master         Best        LP       IP  CPU Real
                  Bound     Objective      Integer       Gap      Gap Time Time
NOTE: Starting phase 1.
         1       0.0000        0.0000            .     0.00%        .    0    0
NOTE: Starting phase 2.
         2   65890.0003   61360.0000   61360.0000     6.88%    6.88%    1    0
         5   65890.0003   61379.0000   61379.0000     6.85%    6.85%    1    0
         6   65890.0003   61406.0000   61406.0000     6.81%    6.81%    2    0
         8   65890.0003   61571.0000   61571.0000     6.55%    6.55%    3    1
         .   65890.0003   61733.0000   61733.0000     6.31%    6.31%    3    1
        10   65439.0015   61733.0000   61733.0000     5.66%    5.66%    4    1
```

Output 15.7.6 *continued*

12	64743.5001	61827.0000	61827.0000	4.50%	4.50%	4	1
13	64743.5001	61934.0000	61934.0000	4.34%	4.34%	5	2
15	64743.5001	62139.0000	62094.0000	4.02%	4.09%	6	2
17	63631.2500	62284.0000	62224.0000	2.12%	2.21%	7	2
18	62964.7500	62299.0000	62239.0000	1.06%	1.15%	7	3
.	62964.7500	62443.4667	62351.0000	0.83%	0.97%	8	3
21	62920.9333	62443.4667	62351.0000	0.76%	0.91%	9	3
22	62773.5013	62472.8333	62351.0000	0.48%	0.67%	9	4
25	62707.5000	62507.3333	62496.0000	0.32%	0.34%	10	4
27	62583.3335	62522.3333	62505.0000	0.10%	0.13%	10	4
28	62580.3339	62522.3333	62505.0000	0.09%	0.12%	11	5
29	62537.3333	62522.3333	62505.0000	0.02%	0.05%	11	5
.	62537.3333	62528.3333	62524.0000	0.01%	0.02%	11	5
30	62537.3333	62528.3333	62524.0000	0.01%	0.02%	11	5
31	62536.3333	62528.3333	62524.0000	0.01%	0.02%	12	5
32	62532.8333	62528.3333	62524.0000	0.01%	0.01%	12	5

```
NOTE: The Decomposition algorithm stopped on the continuous RELOBJGAP= option.
NOTE: Starting branch and bound.
```

Node	Active	Sols	Best Integer	Best Bound	Gap	CPU Time	Real Time
0	1	21	62524.0000	62532.8333	0.01%	12	5
2	0	21	62524.0000	62530.0000	0.01%	17	7

```
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 7.46 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 62524.
```

By using this hybrid method, you can take advantage of the direct method, which finds a good feasible solution quickly, and the strong bounds provided by the decomposition algorithm.

Using a Built-In Hybrid Method in PROC OPTMODEL

Alternatively, you can use the built-in hybrid method in one call to the solver by using the HYBRID=ON option in the DECOMP statement. This algorithm first processes the root node by using standard MILP techniques. It then proceeds with the decomposition algorithm to complete processing. In the macro `%ResourceAllocation_Decomp`, replace the statement

```
solve with milp / presolver=basic decomp;
```

with the statement

```
solve with milp / presolver=basic decomp=(hybrid=on);
```

The solution summary is displayed in Output 15.7.7.

Output 15.7.7 Solution Summary

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalProfit
Solution Status	Optimal within Relative Gap
Objective Value	62524
Relative Gap	0.0000693019
Absolute Gap	4.3333333333
Primal Infeasibility	4.263256E-14
Bound Infeasibility	8.881784E-16
Integer Infeasibility	8.881784E-16
Best Bound	62528.333333
Nodes	4
Iterations	41
Presolve Time	0.08
Solution Time	14.71

The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.7.8.

Output 15.7.8 Log

```
NOTE: There were 2697 observations read from the data set WORK.TASKDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 3300 variables (0 free, 0 fixed).
NOTE: The problem has 3300 binary and 0 integer variables.
NOTE: The problem has 3291 linear constraints (2688 LE, 603 EQ, 0 GE, 0 range).
NOTE: The problem has 28086 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value BASIC is applied.
NOTE: The MILP presolver removed 0 variables and 0 constraints.
NOTE: The MILP presolver removed 0 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 3300 variables, 3291 constraints, and 28086 constraint
      coefficients.
NOTE: The Decomposition algorithm is using the direct MILP solver for the root node.
NOTE: The MILP solver is called.
         Node   Active    Sols    BestInteger      BestBound       Gap      Time
            0        1       3   52549.0000000   73230.2096818   28.24%         0
            0        1       6   60129.0000000   68724.4564404   12.51%         1
            0        1       8   60352.0000000   66375.4142650    9.07%         2
            0        1       8   60352.0000000   64959.3967107    7.09%         2
            0        1       8   60352.0000000   64022.2814465    5.73%         3
            0        1       8   60352.0000000   63561.9969629    5.05%         4
            0        1       8   60352.0000000   63192.8473290    4.50%         5
            0        1       8   60352.0000000   62995.6489462    4.20%         5
            0        1       8   60352.0000000   62927.0282767    4.09%         5
            0        1       8   60352.0000000   62877.9947740    4.02%         5
            0        1       8   60352.0000000   62840.8625639    3.96%         5
            0        1       8   60352.0000000   62821.1059108    3.93%         5
            0        1       8   60352.0000000   62809.7299610    3.91%         6
            0        1       8   60352.0000000   62795.8029927    3.89%         6
            0        1       8   60352.0000000   62783.6627781    3.87%         6
            0        1       9   60912.0000000   62783.6627781    2.98%         6
NOTE: The MILP solver added 1286 cuts with 8726 cut coefficients at the root.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: The problem has a decomposable structure with 68 blocks. The largest block covers 1.49%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 3300 (100.00%) variables and 3930 (85.86%)
      constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter          Best         Master          Best        LP       IP  CPU Real
                    Bound      Objective       Integer       Gap      Gap Time Time
NOTE: Starting phase 1.
         1        0.0000         0.0000             .      0.00%        .    0    0
NOTE: Starting phase 2.
         3    62783.6628    61679.0000    61679.0000      1.76%    1.76%    0    0
         6    62783.6628    61776.0000    61776.0000      1.60%    1.60%    1    0
```

Output 15.7.8 *continued*

8	62783.6628	61860.5000	61844.0000	1.47%	1.50%	2	0
.	62783.6628	61892.5000	61876.0000	1.42%	1.45%	2	1
10	62783.6628	61892.5000	61876.0000	1.42%	1.45%	2	1
13	62783.6628	62068.5000	62052.0000	1.14%	1.17%	3	1
17	62783.6628	62290.3333	62286.0000	0.79%	0.79%	5	2
19	62783.6628	62366.3333	62347.0000	0.66%	0.70%	6	3
22	62783.6628	62407.3333	62390.0000	0.60%	0.63%	7	3
24	62704.3336	62472.3333	62390.0000	0.37%	0.50%	9	4
26	62628.5000	62515.3333	62498.0000	0.18%	0.21%	9	4
27	62573.3340	62515.3333	62498.0000	0.09%	0.12%	10	4
28	62564.3333	62520.3333	62498.0000	0.07%	0.11%	10	5
29	62543.3333	62521.3333	62499.0000	0.04%	0.07%	11	5
.	62543.3333	62521.3333	62512.0000	0.04%	0.05%	11	5
30	62528.3333	62521.3333	62512.0000	0.01%	0.03%	11	5

```
NOTE: The Decomposition algorithm stopped on the continuous RELOBJGAP= option.
```

.	62528.3333	62523.3333	62514.0000	0.01%	0.02%	11	6

```
NOTE: Starting branch and bound.
```

Node	Active	Sols	Best Integer	Best Bound	Gap	CPU Time	Real Time
0	1	27	62514.0000	62528.3333	0.02%	11	6
3	1	28	62524.0000	62528.3333	0.01%	16	8

```
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 8.49 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 62524.
```

The Tradeoff between Coverage and Subproblem Difficulty

The reformulation of this resource allocation problem provides a nice example of the potential tradeoffs in modeling a problem for use with the decomposition algorithm. As seen in Example 15.2, the strength of the bound is an important factor in the overall performance of the algorithm, but it is not always correlated to the magnitude of the subproblem coverage. In this example, the block size determines the number of blocks. Moreover, it determines the number of linking variables that are needed in the reformulation. At one extreme, if the block size is set to be $|S|$, then the number of blocks is 1, and the number of copies of original variables is 0. Using one block would be equivalent to the original staircase formulation and would not yield a model conducive to decomposition. As the number of blocks is increased, the number of linking variables increases (the size of the master problem), the strength of the decomposition bound decreases, and the difficulty of solving the subproblems decreases. In addition, as the number of blocks and their relative difficulty change, the efficient utilization of your machine's parallel architecture can be affected.

The previous section used a block size of 40. The following statement calls the decomposition algorithm and uses a block size of 130:

```
%ResourceAllocation_Decomp(task_data=TaskData, capacity=100, block_size=130);
```

The solution summary is displayed in Output 15.7.9.

Output 15.7.9 Solution Summary

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalProfit
Solution Status	Optimal within Relative Gap
Objective Value	62524
Relative Gap	0.0000959539
Absolute Gap	6
Primal Infeasibility	1.110223E-15
Bound Infeasibility	4.440892E-16
Integer Infeasibility	1.887379E-15
Best Bound	62530
Nodes	1
Iterations	22
Presolve Time	0.06
Solution Time	6.62

The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.7.10.

Output 15.7.10 Log

```
NOTE: There were 2697 observations read from the data set WORK.TASKDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 2877 variables (0 free, 0 fixed).
NOTE: The problem has 2877 binary and 0 integer variables.
NOTE: The problem has 2868 linear constraints (2688 LE, 180 EQ, 0 GE, 0 range).
NOTE: The problem has 27240 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value BASIC is applied.
NOTE: The MILP presolver removed 0 variables and 0 constraints.
NOTE: The MILP presolver removed 0 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 2877 variables, 2868 constraints, and 27240 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: The problem has a decomposable structure with 21 blocks. The largest block covers 4.53%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 2877 (100.00%) variables and 2688 (93.72%)
      constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter          Best       Master        Best      LP     IP  CPU Real
                   Bound    Objective     Integer     Gap    Gap Time Time
NOTE: Starting phase 1.
         1       0.0000       0.0000           .    0.00%      .    1    0
NOTE: Starting phase 2.
         4   63337.0040   58365.0000   58365.0000   7.85%  7.85%    3    1
         6   63337.0040   59142.0000   59142.0000   6.62%  6.62%    5    1
         9   63337.0040   61339.0000   61339.0000   3.15%  3.15%    6    2
        11   63337.0040   61786.0000   61775.0000   2.45%  2.47%    8    2
        13   63337.0040   62230.0000   62230.0000   1.75%  1.75%   10    3
        15   62956.0002   62263.0000   62263.0000   1.10%  1.10%   13    4
        17   62904.0000   62467.0000   62467.0000   0.69%  0.69%   14    5
        18   62808.0008   62482.5000   62467.0000   0.52%  0.54%   16    5
         .   62808.0008   62524.0000   62524.0000   0.45%  0.45%   16    5
        20   62726.6687   62524.0000   62524.0000   0.32%  0.32%   17    5
        21   62600.5000   62524.0000   62524.0000   0.12%  0.12%   17    6
        22   62530.0000   62524.0000   62524.0000   0.01%  0.01%   18    6
NOTE: The Decomposition algorithm stopped on the integer RELOBJGAP= option.
       Node   Active   Sols        Best         Best     Gap    CPU   Real
                              Integer        Bound           Time   Time
          0        0     17   62524.0000   62530.0000   0.01%     18      6
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 6.45 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 62524.
```

This version of the model provides a stronger bound and requires a smaller branch-and-bound search tree to find an optimal solution.

Example 15.8: Vehicle Routing Problem

The vehicle routing problem (VRP) finds a minimum-cost routing of a fixed number of vehicles to service the demands of a set of customers. Define a set $C = \{2, \dots, |C| + 1\}$ of customers, and a demand, d_c, for each customer c. Let $N = C \cup \{1\}$ be the set of nodes, including the vehicle depot, which are designated as node $i = 1$. Let $A = N \times N$ be the set of arcs, V be the set of vehicles (each of which has capacity L), and c_{ij} be the travel time from node i to node j.

Let y_{ik} be a binary variable that, if set to 1, indicates that node i is visited by vehicle k. Let z_{ijk} be a binary variable that, if set to 1, indicates that arc (i, j) is traversed by vehicle k, and let x_{ijk} be a continuous variable that denotes the amount of product (flow) on arc (i, j) that is carried by vehicle k.

A VRP can be formulated as a MILP as follows:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in A} \sum_{k\in V} c_{ij} z_{ijk} \\
\text{subject to} \quad & \sum_{k\in V} y_{ik} \geq 1 && i \in C && \text{(Assignment)} \\
& \sum_{(i,j)\in A} z_{ijk} = y_{ik} && i \in N,\ k \in V && \text{(LeaveNode)} \\
& \sum_{(j,i)\in A} z_{jik} = y_{ik} && i \in N,\ k \in V && \text{(EnterNode)} \\
& \sum_{(j,i)\in A} x_{jik} - \sum_{(i,j)\in A} x_{ijk} = d_i y_{ik} && i \in C,\ k \in V && \text{(FlowBalance)} \\
& x_{ijk} \leq L z_{ijk} && (i, j) \in A,\ k \in V && \text{(VehicleCapacity)} \\
& y_{1k} = 1 && k \in V && \text{(Depot)} \\
& x_{ijk} \geq 0 && (i, j) \in A,\ k \in V \\
& y_{ik} \in \{0, 1\} && i \in N,\ k \in V \\
& z_{ijk} \in \{0, 1\} && (i, j) \in A,\ k \in V
\end{aligned}
$$

In this formulation, the Assignment constraints ensure that each customer is serviced by at least one vehicle. The objective function ensures that there exists an optimal solution that never assigns a customer to more than one vehicle. The LeaveNode and EnterNode constraints enforce the condition that if node i is visited by vehicle k, then vehicle k must use exactly one arc that enters node i and one arc that leaves node i. Conversely, if node i is not visited by vehicle k, then no arcs that enter or leave node i can be used by vehicle k. The FlowBalance constraints define flow conservation at each node for each vehicle. That is, if a node i is visited by vehicle k, then the amount of product from vehicle k that enters and leaves that node must equal the demand at that node. Conversely, if node i is not visited by vehicle k, then the amount of product from vehicle k that enters and leaves that node must be 0. The VehicleCapacity constraints enforce the condition that the amount of product in each vehicle must always be less than or equal to the vehicle capacity L. Finally, the Depot constraints enforce the condition that each vehicle must start and end at the depot node.

In this formulation, the vehicle identifier is arbitrary. Consider a decomposition by vehicle, where the Assignment constraints form the master problem and all other constraints form identical routing subproblems.

As described in the section "Special Case: Identical Blocks and Ryan-Foster Branching" on page 728, this is a situation in which an aggregate formulation can greatly improve performance by reducing symmetry. Because you know that there exists an optimal solution that satisfies the master Assignment constraints at equality, you can force the use of Ryan-Foster branching by specifying the option VARSEL=RYANFOSTER.

VRPLIB, located at `http://www.coin-or.org/SYMPHONY/branchandcut/VRP/data/index.htm`, is a set of benchmark instances of the VRP. The following data set, vrpdata, represents an instance from VRPLIB that has 22 nodes and eight vehicles (P-n22-k8.vrp), which was originally described in Augerat et al. (1995). The data set lists each node, its coordinates, and its demand.

```
/* number of vehicles available */
%let num_vehicles = 8;
/* capacity of each vehicle */
%let capacity = 3000;
/* node, x coordinate, y coordinate, demand */
data vrpdata;
   input node x y demand;
   datalines;
1  145 215    0
2  151 264 1100
3  159 261  700
4  130 254  800
5  128 252 1400
6  163 247 2100
7  146 246  400
8  161 242  800
9  142 239  100
10 163 236  500
11 148 232  600
12 128 231 1200
13 156 217 1300
14 129 214 1300
15 146 208  300
16 164 208  900
17 141 206 2100
18 147 193 1000
19 164 193  900
20 129 189 2500
21 155 185 1800
22 139 182  700
;
```

The following PROC OPTMODEL statements read in the data, declare the optimization model, and use the decomposition algorithm to solve it:

```
proc optmodel;
   /* read the node location and demand data */
   set NODES;
   num x {NODES};
   num y {NODES};
   num demand {NODES};
   num capacity = &capacity;
   num num_vehicles = &num_vehicles;
   read data vrpdata into NODES=[node] x y demand;
```

```
   set ARCS = {i in NODES, j in NODES: i ne j};
   set VEHICLES = 1..num_vehicles;

   /* define the depot as node 1 */
   num depot = 1;

   /* define the arc cost as the rounded Euclidean distance */
   num cost {<i,j> in ARCS} = round(sqrt((x[i]-x[j])^2 + (y[i]-y[j])^2));

   /* Flow[i,j,k] is the amount of demand carried on arc (i,j) by vehicle k */
   var Flow {ARCS, VEHICLES} >= 0 <= capacity;
   /* UseNode[i,k] = 1, if and only if node i is serviced by vehicle k */
   var UseNode {NODES, VEHICLES} binary;
   /* UseArc[i,j,k] = 1, if and only if arc (i,j) is traversed by vehicle k */
   var UseArc {ARCS, VEHICLES} binary;

   /* minimize the total distance traversed */
   min TotalCost = sum {<i,j> in ARCS, k in VEHICLES} cost[i,j] * UseArc[i,j,k];

   /* each non-depot node must be serviced by at least one vehicle */
   con Assignment {i in NODES diff {depot}}:
      sum {k in VEHICLES} UseNode[i,k] >= 1;

