SAS/OR® 14.3 User's Guide
Mathematical Programming
The Decomposition Algorithm

This document is an individual chapter from *SAS/OR® 14.3 User's Guide: Mathematical Programming*.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2017. *SAS/OR® 14.3 User's Guide: Mathematical Programming*. Cary, NC: SAS Institute Inc.

SAS/OR® 14.3 User's Guide: Mathematical Programming

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

September 2017

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

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

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

$$
\begin{array}{lccccccccccc}
\text{minimize} & \mathbf{c}^1\mathbf{x}^1 & + & \mathbf{c}^2\mathbf{x}^2 & + & \cdots & + & \mathbf{c}^\kappa\mathbf{x}^\kappa & + & \mathbf{f}^\top y & & \\
\text{subject to} & \mathbf{D}^1\mathbf{x}^1 & + & \mathbf{D}^2\mathbf{x}^2 & + & \cdots & + & \mathbf{D}^\kappa\mathbf{x}^\kappa & + & \mathbf{B}\mathbf{y} & \{\geq,=,\leq\} & \mathbf{d} \\
 & \mathbf{A}^1\mathbf{x}^1 & & & & & & & & & \{\geq,=,\leq\} & \mathbf{b}^1 \\
 & & & \mathbf{A}^2\mathbf{x}^2 & & & & & & & \{\geq,=,\leq\} & \mathbf{b}^2 \\
 & & & & & \ddots & & & & & \{\geq,=,\leq\} & \vdots \\
 & & & & & & & \mathbf{A}^\kappa\mathbf{x}^\kappa & & & \{\geq,=,\leq\} & \mathbf{b}^\kappa \\
 & & & & \underline{\mathbf{x}} \leq \mathbf{x} \leq \overline{\mathbf{x}} & & & & & & & \\
 & & & & & & & & \underline{\mathbf{y}} \leq \mathbf{y} \leq \overline{\mathbf{y}} & & & \\
 & & & & x_i \in \mathbb{Z} & & & & & & i \in \mathcal{S}_x & \\
 & & & & & & & & & y_i \in \mathbb{Z} & i \in \mathcal{S}_y &
\end{array}
$$

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 628 in Chapter 13, "The OPTMILP Procedure"
- "Details: LP Solver" on page 263 in Chapter 7, "The Linear Programming Solver"
- "Details: MILP Solver" on page 330 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:

$$
\begin{array}{lrcrcrcrcrcll}
\text{maximize} & x_{11} & + & 2x_{21} & + & x_{31} & & + & x_{22} & + & x_{32} & & & \\
\text{subject to} & x_{11} & & & & & + & x_{12} & & & & \geq & 1 & \text{(m)} \\
 & 5x_{11} & + & 7x_{21} & + & 4x_{31} & & & & & & \leq & 11 & \text{(s1)} \\
 & & & & & & & x_{12} & + & 2x_{22} & + & x_{32} \leq & 2 & \text{(s2)} \\
 & & & & & & & & & & & x_{ij} \in & \{0,1\} & i \in \{1,2,3\},\ j \in \{1,2\}
\end{array}
$$

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	1
Presolve Time	0.01
Solution Time	0.11

	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 number of block threads has been reduced to 2 threads.
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%) 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
         1       4.0000            .       4.0000        .    0.00%    0    0
         Node  Active   Sols         Best          Best      Gap    CPU   Real
                                   Integer         Bound            Time   Time
            0       0      2       4.0000       4.0000    0.00%      0      0
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 0.10 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 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 data set to define the subproblem blocks:

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

Now, you can use the following PROC 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 OPTMODEL 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.

You can specify the following decomposition algorithm options in the SOLVE statement in the OPTMODEL procedure:

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

SOLVE WITH MILP / < *options* >
< **DECOMP**< =(*decomp-options*) > >
< **DECOMPMASTER=(**< *master-options* >**)** >
< **DECOMPMASTERIP=(**< *masterip-options* >**)** >
< **DECOMPSUBPROB=(**< *subprob-options* >**)** > ;

You can specify the following statements in the OPTLP procedure:

PROC OPTLP < *options* > ;
DECOMP < *decomp-options* > ;
DECOMPMASTER < *master-options* > ;
DECOMPSUBPROB < *subprob-options* > ;

You can specify the following statements in the OPTMILP procedure:

PROC OPTMILP < *options* > ;
DECOMP < *decomp-options* > ;
DECOMPMASTER < *master-options* > ;
DECOMPMASTERIP < *masterip-options* > ;
DECOMPSUBPROB < *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 shown in Table 15.1. (As the table indicates, 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; 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 256 in Chapter 7, "The Linear Programming Solver," and the section "Syntax: OPTLP Procedure" on page 563 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=	
Specifies whether the model is a maximization or minimization problem	OBJSENSE=	
Specifies the primal solution output data set	PRIMALOUT=	

Table 15.1 (continued)

Description	*option*	Different Default
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 the table indicates, 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; 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 319 and the section "Syntax: OPTMILP Procedure" on page 617.

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

Description	*option*
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=

Table 15.2 (continued)

Description	*option*
Control Options	
Specifies the stopping criterion based on an absolute objective gap	ABSOBJGAP=
Emphasizes feasibility or optimality	EMPHASIS=
Specifies the maximum violation of variables and constraints	FEASTOL=
Specifies the maximum difference allowed 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 process	MAXNODES=
Specifies the maximum number of solutions to find	MAXSOLS=
Specifies the time limit for the optimization process	MAXTIME=
Specifies the tolerance to use 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=
Specifies the stopping criterion based on a 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 number of iterations to perform on each variable for strong branching	STRONGITER=
Specifies the number of candidates for strong branching	STRONGLEN=
Specifies the level of symmetry detection	SYMMETRY=
Specifies the rule for selecting the 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 multicommodity network flow cut level	CUTMULTICOMMODITY=
Specifies the row multiplier factor for cuts	CUTSFACTOR=
Specifies the overall cut aggressiveness	CUTSTRATEGY=
Specifies the zero-half cut level	CUTZEROHALF=

The HYBRID= option in the DECOMP statement indicates the processing mode for the root node of the branch-and-bound search tree. When HYBRID=TRUE, the root node is first processed using standard MILP techniques, as described in the section "Details: MILP Solver" on page 330. The default setting for the decomposition algorithm is HYBRID=FALSE. In this case, the root processing is done solely by the decomposition algorithm, and the following direct MILP options are ignored: EMPHASIS=, SEED=, TARGET=, and all the cut options.

The following search options, listed in Table 15.2, have a different interpretation or a different set of options from what is described in the MILP solver sections:

LOGFREQ=*number*

PRINTFREQ=*number*

specifies how often to print information in the node log. The value of *number* can be any 32-bit integer greater than or equal to 0. The default value is 10. If *number* is 0, then the node log is disabled. If *number* is positive, then an entry is made in the node log at the first node, at the last node, and at intervals dictated by the value of *number*. An entry is also made in the node log each time the solver finds a better integer solution or improved bound.

STRONGITER=*number* | **AUTOMATIC**

specifies the number of pricing iterations that are performed for each variable in the candidate list when you use the strong branching variable selection strategy. The value of *number* can be any positive 32-bit integer. If you specify the keyword AUTOMATIC, the MILP solver uses the default value, which is calculated automatically.

VARSEL=AUTOMATIC | MAXINFEAS | PSEUDO | RYANFOSTER | STRONG

specifies the rule for selecting the branching variable. You can specify the following values:

AUTOMATIC	uses automatic branching variable selection.
MAXINFEAS	selects the variable in the original compact formulation with maximum infeasibility.
PSEUDO	selects the variable in the original compact formulation that maximizes the weighted up and down pseudocosts.
RYANFOSTER	when appropriate, uses a specialized branching rule known as *Ryan-Foster branching*.
STRONG	selects the variable in the original compact formulation that maximizes the estimated improvement in the objective value based on strong branching.

By default, VARSEL=AUTOMATIC. For more information about variable selection, see the sections "Variable Selection" on page 632 and "Special Case: Identical Blocks and Ryan-Foster Branching" on page 727.

DECOMP Statement

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

DECOMPOSITION < *decomp-options* > **;**

The DECOMP statement controls the overall decomposition algorithm.

Table 15.3 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 the table indicates, 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 15.3 Options in the DECOMP Statement

Description	*decomp-option*
Data Set Options (OPTLP and OPTMILP procedures only)	
Specifies the blocks input data set	BLOCKS=
Control Options	
Specifies the stopping criterion based on an absolute objective gap	ABSOBJGAP=
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 block-specification method	METHOD=
Specifies the number of blocks to search for by using METHOD=AUTO	NBLOCKS=
Specifies the number of block threads to use in the decomposition algorithm	NTHREADS=
Specifies the stopping criterion based on a relative objective gap	RELOBJGAP=
Control Options (MILP only)	
Specifies whether to process the root node by using standard MILP techniques	HYBRID=
Specifies how frequently to print the continuous iteration log	LOGFREQ=
Specifies the maximum number of outer iterations for the decomposition algorithm	MAXITER=

You can specify the following *decomp-options*:

ABSOBJGAP=*number*

ABSOLUTEOBJECTIVEGAP=*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 *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 use in the decomposition algorithm if METHOD=USER. For more information, see the section "The BLOCKS= Data Set in PROC OPTMILP and PROC OPTLP" on page 725. To specify blocks in PROC OPTMODEL, use the `.block` constraint suffix instead (see the section "The `.block` Constraint Suffix in PROC OPTMODEL" on page 725).

HYBRID=FALSE | TRUE

specifies whether to first process the root node by using standard MILP techniques, as described in the section "Details: MILP Solver" on page 330. You can specify the following values:

FALSE disables root processing by using standard MILP techniques.

TRUE enables root processing by using standard MILP techniques.

By default, HYBRID=FALSE.

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 32-bit integer. The default value of *number* is 10. If *number* is 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=AUTOMATIC | NONE | BASIC | MODERATE | AGGRESSIVE
controls the amount of information that the decomposition algorithm displays in the SAS log.

You can specify the following values for an LP:

AUTOMATIC prints the continuous iteration log at the interval dictated by the LOGFREQ= main solver option.

NONE turns off printing of all the decomposition algorithm messages to the SAS log.

BASIC prints the continuous iteration log at the interval dictated by the LOGFREQ= main solver option.

MODERATE prints the continuous iteration log and summary information for each iteration at the interval dictated by the LOGFREQ= main solver option.

AGGRESSIVE prints the continuous iteration log and detailed information for each iteration at the interval dictated by the LOGFREQ= main solver option.

You can specify the following values for a MILP:

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.

NONE turns off printing of all the decomposition algorithm messages to the SAS log.

BASIC prints the continuous iteration log for each branch-and-bound node at the interval dictated by the LOGFREQ= option in the DECOMP statement.

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.

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.

By default, LOGLEVEL=AUTOMATIC for both LPs and MILPs.