   /* each vehicle must start at the depot node */
   for{k in VEHICLES} fix UseNode[depot,k] = 1;

   /* some vehicle k traverses an arc that leaves node i
      if and only if UseNode[i,k] = 1 */
   con LeaveNode {i in NODES, k in VEHICLES}:
      sum {<(i),j> in ARCS} UseArc[i,j,k] = UseNode[i,k];

   /* some vehicle k traverses an arc that enters node i
      if and only if UseNode[i,k] = 1 */
   con EnterNode {i in NODES, k in VEHICLES}:
      sum {<j,(i)> in ARCS} UseArc[j,i,k] = UseNode[i,k];

   /* the amount of demand supplied by vehicle k to node i must equal demand
      if UseNode[i,k] = 1; otherwise, it must equal 0 */
   con FlowBalance {i in NODES diff {depot}, k in VEHICLES}:
       sum {<j,(i)> in ARCS} Flow[j,i,k] - sum {<(i),j> in ARCS} Flow[i,j,k]
       = demand[i] * UseNode[i,k];

   /* if UseArc[i,j,k] = 1, then the flow on arc (i,j) must be at most capacity
      if UseArc[i,j,k] = 0, then no flow is allowed on arc (i,j) */
   con VehicleCapacity {<i,j> in ARCS, k in VEHICLES}:
      Flow[i,j,k] <= Flow[i,j,k].ub * UseArc[i,j,k];

   /* decomp by vehicle */
   for {i in NODES, k in VEHICLES} do;
      LeaveNode[i,k].block = k;
      EnterNode[i,k].block = k;
   end;
   for {i in NODES diff {depot}, k in VEHICLES} FlowBalance[i,k].block = k;
   for {<i,j> in ARCS, k in VEHICLES} VehicleCapacity[i,j,k].block = k;
```

```
    /* solve using decomp (aggregate formulation) */
    solve with MILP / varsel=ryanfoster decomp=(logfreq=20);
```

The following OPTMODEL statements create node and edge data for the optimal routing:

```
    /* create solution data set */
    str color {k in VEHICLES} =
       ['red' 'green' 'blue' 'black' 'orange' 'gray' 'maroon' 'purple'];
    create data node_data from [i] x y;
    create data edge_data from [i j k]=
       {<i,j> in ARCS, k in VEHICLES: UseArc[i,j,k].sol > 0.5}
       x1=x[i] y1=y[i] x2=x[j] y2=y[j] linecolor=color[k];
quit;
```

The solution summary is displayed in Output 15.8.1.

Output 15.8.1 Solution Summary

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalCost
Solution Status	Optimal
Objective Value	603
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	7.123635E-12
Bound Infeasibility	0
Integer Infeasibility	2.331468E-15
Best Bound	603
Nodes	1
Iterations	69
Presolve Time	0.60
Solution Time	90.35

The iteration log is displayed in Output 15.8.2.

Output 15.8.2 Log

NOTE: There were 22 observations read from the data set WORK.VRPDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 7568 variables (0 free, 8 fixed).
NOTE: The problem has 3872 binary and 0 integer variables.
NOTE: The problem has 4237 linear constraints (3696 LE, 520 EQ, 21 GE, 0 range).
NOTE: The problem has 22528 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 8 variables and 0 constraints.
NOTE: The MILP presolver removed 16 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 7560 variables, 4237 constraints, and 22512 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: All blocks are identical and the master model is set covering.
WARNING: The master model is not a set partitioning and VARSEL=RYANFOSTER. The objective
         function must ensure that there exists at least one optimal solution that fulfills all
         of the master constraints at equality.
NOTE: The Decomposition algorithm is using an aggregate formulation and Ryan-Foster branching.
NOTE: The problem has a decomposable structure with 8 blocks. The largest block covers 12.44%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 7560 (100.00%) variables and 4216 (99.50%)
      constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.

Iter	Best Bound	Master Objective	Best Integer	LP Gap	IP Gap	CPU Time	Real Time
NOTE: Starting phase 1.							
1	0.0000	20.0000	.	2.00e+01	.	0	0
20	0.0000	0.6364	.	6.36e-01	.	4	3
27	0.0000	0.0000	.	0.00%	.	5	5
NOTE: Starting phase 2.							
28	112.0000	959.0455	.	756.29%	.	14	8
35	112.0000	807.0000	807.0000	620.54%	620.54%	21	12
38	112.0000	691.0000	691.0000	516.96%	516.96%	25	14
.	112.0000	691.0000	691.0000	516.96%	516.96%	26	15
40	112.0000	691.0000	691.0000	516.96%	516.96%	28	17
42	235.0000	691.0000	691.0000	194.04%	194.04%	35	21
44	247.0000	691.0000	691.0000	179.76%	179.76%	41	24
46	251.0000	691.0000	691.0000	175.30%	175.30%	45	26
.	251.0000	651.0000	651.0000	159.36%	159.36%	54	32
52	264.0000	648.0000	648.0000	145.45%	145.45%	58	34
54	310.6667	625.3333	648.0000	101.29%	108.58%	66	39
55	403.6000	613.2000	648.0000	51.93%	60.56%	70	41
57	410.0000	613.2000	648.0000	49.56%	58.05%	77	47
.	410.0000	607.6667	614.0000	48.21%	49.76%	80	49

Output 15.8.2 *continued*

Node	Active	Sols	Best Integer	Best Bound	Gap	CPU Time	Real Time
60			471.6667	607.6667	614.0000	28.83%	30.18%	83	51
62			511.8148	606.6296	614.0000	18.53%	19.97%	91	56
63			553.3333	604.0000	604.0000	9.16%	9.16%	96	60
64			564.0000	604.0000	604.0000	7.09%	7.09%	104	63
65			588.0000	604.0000	604.0000	2.72%	2.72%	119	69
66			589.1667	603.8333	604.0000	2.49%	2.52%	132	74
67			597.1667	603.8333	604.0000	1.12%	1.14%	145	79
68			600.3333	603.0000	603.0000	0.44%	0.44%	157	83
69			603.0000	603.0000	603.0000	0.00%	0.00%	172	89

Node	Active	Sols	Best Integer	Best Bound	Gap	CPU Time	Real Time
0	0	9	603.0000	603.0000	0.00%	172	89

NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 89.73 seconds.
NOTE: Optimal.
NOTE: Objective = 603.
NOTE: The data set WORK.NODE_DATA has 22 observations and 3 variables.
NOTE: The data set WORK.EDGE_DATA has 29 observations and 8 variables.

The following DATA step and call to PROC SGPLOT generate a plot of the optimal routing. The plot is displayed in Figure 15.8.3.

```
data sganno(drop=i j);
   retain drawspace "datavalue" linethickness 1;
   set edge_data;
   function = 'line';
run;