MAXBLOCKS=*number*
specifies the maximum number of blocks to allow. If the defined number of blocks exceeds *number*, the algorithm creates superblocks by using a very simple round-robin scheme. The value of *number* can be any positive 32-bit integer. The default is the largest number that can be represented by a 32-bit integer.

MAXITER=*number*
specifies (for MILP problems only) the maximum number of outer iterations for the decomposition algorithm. The value of *number* can be any positive 32-bit integer. The default is the largest number that can be represented by a 32-bit integer.

METHOD=AUTO | CONCOMP | NETWORK | SET | USER
specifies the block-specification method. You can specify the following values:

AUTO 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 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 attempts to find an embedded network similar to what is described in the section "The Network Simplex Algorithm" on page 263. The weakly connected components of this network are used as the blocks.

SET attempts to find a set partitioning or set covering structure in the constraint matrix and defines this as the master (linking) constraints. The weakly connected components of the remaining constraints are used as the blocks.

USER uses a user-defined method to specify which rows belong to which blocks (subproblems). In PROC OPTMODEL, use the `.block` constraint suffix. In PROC OPTLP and PROC OPTMILP, use the BLOCKS= data set instead.

By default, METHOD=USER if blocks are defined, and METHOD=AUTO otherwise.

NBLOCKS=*number*
NUMBLOCKS=*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 block threads that are used for processing. This is equivalent to the value of the NTHREADS= option in the DECOMP statement. For more information about parallel execution, see the section "Parallel Processing" on page 727.

NTHREADS=*number*
NUMTHREADS=*number*
specifies the number of block threads to use in the decomposition algorithm. The value of the NTHREADS= option in the main solver statement serves as the overall capacity for the number of active threads that can run at one time. By default, the number of block threads is $t = \min(p, d, b)$, where p is the value of the NTHREADS= option in the main solver statement, d is the value of the NTHREADS= option in the DECOMP statement, and b is the number of blocks that the decomposition algorithm sets or finds.

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.

DECOMPMASTER Statement

DECOMPMASTER < *master-options* > **;**

DECOMPOSITIONMASTER < *master-options* > **;**

MASTER < *master-options* > **;**

The DECOMPMASTER statement controls the master problem.

Table 15.4 summarizes the options available in the DECOMPMASTER statement. These options control the master LP solver in the decomposition algorithm during the solution of an LP or a MILP. (As the table indicates, you can specify the PRINTLEVEL= option only in the OPTLP procedure.) For descriptions of these options, see the section "LP Solver Options" on page 257 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 15.4.

Table 15.4 Options in the DECOMPMASTER Statement

Description	*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†
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=	NONE
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=	
Specifies the type of initial basis	BASIS=	WARMSTART†
Specifies the type of pricing strategy	PRICETYPE=	
Specifies the queue size for determining the entering variable	QUEUESIZE=	

Table 15.4 (continued)

Description	*master-option*	Different Default
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	DUALITYGAP=	

† The reason for the different defaults (ALGORITHM=PS, PRESOLVER=NONE, and BASIS=WARMSTART) is 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 at each iteration.

The following options, listed in Table 15.4, are specific to the DECOMPMASTER statement and are not described in the LP solver sections:

INITPRESOLVER=AUTOMATIC | NONE | BASIC | MODERATE | AGGRESSIVE
INITPRESOL=AUTOMATIC | NONE | BASIC | MODERATE | AGGRESSIVE
specifies, for the first master solve only, the presolve level. You can specify the following values:

AUTOMATIC	applies the default level of presolve processing.
NONE	disables the presolver.
BASIC	performs minimal presolve processing.
MODERATE	applies a higher level of presolve processing.
AGGRESSIVE	applies the highest level of presolve processing.

By default, INITPRESOLVER=AUTOMATIC.

NTHREADS=*number*
NUMTHREADS=*number*
specifies the number of threads to use in the master solver (if the selected solver method supports multithreading). The value of the NTHREADS= option in the main solver statement serves as the overall capacity for the number of active threads that can run at one time. By default, the number of master threads is the value of the NTHREADS= option in the main solver statement.

DECOMPMASTERIP Statement

DECOMPMASTERIP < *masterip-options* > **;**

DECOMPOSITIONMASTERIP < *masterip-options* > **;**

MASTERIP < *masterip-options* > **;**

For mixed integer linear programming problems, the DECOMPMASTERIP 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 15.5 summarizes the options available in the DECOMPMASTERIP 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 321 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 15.5.

Table 15.5 Options in the DECOMPMASTERIP Statement

Description	*masterip-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 difference allowed 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=	NONE
Specifies the maximum number of nodes to process	MAXNODES=†	
Specifies the maximum number of solutions to find	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 to use in determining 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 a target objective value	TARGET=	
Specifies whether time units are CPU time or real time	TIMETYPE=	
Heuristics Option		
Specifies the primal heuristics level	HEURISTICS=	

Table 15.5 (continued)

Description	*masterip-option*	Different Default
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 iterations to perform on each variable for 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 multicommodity network flow cut level	CUTMULTICOMMODITY=	
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 15.5 but are not described in the MILP solver sections. These options are specific to the DECOMPMASTERIP statement.

NTHREADS=*number*

NUMTHREADS=*number*

specifies the number of threads to use in the master integer solver (if the selected solver method supports multithreading). The value of the NTHREADS= option in the main solver statement serves as the overall capacity for the number of active threads that can run at one time. By default, the number of master integer threads is the value of the NTHREADS= option in the main solver statement.

PRIMALIN=FALSE | TRUE

PIN=FALSE | TRUE

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, a warm start can reduce solution time significantly. You can specify the following values:

FALSE ignores the previous solution.

TRUE starts from the previous solution.

By default, PRIMALIN=TRUE.

DECOMPSUBPROB Statement

DECOMPSUBPROB < *subprob-options* > **;**

DECOMPOSITIONSUBPROB < *subprob-options* > **;**

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

The DECOMPSUBPROB statement controls the subproblem.

Table 15.6 summarizes the options available for the decomposition algorithm in the DECOMPSUBPROB statement when the chosen subproblem algorithm is an LP algorithm. (As the table indicates, you can specify the PRINTLEVEL= option only in the OPTLP procedure.) For descriptions of these options, see the section "LP Solver Options" on page 257 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 15.6.

Table 15.6 Options in the DECOMPSUBPROB Statement Used with an LP Algorithm

Description	*subprob-option*	Different Default
Algorithm Option		
Specifies the subproblem algorithm	ALGORITHM=	†
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†
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=	NONE
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=	

Table 15.6 (continued)

Description	*subprob-option*	Different Default
Simplex Algorithm Options		
Specifies the type of initial basis	BASIS=	WARMSTART†
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 a duality gap	DUALITYGAP=	

† When you specify METHOD=USER or METHOD=AUTO in the DECOMP statement, ALGORITHM=PS, PRESOLVER=NONE, and BASIS=WARMSTART by default. The reason for these defaults is 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 at each iteration. When METHOD=NETWORK, ALGORITHM=NETWORKPURE by default because each subproblem is a pure network, causing the specialized pure network solver to usually be the most efficient choice. When METHOD=CONCOMP, ALGORITHM=DS by default because dual simplex generally has the best performance on linear programs and, in this case, one outer iteration is sufficient (that is, warm starts are not required).

Table 15.7 summarizes the options available in the DECOMPSUBPROB statement when the chosen subproblem algorithm 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 321 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 15.7 Options in the DECOMPSUBPROB Statement Used with a MILP Algorithm

Description	*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 an absolute objective gap	ABSOBJGAP=	
Emphasizes feasibility or optimality	EMPHASIS=	
Specifies the maximum violation on variables and constraints	FEASTOL=	1E–7
Specifies the maximum difference allowed 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 process	MAXNODES=	
Specifies the maximum number of solutions to find	MAXSOLS=	
Specifies the time limit for the optimization process	MAXTIME=	

Table 15.7 (continued)

Description	*subprob-option*	Different Default
Specifies the number of threads to use in the subproblem solver	NTHREADS=	
Specifies the tolerance to use in determining 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=	
Specifies the scale of the problem matrix	SCALE=	
Specifies the stopping criterion based on a 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=	
Specifies the initial seed for the random number generator	SEED=	
Specifies the number of iterations to perform for strong branching	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 multicommodity network flow cut level	CUTMULTICOMMODITY=	
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 15.6 and Table 15.7, are specific to the DECOMPSUBPROB statement and are not described in the LP or MILP solver sections:

ALGORITHM=*string*
SOLVER=*string*
SOL=*string*
specifies the algorithm to use for the subproblem solves. You can specify the following values (the valid abbreviated value for each *string* follows the vertical bar):

PRIMAL | PS uses the primal simplex algorithm.

DUAL | DS uses the dual simplex algorithm.

NETWORK | NS uses the network simplex algorithm.

NETWORKPURE | NSPURE uses the network simplex algorithm for pure networks.

INTERIORPOINT | IP uses the interior point algorithm.

MILP uses the mixed integer linear solver.

By default, ALGORITHM=NETWORKPURE if METHOD=NETWORK, ALGORITHM=MILP for mixed integer linear programming subproblems, and ALGORITHM=PS for linear programming subproblems.

INITPRESOLVER=AUTOMATIC | NONE | BASIC | MODERATE | AGGRESSIVE
INITPRESOL=AUTOMATIC | NONE | BASIC | MODERATE | AGGRESSIVE
specifies, for the first subproblem solve only, the presolve level. You can specify the following values:

AUTOMATIC applies the default level of presolve processing.

NONE disables the presolver.

BASIC performs minimal presolve processing.

MODERATE applies a higher level of presolve processing.

AGGRESSIVE applies the highest level of presolve processing.

By default, INITPRESOLVER=AUTOMATIC.

NTHREADS=*number*
NUMTHREADS=*number*
specifies the number of threads to use in the subproblem solver (if the selected solver method supports multithreading). The value of the NTHREADS= option in the main solver statement serves as the overall capacity for the number of active threads that can run at one time. By default, the number of subproblem threads is $t = \max(1, \lfloor p/n \rfloor)$, where p is the value of the NTHREADS= option in the main solver statement; $n = \min(p, d)$, which is the number of blocks being processed simultaneously; and d is the number of block threads.

PRIMALIN=FALSE | TRUE
PIN=FALSE | TRUE
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, a warm start can reduce solution time significantly. You can specify the following values:

FALSE ignores the previous solution.

TRUE starts from the previous solution.

By default, PRIMALIN=TRUE.

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 708 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 706 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 DECOMPMASTER statement), one subproblem for each block (controlled by options in the DECOMPSUBPROB statement) and, for MILPs, the integer version of the master problem (controlled by options in the DECOMPMASTERIP 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 630. 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 704, the subproblem often has a block-angular structure that enables the solver to process each block independently.