proc sgplot data=node_data sganno=sganno;
   scatter x=x y=y / datalabel=i;
   xaxis display=none;
   yaxis display=none;
run;
```

Output 15.8.3 Optimal Routing
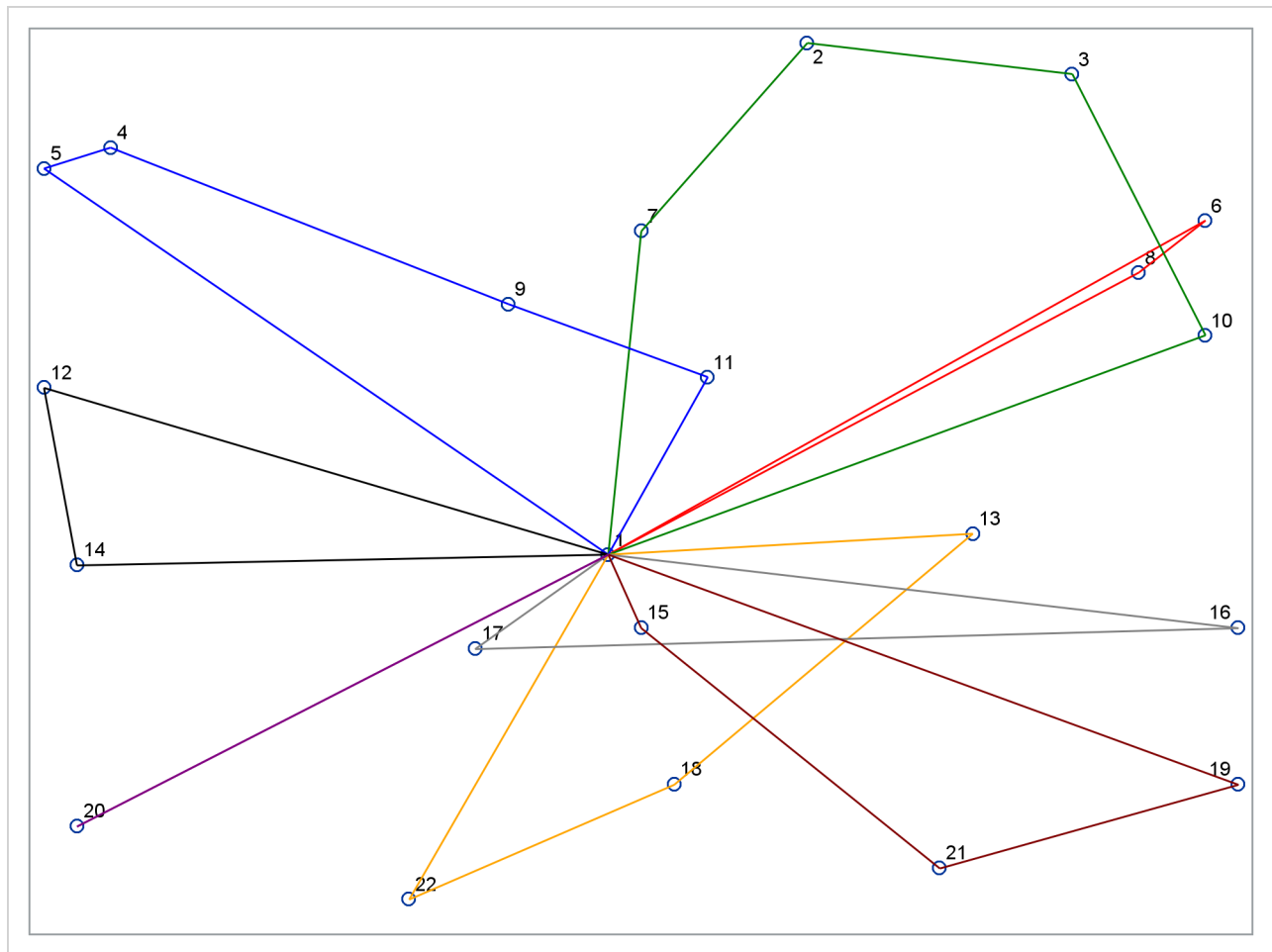
Example 15.9: ATM Cash Management in Single-Machine Mode

This example describes an optimization model that is used in the management of cash flow for a bank's automated teller machine (ATM) network. The goal of the model is to determine a replenishment schedule for the bank to use in allocating cash inventory at its branches when servicing a preassigned subset of ATMs. Given a history of withdrawals per day for each ATM, the bank can use SAS forecasting tools to predict the expected cash need. The modeling of this prediction depends on various seasonal factors, including the days of the week, weeks of the month, holidays, typical salary disbursement days, location of the ATMs, and other demographic data. The prediction is a parametric mixture of models whose parameters depend on each ATM.

The optimization model performs a polynomial regression that minimizes the error (measured by the L_1 norm) between the predicted and actual withdrawals. The parameter settings in the regression determine the replenishment policy. The amount of cash that is allocated to each day is subject to a budget constraint. In addition, a constraint for each ATM limits the number of days that a *cash-out* (a situation in which the cash flow is less than the predicted withdrawal) can occur. The goal is to determine a policy for cash distribution that balances the predicted inventory levels while satisfying the budget and cash-out constraints. By keeping

too much cash on hand for ATM fulfillment, the bank loses an investment opportunity. Moreover, regulatory agencies in many countries enforce a minimum cash reserve ratio at branch banks; according to regulatory policy, the cash in ATMs or in transit does not contribute toward this threshold.

Mixed Integer Nonlinear Programming Formulation

The most natural formulation for this model is in the form of a mixed integer nonlinear program (MINLP). Let A denote the set of ATMs and D denote the set of days that are used in the training data. The predictive model fit is defined by the following data for each ATM a on each day d: $c_{ad}, c^x_{ad}, c^y_{ad}, c^z_{ad}$, and c^u_{ad}. The model-fitting parameters define the variables (x_a, y_a, u_a) for each ATM that, when applied to the predictive model, estimate the necessary cash flow per day per ATM. In addition, define a surrogate variable f_{ad} for each ATM on each day that defines the cash inventory (replenished from the branch) minus withdrawals. The variable f_{ad} also represents the error in the regression model. Let B_d define the budget per day, K_a define the limit on cash-outs per ATM, and w_{ad} define the historical withdrawals at a particular ATM on a particular day. Then the following MINLP models this problem:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{a \in A} \sum_{d \in D} |f_{ad}| \\
\text{subject to} \quad & c^x_{ad} x_a + c^y_{ad} y_a + \\
& c^z_{ad} x_a y_a + c^u_{ad} u_a + c_{ad} - w_{ad} = f_{ad} & a \in A,\ d \in D & \quad \text{(CashFlowDefCon)} \\
& \sum_{a \in A} (f_{ad} + w_{ad}) \le B_d & d \in D & \quad \text{(BudgetCon)} \\
& |\{d \in D \mid f_{ad} < 0\}| \le K_a & a \in A & \quad \text{(CashOutLimitCon)} \\
& x_a, y_a \in [0, 1] & a \in A \\
& u_a \ge 0 & a \in A \\
& f_{ad} \ge -w_{ad} & a \in A,\ d \in D
\end{aligned}
$$

The CashFlowDefCon constraint defines the surrogate variable f_{ad}, which gives the estimated net cash flow. The BudgetCon and CashOutLimitCon constraints ensure that the solution satisfies the budget and cash-out constraints, respectively.

To express this model in a more standard form, you can first use some standard model reformulations to linearize the absolute value and the CashOutLimitCon constraint.

Linearization of Absolute Value
A well-known reformulation for linearizing the absolute value of a variable is to introduce one variable for each side of the absolute value. The following systems are equivalent:

$$
\begin{aligned}
\text{minimize} \quad & |y| \\
\text{subject to} \quad & Ay \le b
\end{aligned}
\qquad \text{is equivalent to} \qquad
\begin{aligned}
\text{minimize} \quad & y^+ + y^- \\
\text{subject to} \quad & A(y^+ - y^-) \le b \\
& y^+, y^- \ge 0
\end{aligned}
$$

Let f_{ad}^{+} and f_{ad}^{-} represent the positive and negative parts, respectively, of the net cash flow f_{ad}. Then you can rewrite the model, removing the absolute value, as the following:

$$\text{minimize} \quad \sum_{a \in A} \sum_{d \in D} \left(f_{ad}^{+} + f_{ad}^{-} \right)$$

subject to

$$c_{ad}^{x} x_a + c_{ad}^{y} y_a +$$
$$c_{ad}^{z} x_a y_a + c_{ad}^{u} u_a + c_{ad} - w_{ad} = f_{ad}^{+} - f_{ad}^{-} \qquad a \in A,\ d \in D$$

$$\sum_{a \in A} \left(f_{ad}^{+} - f_{ad}^{-} + w_{ad} \right) \leq B_d \qquad d \in D$$

$$\left| \{ d \in D \mid (f_{ad}^{+} - f_{ad}^{-}) < 0 \} \right| \leq K_a \qquad a \in A$$

$$x_a, y_a \in [0, 1] \qquad a \in A$$

$$u_a \geq 0 \qquad a \in A$$

$$f_{ad}^{+} \geq 0 \qquad a \in A,\ d \in D$$

$$f_{ad}^{-} \in [0, w_{ad}] \qquad a \in A,\ d \in D$$

Modeling the Cash-Out Constraints

To count the number of times a cash-out occurs, you need to introduce a binary variable to keep track of when this event occurs. Let v_{ad} be an indicator variable that takes the value 1 when the net cash flow is negative. You can model the implication $f_{ad}^{-} > 0 \Rightarrow v_{ad} = 1$, or its contrapositive $v_{ad} = 0 \Rightarrow f_{ad}^{-} \leq 0$, by adding the constraint

$$f_{ad}^{-} \leq w_{ad} v_{ad} \quad a \in A,\ d \in D$$

Now you can model the cash-out constraint by counting the number of days that the net-cash flow is negative for each ATM, as follows:

$$\sum_{d \in D} v_{ad} \leq K_a \quad a \in A$$

The MINLP model can now be written as follows:

$$\text{minimize} \quad \sum_{a \in A} \sum_{d \in D} \left(f_{ad}^{+} + f_{ad}^{-} \right)$$

subject to

$$c_{ad}^{x} x_a + c_{ad}^{y} y_a +$$
$$c_{ad}^{z} x_a y_a + c_{ad}^{u} u_a + c_{ad} - w_{ad} = f_{ad}^{+} - f_{ad}^{-} \qquad a \in A,\ d \in D$$

$$\sum_{a \in A} \left(f_{ad}^{+} - f_{ad}^{-} + w_{ad} \right) \leq B_d \qquad d \in D$$

$$f_{ad}^{-} \leq w_{ad} v_{ad} \qquad a \in A,\ d \in D$$

$$\sum_{d \in D} v_{ad} \leq K_a \qquad a \in A$$

$$x_a, y_a \in [0, 1] \qquad a \in A$$

$$u_a \geq 0 \qquad a \in A$$

$$f_{ad}^{+} \geq 0 \qquad a \in A,\ d \in D$$

$$f_{ad}^{-} \in [0, w_{ad}] \qquad a \in A,\ d \in D$$

$$v_{ad} \in \{0, 1\} \qquad a \in A,\ d \in D$$

This MINLP is difficult to solve, in part because the prediction function is not convex. Another approach is to use mixed integer linear programming (MILP) to formulate an approximation of the problem, as described in the next section.

Mixed Integer Linear Programming Approximation

Because the predictive model is a forecast, finding the optimal parameters that are based on nondeterministic data is not of primary importance. Rather, you want to provide as good a solution as possible in a reasonable amount of time. So using MILP to approximate the MINLP is perfectly acceptable. In the original problem you have products of two continuous variables that are both bounded by 0 (lower bound) and 1 (upper bound). This arrangement enables you to create an approximate linear model by using a few standard modeling reformulations.

Discretization of Continuous Variables

The first step is to discretize one of the continuous variables x_a. The goal is to transform the product $x_a y_a$ of a continuous variable and another continuous variable instead to the product of a continuous variable and a binary variable. This transformation enables you to linearize the product form.

You must assume some level of approximation by defining a binary variable (from some discrete set) for each possible setting of the continuous variable For example, if you let $n = 10$, then you allow x to be chosen from the set $\{0.0, 0.1, 0.2, 0.3, ..., 1.0\}$. Let $T = \{0, 1, 2, ..., n\}$ represent the possible steps and $c_t = t/n$. Then you apply the following transformation to variable x_a:

$$\sum_{t \in T} c_t x_{at} = x_a$$

$$\sum_{t \in T} x_{at} = 1$$

$$x_{at} \in \{0, 1\} \quad t \in T$$

The MINLP model can now be approximated as the following:

minimize
$$\sum_{a \in A} \sum_{d \in D} \left(f_{ad}^+ + f_{ad}^- \right)$$

subject to
$$c_{ad}^x \sum_{t \in T} c_t x_{at} + c_{ad}^y y_a +$$

$$c_{ad}^z \sum_{t \in T} c_t x_{at} y_a + c_{ad}^u u_a + c_{ad} - w_{ad} = f_{ad}^+ - f_{ad}^- \qquad a \in A, \, d \in D$$

$$\sum_{t \in T} x_{at} = 1 \qquad a \in A$$

$$\sum_{a \in A} \left(f_{ad}^+ - f_{ad}^- + w_{ad} \right) \leq B_d \qquad d \in D$$

$$f_{ad}^- \leq w_{ad} v_{ad} \qquad a \in A, \, d \in D$$

$$\sum_{d \in D} v_{ad} \leq K_a \qquad a \in A$$

$$y_a \in [0, 1] \qquad a \in A$$

$$u_a \geq 0 \qquad a \in A$$

$$f_{ad}^+ \geq 0 \qquad a \in A, \, d \in D$$

$$f_{ad}^- \in [0, w_{ad}] \qquad a \in A, \, d \in D$$

$$v_{ad} \in \{0, 1\} \qquad a \in A, \, d \in D$$

$$x_{at} \in \{0, 1\} \qquad a \in A, \, t \in T$$

Linearization of Products

You still need to linearize the product terms $x_{at} y_a$ in the cash flow constraint. Because these terms are products of a bounded continuous variable and a binary variable, you can linearize them by introducing for each product another variable, z_{at}, which serves as a surrogate. In general, you know the following relationship between the original variables and their surrogates:

$$
\begin{aligned}
z_t &= x_t y & t \in T \\
\sum_{t \in T} x_t &= 1 \\
x_t &\in \{0, 1\} & t \in T \\
y &\in [0, 1]
\end{aligned}
$$

is equivalent to

$$
\begin{aligned}
z_t &\geq 0 & t \in T \\
z_t &\leq x_t & t \in T \\
\sum_{t \in T} x_t &= 1 \\
\sum_{t \in T} z_t &= y \\
x_t &\in \{0, 1\} & t \in T \\
y &\in [0, 1]
\end{aligned}
$$

Using this relationship to replace each product form, you now can write the problem as an approximate MILP as follows:

$$\text{minimize} \qquad \sum_{a \in A} \sum_{d \in D} \left(f_{ad}^+ + f_{ad}^- \right)$$

$$\text{subject to} \qquad c_{ad}^x \sum_{t \in T} c_t x_{at} + c_{ad}^y y_a +$$

$$c_{ad}^z \sum_{t \in T} c_t z_{at} + c_{ad}^u u_a + c_{ad} - w_{ad} = f_{ad}^+ - f_{ad}^- \qquad a \in A, \ d \in D$$

$$\sum_{t \in T} x_{at} = 1 \qquad a \in A$$

$$\sum_{a \in A} \left(f_{ad}^+ - f_{ad}^- + w_{ad} \right) \le B_d \qquad d \in D \qquad \text{(BudgetCon)}$$

$$f_{ad}^- \le w_{ad} v_{ad} \qquad a \in A, \ d \in D$$

$$\sum_{d \in D} v_{ad} \le K_a \qquad a \in A$$

$$z_{at} \le x_{at} \qquad a \in A, \ t \in T$$

$$\sum_{t \in T} z_{at} = y_a \qquad a \in A$$

$$z_{at} \ge 0 \qquad a \in A, \ t \in T$$

$$y_a \in [0, 1] \qquad a \in A$$

$$u_a \ge 0 \qquad a \in A$$

$$f_{ad}^+ \ge 0 \qquad a \in A, \ d \in D$$

$$f_{ad}^- \in [0, w_{ad}] \qquad a \in A, \ d \in D$$

$$v_{ad} \in \{0, 1\} \qquad a \in A, \ d \in D$$

$$x_{at} \in \{0, 1\} \qquad a \in A, \ t \in T$$

PROC OPTMODEL Code

Because it is difficult to solve the MINLP model directly, the approximate MILP formulation is attractive. Unfortunately, the approximate MILP is much larger than the associated MINLP. Direct methods for solving this MILP do not work well. However, the problem is nicely suited for the decomposition algorithm.

When you examine the structure of the MILP model, you see clearly that the constraints can be easily decomposed by ATM. In fact, the only set of constraints that involve decision variables across ATMs is the BudgetCon constraint. That is, if you relax the budget constraint, you are left with independent blocks of constraints, one for each ATM.

To show how this is done in PROC OPTMODEL, consider the following data sets, which describe an example that tracks 20 ATMs over a period of 100 days. This particular example was submitted to MIPLIB 2010, which is a collection of difficult MILPs in the public domain (Koch et al. 2011).

The first data set, budget_data, provides the cash budget on each particular day:

```
data budget_data;
   input d $ budget;
   datalines;
DATE0      70079
DATE1      66418
DATE10     52656
DATE11     50439
DATE12     58688
DATE13     45002
DATE14     52369
...
;
```

The second data set, cashout_data, provides the limit on the number of cash-outs that are allowed at each ATM:

```
data cashout_data;
   input a $ cashOutLimit;
   datalines;
ATM0             31
ATM1             24
ATM2             41
ATM3             43
ATM4             29
ATM5             24
ATM6             52
ATM7             44
ATM8             35
ATM9             48
ATM10            31
ATM11            47
ATM12            26
ATM13            34
ATM14            29
ATM15            32
ATM16            33
ATM17            32
ATM18            43
ATM19            28
;
```

The final data set, polyfit_data, provides the polynomial fit coefficients for each ATM on each date. It also provides the historical cash withdrawals.

```
data polyfit_data;
   input a $ d $ cx cy cz cu c withdrawal;
   datalines;
ATM0    DATE0      2822    1984   -1984    1045    1373      780
ATM0    DATE1      1337    2530   -2530    1510     174     2351
ATM0    DATE2      2685     -67      67     145    2820     2288
```

```
ATM0     DATE3       -595    -3135    3135     581    3319    1357
...
ATM19    DATE96      -734     3392   -3392     162    1648     914
ATM19    DATE97     -1062      969    -969     444    1746    2264
ATM19    DATE98      7676     2308   -2308      59    1388     972
ATM19    DATE99      3062     1308   -1308    1080     654     698
;
```

The following PROC OPTMODEL statements read in the data and define the necessary sets and parameters:

```
proc optmodel;
   set<str> DATES;
   set<str> ATMS;

   /* cash budget per date */
   num budget{DATES};

   /* maximum number of cash-outs allowed at each atm */
   num cashOutLimit{ATMS};

   /* historical withdrawal amount per atm each date */
   num withdrawal{ATMS, DATES};

   /* polynomial fit coefficients for predicted cash flow needed */
   num c {ATMS, DATES};
   num cx{ATMS, DATES};
   num cy{ATMS, DATES};
   num cz{ATMS, DATES};
   num cu{ATMS, DATES};

   /* number of points used in approximation of continuous range */
   num nSteps = 10;
   set STEPS = {0..nSteps};

   read data budget_data into DATES=[d] budget;
   read data cashout_data into ATMS=[a] cashOutLimit;
   read data polyfit_data into [a d] cx cy cz cu c withdrawal;
```

The following statements declare the variables:

```
   var x{ATMS,STEPS}                     binary;
   var v{ATMS,DATES}                     binary;
   var z{ATMS,STEPS}                     >= 0 <= 1;
   var y{ATMS}                           >= 0 <= 1;
   var u{ATMS}                           >= 0;
   var fPlus{ATMS,DATES}                 >= 0;
   var fMinus{a in ATMS, d in DATES} >= 0 <= withdrawal[a,d];
```

The following statements declare the objective and the constraints:

```
   min CashFlowDiff =
      sum{a in ATMS, d in DATES} (fPlus[a,d] + fMinus[a,d]);

   con BudgetCon{d in DATES}:
      sum{a in ATMS} (fPlus[a,d] - fMinus[a,d] + withdrawal[a,d])
         <= budget[d];
```

```
con CashFlowDefCon{a in ATMS, d in DATES}:
   cx[a,d] * sum{t in STEPS} (t/nSteps) * x[a,t] +
   cy[a,d] * y[a]                                 +
   cz[a,d] * sum{t in STEPS} (t/nSteps) * z[a,t] +
   cu[a,d] * u[a]                                 +
   c[a,d] - withdrawal[a,d] = fPlus[a,d] - fMinus[a,d];

con PickOneStepCon{a in ATMS}:
   sum{t in STEPS} x[a,t] = 1;

con CashOutLinkCon{a in ATMS, d in DATES}:
   fMinus[a,d] <= withdrawal[a,d] * v[a,d];

con CashOutLimitCon{a in ATMS}:
   sum{d in DATES} v[a,d] <= cashOutLimit[a];

con Linear1Con{a in ATMS, t in STEPS}:
   z[a,t] <= x[a,t];

con Linear2Con{a in ATMS}:
   sum{t in STEPS} z[a,t] = y[a];
```

The following statements define the block decomposition by ATM. The `.block` suffix expects numeric indices, whereas the **SET<STR> ATMS** statement declares a set of strings. You can create a mapping from the string identifier to a numeric identifier as follows:

```
/* create numeric block index */
num blockIndex {ATMS};
num index init 0;
for{a in ATMS} do;
   blockIndex[a] = index;
   index = index + 1;
end;
```

Then, each constraint can be added to its associated ATM block as follows:

```
/* define blocks for each ATM */
for{a in ATMS} do;
   PickOneStepCon[a].block  = blockIndex[a];
   CashOutLimitCon[a].block = blockIndex[a];
   Linear2Con[a].block      = blockIndex[a];
   for{d in DATES} do;
      CashFlowDefCon[a,d].block = blockIndex[a];
      CashOutLinkCon[a,d].block = blockIndex[a];
   end;
   for{t in STEPS} do;
      Linear1Con[a,t].block = blockIndex[a];
   end;
end;
```

The budget constraint links all the ATMs, and it remains in the master problem. Finally, the following statements use DECOMP to solve the problem:

```
/* set the number of threads and get performance details */
performance details nthreads=4;

/* solve with the decomposition algorithm */
solve with milp / decomp;
quit;
```

The solution summary, performance information, and procedure task timing tables are displayed in Output 15.9.1.

Output 15.9.1 Performance Information, Solution Summary, and Task Timing Tables

The OPTMODEL Procedure

Performance Information	
Execution Mode	Single-Machine
Number of Threads	4

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	CashFlowDiff
Solution Status	Optimal within Relative Gap
Objective Value	2463699.6896
Relative Gap	0.0000626319
Absolute Gap	154.29644768
Primal Infeasibility	1.76442E-10
Bound Infeasibility	4.440892E-16
Integer Infeasibility	4.207745E-14
Best Bound	2463545.3932
Nodes	15
Iterations	25
Presolve Time	1.03
Solution Time	188.80

Procedure Task Timing		
Task	**Time (sec.)**	**Time**
Problem Generation	0.04	0.02%
Solver Initialization	0.04	0.02%
Code Generation	0.00	0.00%
Solver	188.80	99.95%
Solver Postprocessing	0.02	0.01%

The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.9.2.

Output 15.9.2 Log

```
NOTE: There were 100 observations read from the data set WORK.BUDGET_DATA.
NOTE: There were 20 observations read from the data set WORK.CASHOUT_DATA.
NOTE: There were 2000 observations read from the data set WORK.POLYFIT_DATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 6480 variables (0 free, 0 fixed).
NOTE: The problem has 2220 binary and 0 integer variables.
NOTE: The problem has 4380 linear constraints (2340 LE, 2040 EQ, 0 GE, 0 range).
NOTE: The problem has 58878 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 553 variables and 385 constraints.
NOTE: The MILP presolver removed 1303 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 5927 variables, 3995 constraints, and 57575 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: The problem has a decomposable structure with 20 blocks. The largest block covers 5.13%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 5927 (100.00%) variables and 3895 (97.50%)
      constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter         Best        Master        Best        LP      IP  CPU Real
                  Bound     Objective     Integer       Gap     Gap Time Time
NOTE: Starting phase 1.
         1       0.0000        1.1767          . 1.18e+00         .   71   25
         2       0.0000        0.0000          .    0.00%         .   71   25
NOTE: Starting phase 2.
         .    2.4432e+06    2.6922e+06   2.8203e+06    10.19%  15.44%   71   26
         4    2.4526e+06    2.4878e+06   2.8203e+06     1.44%  14.99%  162   54
         5    2.4630e+06    2.4642e+06   2.8203e+06     0.05%  14.51%  193   62
NOTE: The Decomposition algorithm stopped on the continuous RELOBJGAP= option.
         .    2.4630e+06    2.4632e+06   2.4701e+06     0.01%   0.29%  194   62
NOTE: Starting branch and bound.
         Node   Active   Sols        Best          Best      Gap    CPU    Real
                                  Integer         Bound            Time    Time
            0        1      2   2.4701e+06    2.4630e+06    0.29%    194      62
            5        7      3   2.4660e+06    2.4635e+06    0.10%    377     113
           10       10      4   2.4654e+06    2.4635e+06    0.07%    524     155
           14        4      5   2.4637e+06    2.4635e+06    0.01%    636     187
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 187.65 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 2463699.6896.
```

Example 15.10: ATM Cash Management in Distributed Mode

This section illustrates how you can use PROC OPTMODEL and the decomposition algorithm in distributed mode. The problem is the same as the one described in Example 15.9 for managing the cash flow of an ATM network. The only difference between single-machine and distributed mode is that the PERFORMANCE statement specifies the number of threads to be used in single-machine mode or the number of threads and nodes to be used in distributed mode.

The following statement changes the operating mode to distributed mode:

```
/* set the number of nodes and threads and get performance details */
performance details nodes=5 nthreads=4;
```

The performance information is displayed in Output 15.10.1. When you specify NODES=5 and NTHREADS=4 in the PERFORMANCE statement in distributed mode, each grid node processes up to four threads simultaneously.

Output 15.10.1 Performance Information

Performance Information

Host Node	<< your grid host >
Execution Mode	Distributed
Number of Compute Nodes	5
Number of Threads per Node	4

The solution summary and procedure task timing tables are displayed in Output 15.10.2.

Output 15.10.2 Performance Information, Solution Summary, and Task Timing Tables

Solution Summary

Solver	MILP
Algorithm	Decomposition
Objective Function	CashFlowDiff
Solution Status	Optimal within Relative Gap
Objective Value	2463561.0129
Relative Gap	6.3403263E-6
Absolute Gap	15.619681743
Primal Infeasibility	7.944436E-10
Bound Infeasibility	3.4861E-14
Integer Infeasibility	2.027267E-13
Best Bound	2463545.3932
Nodes	9
Iterations	16
Presolve Time	1.03
Solution Time	46.22

Output 15.10.2 *continued*

Procedure Task Timing

Task	Time (sec.)	Time
Problem Generation	0.04	0.09%
Solver Initialization	0.23	0.50%
Code Generation	0.00	0.00%
Solver	46.22	98.82%
Solver Postprocessing	0.28	0.59%

Performance Information

Host Node	<< your grid host >
Execution Mode	Distributed
Number of Compute Nodes	5
Number of Threads per Node	4

The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.10.3.

Output 15.10.3 Log