You can control the number of threads that are used by specifying the NTHREADS= option in the main solver statement. In addition, you can use the NTHREADS= option in each subcomponent statement to specify the number of threads to use for that solver. This is discussed in detail in the following sections:

- NTHREADS= option in the DECOMP statement on page 716
- NTHREADS= option in the DECOMPMASTER statement on page 718
- NTHREADS= option in the DECOMPMASTERIP statement on page 720
- NTHREADS= option in the DECOMPSUBPROB statement on page 724

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 704) that have equivalent feasible regions (and optima) when they are projected. Algebraically, this means that

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

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:

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

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

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	f
Solution Status	Optimal
Objective Value	5
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	0
Bound Infeasibility	0
Integer Infeasibility	0
Best Bound	5
Nodes	1
Iterations	2
Presolve Time	0.00
Solution Time	0.02

	x	
	1	**2**
1	1	0
2	0	1
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 number of block threads has been reduced to 1 threads.
NOTE: The problem has a decomposable structure with 2 blocks. The largest block covers 25% of
      the constraints in the problem.
NOTE: The decomposition subproblems cover 6 (100%) variables and 2 (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
         .       6.0000       5.0000        5.0000   16.67%   16.67%    0    0
         1       6.0000       5.0000        5.0000   16.67%   16.67%    0    0
         2       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 0.02 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 15.5, which shows the bin packing problem, and Example 15.7, 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 15.9, 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 704, 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 731 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 value 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 value 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 (DECOMPMASTER, DECOMPMASTERIP, and DECOMPSUBPROB). By default, the individual component log levels are all disabled.

Examples: Decomposition Algorithm

Example 15.1: Multicommodity Flow Problem and METHOD=NETWORK

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}.

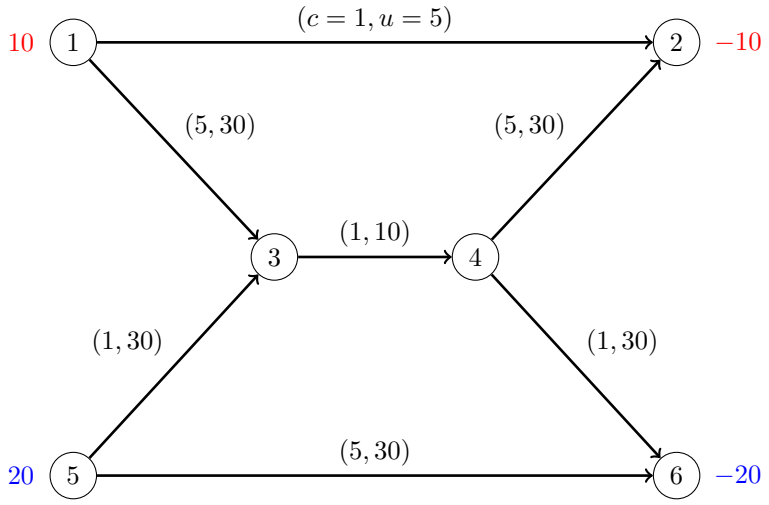
The problem can be modeled as a linear programming problem as follows:

$$\begin{aligned}
&\text{minimize} && \sum_{(i,j)\in A}\sum_{k\in K} c_{ij}^k x_{ij}^k && && \\
&\text{subject to} && \sum_{k\in K} x_{ij}^k \le u_{ij} && (i,j)\in A && \text{(Capacity)}\\
& && \sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = b_i^k && i\in N,\ k\in K && \text{(Balance)}\\
& && x_{ij}^k \ge 0 && (i,j)\in A,\ k\in K &&
\end{aligned}$$

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

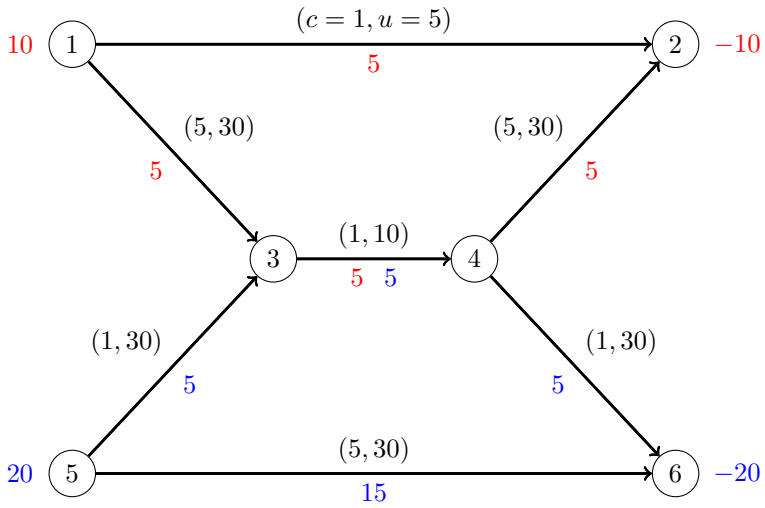
The OPTMODEL Procedure

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	3
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 number of block threads has been reduced to 2 threads.
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%) 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        1.0000  1.00e+00       0.0       0.0
          2        0.0000        0.0000     0.00%       0.0       0.0
NOTE: Starting phase 2.
          3      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:

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

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

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

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

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
   decompsubprob=(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
Objective Value	563
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	1.110223E-16
Bound Infeasibility	0
Integer Infeasibility	1.110223E-16
Best Bound	563
Nodes	931
Iterations	2834
Presolve Time	0.00
Solution Time	7.68

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalProfit
Solution Status	Optimal
Objective Value	563
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	4.440892E-16
Bound Infeasibility	4.440892E-16
Integer Infeasibility	4.440892E-16
Best Bound	563
Nodes	5
Iterations	94
Presolve Time	0.00
Solution Time	0.46

The iteration log for both decompositions is shown in Output 15.2.2. This example is interesting because it shows the trade-off 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 trade-off, 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 5 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.
WARNING: 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 3.125%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 192 (100%) variables and 24 (75%) 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
         .      574.0000     559.0836      552.0000    2.60%     3.83%    0    0
         5      568.6281     568.6281      562.0000    0.00%     1.17%    0    0
NOTE: Starting branch and bound.
         Node  Active   Sols         Best         Best      Gap    CPU   Real
                                  Integer        Bound            Time   Time
            0       1      8     562.0000     568.6281    1.17%      0      0
          930       0      9     563.0000     563.0000    0.00%      3      7
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 7.68 seconds.
NOTE: Optimal.
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 5 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 problem has a decomposable structure with 8 blocks. The largest block covers 3.125%
      of the constraints in the problem.
```

Output 15.2.2 *continued*

```
NOTE: The decomposition subproblems cover 192 (100%) variables and 8 (25%) 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
.	820.0000	474.0000	474.0000	42.20%	42.20%	0	0
1	820.0000	474.0000	474.0000	42.20%	42.20%	0	0
3	755.0000	474.0000	474.0000	37.22%	37.22%	0	0
6	755.0000	553.0000	553.0000	26.75%	26.75%	0	0
7	670.3333	553.0000	553.0000	17.50%	17.50%	0	0
8	638.8824	553.0000	553.0000	13.44%	13.44%	0	0
9	599.0000	553.0000	553.0000	7.68%	7.68%	0	0
.	599.0000	553.0000	553.0000	7.68%	7.68%	0	0
10	599.0000	553.0000	553.0000	7.68%	7.68%	0	0
12	596.1429	553.0000	553.0000	7.24%	7.24%	0	0
13	583.0000	553.0000	553.0000	5.15%	5.15%	0	0
15	583.0000	563.0000	563.0000	3.43%	3.43%	0	0
16	572.9333	563.0000	563.0000	1.73%	1.73%	0	0
17	571.2857	563.0000	563.0000	1.45%	1.45%	0	0
18	570.0000	563.5000	563.0000	1.14%	1.23%	0	0
.	570.0000	564.0000	563.0000	1.05%	1.23%	0	0
20	570.0000	564.0000	563.0000	1.05%	1.23%	0	0
21	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 0.46 seconds.
NOTE: Optimal.
NOTE: Objective = 563.
```

Example 15.3: Block-Diagonal Structure and METHOD=CONCOMP

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

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
   nthreads = 4
   data     = mpsdata;
run;
```

The solution summary is shown in Output 15.3.1.

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	5.932461E-13
Bound Infeasibility	3.010925E-13
Integer Infeasibility	1.001421E-13
Best Bound	120
Nodes	289
Iterations	38546
Presolve Time	0.02
Solution Time	1.82

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 parallel Branch and Cut algorithm is used.
NOTE: The Branch and Cut algorithm is using up to 4 threads.
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      2    161.0000000              0    161.0       0
            0        1      2    161.0000000     91.4479396   76.06%       0
            0        1      2    161.0000000    111.7932692   44.02%       1
            0        1      2    161.0000000    111.7932692   44.02%       1
NOTE: The MILP presolver is applied again.
            0        1      3    128.0000000    111.7932692   14.50%       1
            0        1      4    127.0000000    112.1093044   13.28%       1
NOTE: The MILP solver added 1 cuts with 5 cut coefficients at the root.
           59       37      5    120.0000000    114.3520000    4.94%       1
          288        0      5    120.0000000    120.0000000    0.00%       1
NOTE: Optimal.
NOTE: Objective = 120.
NOTE: There were 6215 observations read from the data set WORK.MPSDATA.
```

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.

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

The solution summary is displayed in Output 15.3.3.

Output 15.3.3 Solution Summary

The OPTMILP Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	R0001298
Solution Status	Optimal
Objective Value	120
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	3.996803E-15
Bound Infeasibility	2.664535E-15
Integer Infeasibility	4.440892E-16
Best Bound	120
Nodes	1
Iterations	1
Presolve Time	0.01
Solution Time	0.75

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 PROC OPTMILP statement, the blocks are processed simultaneously on four threads.

Output 15.3.4 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 Decomposition algorithm is used.
NOTE: The Decomposition algorithm is executing in single-machine mode.
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%) variables and 1260 (100%) 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 161 to provide initial columns.
NOTE: Using a starting solution with objective value 231 to provide initial columns.
NOTE: The initial column pool using the starting solution contains 8 columns.
NOTE: The subproblem solver for 4 blocks at iteration 0 is starting.
NOTE: The subproblem solver for 4 blocks used 0.70 (cpu: 1.88, max: 0.70) seconds.
NOTE: The initial column pool after generating initial variables contains 12 columns.
      Iter         Best       Master         Best       LP       IP  CPU Real
                  Bound    Objective      Integer      Gap      Gap Time Time
NOTE: The master solver at iteration 1 is starting.
NOTE: The master solver used 0.00 (cpu: 0.00) seconds and 0 iterations.
         1     120.0000     120.0000     120.0000    0.00%    0.00%    1    0
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 0.73 seconds.
NOTE: Optimal.
NOTE: Objective = 120.
NOTE: There were 6215 observations read from the data set WORK.MPSDATA.
```

In this case, the solver finds that after the 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=MODERATE 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-Angular Structure and METHOD=AUTO

This example demonstrates how you can use the METHOD=AUTO option in the DECOMP statement to run the decomposition algorithm in single-machine 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 attempt to solve the problem by using standard methods and a 60-second time limit.

```
proc optmilp
   nthreads = 4
   maxtime  = 60
   data     = mpsdata;
run;
```

The solution summary is shown in Output 15.4.1.