```
NOTE: There were 100 observations read from the data set WORK.BUDGET_DATA.
NOTE: There were 20 observations read from the data set WORK.CASHOUT_DATA.
NOTE: There were 2000 observations read from the data set WORK.POLYFIT_DATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 6480 variables (0 free, 0 fixed).
NOTE: The problem has 2220 binary and 0 integer variables.
NOTE: The problem has 4380 linear constraints (2340 LE, 2040 EQ, 0 GE, 0 range).
NOTE: The problem has 58878 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 553 variables and 385 constraints.
NOTE: The MILP presolver removed 1303 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 5927 variables, 3995 constraints, and 57575 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in the distributed computing environment with 5
      worker nodes.
NOTE: The DECOMP method value USER is applied.
NOTE: The problem has a decomposable structure with 20 blocks. The largest block covers 5.13%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 5927 (100.00%) variables and 3895 (97.50%)
      constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
      Iter          Best        Master        Best        LP       IP Real
                   Bound     Objective     Integer       Gap      Gap Time
NOTE: Starting phase 1.
         1        0.0000        1.1767          . 1.18e+00        .    6
         2        0.0000        0.0000          .     0.00%       .    6
NOTE: Starting phase 2.
         .    2.4432e+06    2.6909e+06   2.8092e+06    10.14%   14.98%    7
         4    2.4526e+06    2.4878e+06   2.8092e+06     1.44%   14.54%   19
         5    2.4630e+06    2.4642e+06   2.8092e+06     0.05%   14.06%   22
NOTE: The Decomposition algorithm stopped on the continuous RELOBJGAP= option.
         .    2.4630e+06    2.4632e+06   2.4701e+06     0.01%    0.29%   22
NOTE: Starting branch and bound.
      Node   Active    Sols         Best         Best      Gap    Real
                                 Integer        Bound             Time
         0        1       2   2.4701e+06   2.4630e+06    0.29%     22
         8        2       3   2.4636e+06   2.4635e+06    0.00%     43
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 43.19 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 2463561.0129.
NOTE: The data set WORK.PERFINFO has 4 observations and 3 variables.
```

Notice how this iteration log differs from the iteration log from single-machine mode in Example 15.9. In distributed mode, the processing is done on multiple grid machines, as opposed to being done on one client machine in single-machine mode. In this example, the grid machines and the client machine have different operating systems, and some numerical rounding off leads to different paths in the search space. When you compare two runs on different operating systems (or that use different compilers), this behavior is expected.

Example 15.11: Kidney Donor Exchange

This example looks at an application of integer programming to help create a kidney donor exchange. Suppose someone needs a kidney transplant and a family member is willing to donate a kidney. If the donor and recipient are incompatible (because of conflicting blood types, tissue mismatch, and so on), the transplant cannot proceed. Now suppose two donor-recipient pairs, A and B, are in this same situation, but donor A is compatible with recipient B and donor B is compatible with recipient A. Then two transplants can take place in a two-way swap, which is shown graphically in Figure 15.8.

Figure 15.8 Kidney Donor Exchange Two-Way Swap
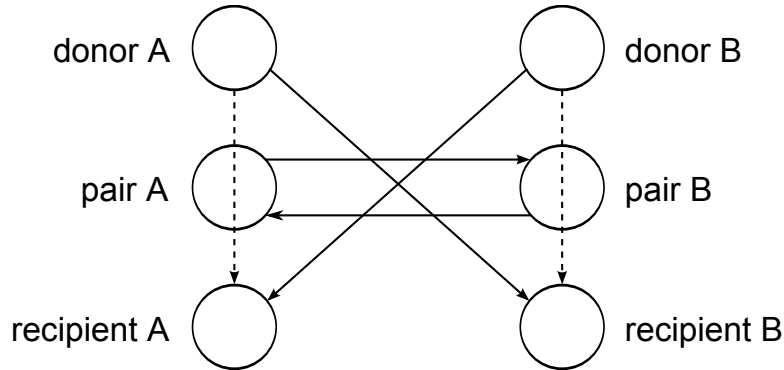
More generally, an *n*-way swap that involves *n* donors and *n* recipients can be performed (Willingham 2009). To model this problem, define a directed graph as follows. Each node is an incompatible donor-recipient pair. Link (i, j) exists if the donor from node i is compatible with the recipient from node j. Let N define the set of nodes and A define the set of arcs. The link weight, w_{ij}, is a measure of the quality of the match. By introducing dummy links whose weight is 0, you can also include altruistic donors who have no recipients or recipients who have no donors. The idea is to find a maximum-weight node-disjoint union of directed cycles. You want the union to be node-disjoint so that no kidney is donated more than once, and you want cycles so that the donor from node i gives up a kidney if and only if the recipient from node i receives a kidney.

Without any other constraints, the problem could be solved as a linear assignment problem. But doing so would allow arbitrarily long cycles in the solution. Because of practical considerations (such as travel) and to mitigate risk, each cycle must have no more than L links. The kidney exchange problem is to find a maximum-weight node-disjoint union of short directed cycles.

Define an index set $M = \{1, \ldots, |N|/2\}$ of candidate disjoint unions of short cycles (called *matchings*). Let x_{ijm} be a binary variable, which, if set to 1, indicates that arc (i, j) is in a matching m. Let y_{im} be a binary variable that, if set to 1, indicates that node i is covered by matching m. In addition, let s_i be a binary slack variable that, if set to 1, indicates that node i is not covered by any matching.

The kidney donor exchange can be formulated as a MILP as follows:

$$
\text{maximize} \quad \sum_{(i,j)\in A}\sum_{m\in M} w_{ij} x_{ijm}
$$

$$
\begin{array}{llll}
\text{subject to} & \displaystyle\sum_{m\in M} y_{im} + s_i = 1 & i \in N & \text{(Packing)} \\[2ex]
& \displaystyle\sum_{(i,j)\in A} x_{ijm} = y_{im} & i \in N,\ m \in M & \text{(Donate)} \\[2ex]
& \displaystyle\sum_{(i,j)\in A} x_{ijm} = y_{jm} & j \in N,\ m \in M & \text{(Receive)} \\[2ex]
& \displaystyle\sum_{(i,j)\in A} x_{ijm} \le L & m \in M & \text{(Cardinality)} \\[2ex]
& x_{ijm} \in \{0,1\} & (i,j) \in A,\ m \in M & \\
& y_{im} \in \{0,1\} & i \in N,\ m \in M & \\
& s_i \in \{0,1\} & i \in N &
\end{array}
$$

In this formulation, the Packing constraints ensure that each node is covered by at most one matching. The Donate and Receive constraints enforce the condition that if node i is covered by matching m, then the matching m must use exactly one arc that leaves node i (Donate) and one arc that enters node i (Receive). Conversely, if node i is not covered by matching m, then no arcs that enter or leave node i can be used by matching m. The Cardinality constraints enforce the condition that the number of arcs in matching m must not exceed L.

In this formulation, the matching identifier is arbitrary. Because it is not necessary to cover each incompatible donor-recipient pair (node), the Packing constraints can be modeled by using set partitioning constraints and the slack variable s. Consider a decomposition by matching, in which the Packing constraints form the master problem and all other constraints form identical matching subproblems. As described in the section "Special Case: Identical Blocks and Ryan-Foster Branching" on page 728, this is a situation in which an aggregate formulation and Ryan-Foster branching can greatly improve performance by reducing symmetry.

The following DATA step sets up the problem, first creating a random graph on n nodes with link probability p and Uniform(0,1) weight:

```
/* create random graph on n nodes with arc probability p
   and uniform(0,1) weight */
%let n = 100;
%let p = 0.02;
data ArcData;
   do i = 0 to &n - 1;
      do j = 0 to &n - 1;
         if i eq j then continue;
         else if ranuni(1) < &p then do;
            weight = ranuni(2);
            output;
         end;
      end;
   end;
run;
```

The following PROC OPTMODEL statements read in the data, declare the optimization model, and use the decomposition algorithm to solve it:

```
%let max_length = 10;
proc optmodel;
   set <num,num> ARCS;
   num weight {ARCS};
   read data ArcData into ARCS=[i j] weight;
   print weight;
   set NODES = union {<i,j> in ARCS} {i,j};
   set MATCHINGS = 1..card(NODES)/2;

   /* UseNode[i,m] = 1 if node i is used in matching m, 0 otherwise */
   var UseNode {NODES, MATCHINGS} binary;

   /* UseArc[i,j,m] = 1 if arc (i,j) is used in matching m, 0 otherwise */
   var UseArc {ARCS, MATCHINGS} binary;

   /* maximize total weight of arcs used */
   max TotalWeight
      = sum {<i,j> in ARCS, m in MATCHINGS} weight[i,j] * UseArc[i,j,m];

   /* each node appears in at most one matching */
   /* rewrite as set partitioning (so decomp uses identical blocks)
      sum{} x <= 1 => sum{} x + s = 1, s >= 0 with no associated cost */
   var Slack {NODES} binary;
   con Packing {i in NODES}:
      sum {m in MATCHINGS} UseNode[i,m] + Slack[i] = 1;

   /* at most one recipient for each donor */
   con Donate {i in NODES, m in MATCHINGS}:
      sum {<(i),j> in ARCS} UseArc[i,j,m] = UseNode[i,m];

   /* at most one donor for each recipient */
   con Receive {j in NODES, m in MATCHINGS}:
      sum {<i,(j)> in ARCS} UseArc[i,j,m] = UseNode[j,m];

   /* exclude long matchings */
   con Cardinality {m in MATCHINGS}:
      sum {<i,j> in ARCS} UseArc[i,j,m] <= &max_length;

   /* decompose by matching (aggregate formulation) */
   for {i in NODES, m in MATCHINGS} Donate[i,m].block = m;
   for {j in NODES, m in MATCHINGS} Receive[j,m].block = m;
   for {m in MATCHINGS} Cardinality[m].block = m;
   solve with milp / presolver=basic decomp;