Output 15.4.1 Solution Summary

The OPTMILP Procedure

Solution Summary	
Solver	MILP
Algorithm	Branch and Cut
Objective Function	Total_Profit
Solution Status	Time Limit Reached
Objective Value	6151.1464478
Relative Gap	0.1506142534
Absolute Gap	1090.7297814
Primal Infeasibility	0
Bound Infeasibility	0
Integer Infeasibility	0
Best Bound	7241.8762293
Nodes	1
Iterations	44550
Presolve Time	0.84
Solution Time	78.47

The iteration log, which contains the problem statistics and the progress of the solution, is shown in Output 15.4.2.

Output 15.4.2 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 734 constraints.
NOTE: The MILP presolver removed 17616 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 52638 variables, 3215 constraints, and 131250 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The parallel Branch and Cut algorithm is used.
NOTE: The Branch and Cut algorithm is using up to 4 threads.
          Node   Active   Sols    BestInteger      BestBound      Gap    Time
             0        1      1   6151.1464478   8590.4503506   28.40%       1
             0        1      1   6151.1464478   7342.1209241   16.22%       1
             0        1      1   6151.1464478   7332.7481163   16.11%       7
             0        1      1   6151.1464478   7298.5629031   15.72%      15
             0        1      1   6151.1464478   7270.5758647   15.40%      32
             0        1      1   6151.1464478   7241.8762293   15.06%      53
NOTE: The MILP solver added 8806 cuts with 195585 cut coefficients at the root.
NOTE: Real time limit reached.
NOTE: Objective of the best integer solution found = 6151.1464478.
NOTE: There were 159467 observations read from the data set WORK.MPSDATA.
```

Standard MILP techniques struggle to solve the problem 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
   nthreads  = 4
   data      = mpsdata;
   decomp
      method = auto;
run;
```

The solution summary is displayed in Output 15.4.3.

Output 15.4.3 Solution Summary

The OPTMILP Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	Total_Profit
Solution Status	Optimal within Relative Gap
Objective Value	6972.3309349
Relative Gap	3.1210814E-9
Absolute Gap	0.0000217612
Primal Infeasibility	3.419487E-14
Bound Infeasibility	8.7366867E-8
Integer Infeasibility	8.548717E-15
Best Bound	6972.3309567
Nodes	1
Iterations	6
Presolve Time	0.89
Solution Time	39.86

The iteration log, which contains the problem statistics and the progress of the solution, is shown in Output 15.4.4.