   /* save solution to a data set */
   create data Solution from
      [m i j]={m in MATCHINGS, <i,j> in ARCS: UseArc[i,j,m].sol > 0.5}
      weight[i,j];
quit;
```

In this case, the PRESOLVER=BASIC option ensures that the model maintains its specified symmetry, enabling the algorithm to use the aggregate formulation and Ryan-Foster branching. The solution summary is displayed in Output 15.11.1.

Output 15.11.1 Solution Summary

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalWeight
Solution Status	Optimal
Objective Value	26.020287142
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	1.776357E-15
Bound Infeasibility	2.220446E-16
Integer Infeasibility	1.554312E-15
Best Bound	26.020287142
Nodes	27
Iterations	151
Presolve Time	1.47
Solution Time	19.98

The iteration log is displayed in Output 15.11.2.

Output 15.11.2 Log

```
NOTE: There were 194 observations read from the data set WORK.ARCDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 14065 variables (0 free, 0 fixed).
NOTE: The problem has 14065 binary and 0 integer variables.
NOTE: The problem has 9457 linear constraints (48 LE, 9409 EQ, 0 GE, 0 range).
NOTE: The problem has 42001 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The MILP presolver value BASIC is applied.
NOTE: The MILP presolver removed 4786 variables and 3298 constraints.
NOTE: The MILP presolver removed 14290 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 9279 variables, 6159 constraints, and 27711 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
NOTE: The DECOMP method value USER is applied.
NOTE: All blocks are identical and the master model is set partitioning.
NOTE: The Decomposition algorithm is using an aggregate formulation and Ryan-Foster branching.
NOTE: The problem has a decomposable structure with 48 blocks. The largest block covers 2.06%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 9216 (99.32%) variables and 6096 (98.98%) constraints.
NOTE: The deterministic parallel mode is enabled.
NOTE: The Decomposition algorithm is using up to 4 threads.
```

Iter	Best Bound	Master Objective	Best Integer	LP Gap	IP Gap	CPU Time	Real Time
NOTE: Starting phase 1.							
1	0.0000	0.0000	.	0.00%	.	0	0
NOTE: Starting phase 2.							
.	390.3703	9.2503	9.2503	97.63%	97.63%	0	0
2	388.4996	9.2503	9.2503	97.62%	97.62%	0	0
3	359.1937	10.9240	10.9240	96.96%	96.96%	0	0
4	352.0565	18.1796	18.1796	94.84%	94.84%	0	0
5	331.3193	18.1796	18.1796	94.51%	94.51%	0	0
7	312.6581	19.5866	18.1796	93.74%	94.19%	0	0
8	261.6997	21.6218	18.1796	91.74%	93.05%	0	0
.	261.6997	22.5854	18.1796	91.37%	93.05%	0	0
10	204.1753	22.5854	18.1796	88.94%	91.10%	0	0
11	202.5642	22.9637	18.1796	88.66%	91.03%	0	0
12	175.9382	22.9637	18.1796	86.95%	89.67%	0	0
17	131.8080	24.4619	18.1796	81.44%	86.21%	0	0
.	131.8080	24.6174	21.9864	81.32%	83.32%	0	0
20	126.8981	24.6174	21.9864	80.60%	82.67%	0	0
23	98.8871	25.1987	21.9864	74.52%	77.77%	1	1
24	85.7817	25.4717	21.9864	70.31%	74.37%	1	1
27	80.5739	25.9325	21.9864	67.82%	72.71%	1	1
.	80.5739	26.2400	21.9864	67.43%	72.71%	1	1
30	73.9660	26.2400	21.9864	64.52%	70.28%	1	1
31	50.6940	26.4105	21.9864	47.90%	56.63%	1	1

Output 15.11.2 *continued*

33	47.0819	26.4542	21.9864	43.81%	53.30%	1	1
38	38.4334	26.7561	21.9864	30.38%	42.79%	2	2
39	34.1428	26.7804	21.9864	21.56%	35.60%	2	2
.	34.1428	26.7804	23.4755	21.56%	31.24%	2	2
40	34.1428	26.7804	23.4755	21.56%	31.24%	3	3
41	32.3711	26.7804	23.4755	17.27%	27.48%	3	3
42	30.1400	26.7804	23.4755	11.15%	22.11%	3	3
43	26.7804	26.7804	23.4755	0.00%	12.34%	3	3

```
NOTE: Starting branch and bound.
```

Node	Active	Sols	Best Integer	Best Bound	Gap	CPU Time	Real Time
0	1	9	23.4755	26.7804	12.34%	3	3
5	7	10	24.8542	26.4468	6.02%	9	8
9	7	11	25.9955	26.3728	1.43%	12	10
10	8	11	25.9955	26.3654	1.40%	12	11
17	7	13	26.0203	26.1602	0.53%	17	14
20	4	13	26.0203	26.0679	0.18%	19	16
26	0	13	26.0203	26.0203	0.00%	21	18

```
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 18.46 seconds.
NOTE: Optimal.
NOTE: Objective = 26.020287142.
NOTE: The data set WORK.SOLUTION has 42 observations and 4 variables.
```

The solution is a set of arcs that define a union of short directed cycles (matchings). The following call to PROC OPTNET extracts the corresponding cycles from the list of arcs and outputs them to the data set Cycles:

```
proc optnet
   direction  = directed
   data_links = Solution;
   data_links_var
      from    = i
      to      = j;
   cycle
      mode    = all_cycles
      out     = Cycles;
run;
```

For more information about PROC OPTNET, see *SAS/OR User's Guide: Network Optimization Algorithms*. Alternatively, you can extract the cycles by using the SOLVE WITH NETWORK statement in PROC OPTMODEL (see Chapter 9, "The Network Solver"). The optimal donor exchanges from the output data set Cycles are displayed in Figure 15.9.

Figure 15.9 Optimal Donor Exchanges

cycle=1

order	node
1	5
2	19
3	56
4	12
5	33
6	70
7	63
8	43
9	15
10	5

cycle=2

order	node
1	13
2	74
3	65
4	41
5	59
6	50
7	49
8	98
9	13

cycle=3

order	node
1	16
2	91
3	17
4	57
5	87
6	72
7	64
8	22
9	88
10	16

Figure 15.9 *continued*

cycle=4

order	node
1	8
2	32
3	79
4	71
5	69
6	26
7	9
8	18
9	95
10	35
11	8

cycle=5

order	node
1	52
2	77
3	94
4	81
5	52

cycle=6

order	node
1	24
2	92
3	24

References

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993), *Network Flows: Theory, Algorithms, and Applications*, Englewood Cliffs, NJ: Prentice-Hall.

Augerat, P., Belenguer, J. M., Benavent, E., Corberán, A., Naddef, D., and Rinaldi, G. (1995), *Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem*, Technical Report 949-M, Université Joseph Fourier, Grenoble.

Aykanat, C., Pinar, A., and Çatalyürek, Ü. V. (2004), "Permuting Sparse Rectangular Matrices into Block-Diagonal Form," *SIAM Journal on Scientific Computing*, 25, 1860–1879.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W. P., and Vance, P. H. (1998), "Branch-and-Price: Column Generation for Solving Huge Integer Programs," *Operations Research*, 46, 316–329.

Caprara, A., Furini, F., and Malaguti, E. (2010), *Exact Algorithms for the Temporal Knapsack Problem*, Technical Report OR-10-7, University of Bologna, Department of Electronics, Computer Science, and Systems.

Dantzig, G. B. and Wolfe, P. (1960), "Decomposition Principle for Linear Programs," *Operations Research*, 8, 101–111.
URL http://www.jstor.org/stable/167547

Galati, M. V. (2009), *Decomposition in Integer Linear Programming*, Ph.D. diss., Lehigh University.

Gamrath, G. (2010), *Generic Branch-Cut-and-Price*, diploma thesis, Technische Universität Berlin.

Grcar, J. F. (1990), *Matrix Stretching for Linear Equations*, Technical Report SAND90-8723, Sandia National Laboratories.

Koch, T., Achterberg, T., Andersen, E. D., Bastert, O., Berthold, T., Bixby, R. E., Danna, E., Gamrath, G., Gleixner, A. M., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D. E., and Wolter, K. (2011), "MIPLIB 2010," *Mathematical Programming Computation*, 3, 103–163.
URL http://mpc.zib.de/index.php/MPC/article/view/56/28

Ralphs, T. K. and Galati, M. V. (2006), "Decomposition and Dynamic Cut Generation in Integer Linear Programming," *Mathematical Programming*, 106, 261–285.

Vanderbeck, F. and Savelsbergh, M. W. P. (2006), "A Generic View of Dantzig-Wolfe Decomposition in Mixed Integer Programming," *Operations Research Letters*, 34, 296–306.

Willingham, V. (2009), "Massive Transplant Effort Pairs 13 Kidneys to 13 Patients," CNN Health, http://www.cnn.com/2009/HEALTH/12/14/kidney.transplant/index.html, accessed March 16, 2011.

Subject Index

Syntax Index