Output 15.4.4 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 734 constraints.
NOTE: The MILP presolver removed 17616 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 52638 variables, 3215 constraints, and 131250 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 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
      0.2488% of the constraints in the problem.
NOTE: The decomposition subproblems cover 52638 (100%) variables and 3207 (99.75%) 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
         .    7963.9763    6457.0911    6457.0911   18.92%   18.92%   16   12
         2    7024.6025    6457.0911    6457.0911    8.08%    8.08%   34   18
         5    7020.4193    6961.7914    6961.7914    0.84%    0.84%   79   37
         6    6972.3310    6972.3309    6972.3309    0.00%    0.00%   82   38
         Node  Active   Sols          Best          Best      Gap    CPU   Real
                                    Integer         Bound             Time   Time
            0       0      5    6972.3309    6972.3310    0.00%     82     38
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 38.84 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 6972.3309349.
NOTE: There were 159467 observations read from the data set WORK.MPSDATA.
```

As stated in the log, the automated method attempts to find a balanced block-angular form that contains four blocks (the default is the same as the value of 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 more than 99% subproblem coverage.

Example 15.5: 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:

$$\begin{array}{lrll}
\text{minimize} & \displaystyle\sum_{b \in B} y_b & & \\
\text{subject to} & \displaystyle\sum_{b \in B} x_{pb} = 1 & p \in P & \text{(Assignment)} \\
& \displaystyle\sum_{p \in P} s_p x_{pb} \leq C y_b & b \in B & \text{(Capacity)} \\
& x_{pb} \in \{0, 1\} & p \in P,\ b \in B & \\
& y_b \in \{0, 1\} & b \in B &
\end{array}$$

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 727, 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 recorded 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 PROC 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 15.5.1.

Output 15.5.1 Solution Summary

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	Objective
Solution Status	Optimal
Objective Value	11
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	1.332268E-15
Bound Infeasibility	1.332268E-15
Integer Infeasibility	1.332268E-15
Best Bound	11
Nodes	1
Iterations	23
Presolve Time	0.01
Solution Time	0.09

The iteration log is displayed in Output 15.5.2.

Output 15.5.2 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 384 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 number of block threads has been reduced to 1 threads.
NOTE: The problem has a decomposable structure with 24 blocks. The largest block covers 2.083%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 600 (100%) variables and 24 (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
         .       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
         .       3.0000      11.0000      11.0000  266.67%  266.67%    0    0
        20       3.0000      11.0000      11.0000  266.67%  266.67%    0    0
        23      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 0.08 seconds.
NOTE: Optimal.
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
VT	1.41
WM	1.25
WM2	1.23
bin	**3.89**

bin=2

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

bin=3

product	size
Miami	1.22
VA2	2.48
bin	**3.70**

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
Vermont	1.28
bin	**3.89**

bin=9

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

bin=10

product	size
Clemson	1.36
FSU	2.56
bin	**3.92**

bin=11

product	size
Duke	2.76
Wake	1.61
bin	**4.37**
	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 15.7.

Example 15.6: 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 that is 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 $s \in S$. You can model the problem as a mixed integer linear programming problem as follows:

$$\begin{array}{lllll} \text{maximize} & \displaystyle\sum_{i \in I} p_i x_i & & & \\ \text{subject to} & \displaystyle\sum_{i \in L_s} w_i x_i \leq C & & s \in S & \text{(CapacityCon)} \\ & x_i \in \{0,1\} & & i \in I & \end{array}$$

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 online and offline:

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

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, you can write the original constraints 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. You can now model the problem in block-angular form as follows:

$$\begin{array}{lllll} \text{maximize} & \displaystyle\sum_{i \in I} p_i x_i^{m_i} & & & \\ \text{subject to} & x_i^b = x_i^{m_i} & i \in I,\ b \in B_i \setminus \{m_i\} & & \text{(LinkDupVarsCon)} \\ & \displaystyle\sum_{i \in L_s} w_i x_i^b \leq C & b \in B,\ s \in S_b & & \text{(CapacityCon)} \\ & x_i^b \in \{0,1\} & i \in I,\ b \in B_i & & \end{array}$$

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:

$$
\begin{array}{lrrrrrrrrrrrrrrrl}
\text{maximize} & 6x_1^1 & + & 8x_2^1 & & & + & 5x_3^2 & + & 9x_4^2 & & & & + & 8x_5^3 & \\
\text{subject to} & & & x_2^1 & - & x_2^2 & & & & & & & & & & = 0 \\
 & & & & & & & x_3^2 & & & - & x_3^3 & & & & = 0 \\
 & & & & & & & & & x_4^2 & & & - & x_4^3 & & = 0 \\
 & 8x_1^1 & & & & & & & & & & & & & & \leq 10 \\
 & 8x_1^1 & + & 5x_2^1 & & & & & & & & & & & & \leq 10 \\
 & & & & & 5x_2^2 & + & 3x_3^2 & & & & & & & & \leq 10 \\
 & & & & & 5x_2^2 & + & 3x_3^2 & + & 4x_4^2 & & & & & & \leq 10 \\
 & & & & & & & & & & & 3x_3^3 & + & 4x_4^3 & + & 3x_5^3 & \leq 10 \\
 & & & & & & & & & & & & & & & x_i^b & \in \{0,1\}\ i \in I, b \in B_i
\end{array}
$$

To see how to apply Lagrangian decomposition in PROC OPTMODEL, consider the data set TaskData from Caprara, Furini, and Malaguti (2010), which consists of $|I| = 2{,}916$ tasks:

```
data TaskData;
   input profit weight start end;
   datalines;
99 92 1 9
56 30 1 3
39 73 1 20
86 76 1 9
...
24 94 768 769
95 40 768 769
66 17 768 769
18 48 768 769
97 23 768 769
;
```

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 with milp / maxtime=200 logfreq=10000;
   quit;
%mend ResourceAllocation_Direct;

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

The problem summary and solution summary are displayed in Output 15.6.1.

Output 15.6.1 Problem Summary and Solution Summary

The OPTMODEL Procedure

Problem Summary	
Objective Sense	Maximization
Objective Function	TotalProfit
Objective Type	Linear
Number of Variables	2916
Bounded Above	0
Bounded Below	0
Bounded Below and Above	2916
Free	0
Fixed	0
Binary	2916
Integer	0
Number of Constraints	768
Linear LE (<=)	768
Linear EQ (=)	0
Linear GE (>=)	0
Linear Range	0
Constraint Coefficients	23236

Output 15.6.1 *continued*

Solution Summary	
Solver	MILP
Algorithm	Branch and Cut
Objective Function	TotalProfit
Solution Status	Time Limit Reached
Objective Value	40980
Relative Gap	0.0007331077
Absolute Gap	30.064793335
Primal Infeasibility	0
Bound Infeasibility	0
Integer Infeasibility	0
Best Bound	41010.064793
Nodes	129193
Iterations	2743208
Presolve Time	0.17
Solution Time	199.55

The iteration log, which contains the problem statistics, the progress of the solution, and the best integer feasible solution found, is shown in Output 15.6.2.

Output 15.6.2 Log

```
NOTE: There were 2916 observations read from the data set WORK.TASKDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 2916 variables (0 free, 0 fixed).
NOTE: The problem has 2916 binary and 0 integer variables.
NOTE: The problem has 768 linear constraints (768 LE, 0 EQ, 0 GE, 0 range).
NOTE: The problem has 23236 linear constraint coefficients.
NOTE: The problem has 0 nonlinear constraints (0 LE, 0 EQ, 0 GE, 0 range).
NOTE: The remaining solution time after problem generation and solver initialization is 199.52
      seconds.
NOTE: The MILP presolver value AUTOMATIC is applied.
NOTE: The MILP presolver removed 1021 variables and 126 constraints.
NOTE: The MILP presolver removed 12544 constraint coefficients.
NOTE: The MILP presolver modified 987 constraint coefficients.
NOTE: The presolved problem has 1895 variables, 642 constraints, and 10692 constraint
      coefficients.
NOTE: The MILP solver is called.
NOTE: The parallel Branch and Cut algorithm is used.
NOTE: The Branch and Cut algorithm is using up to 4 threads.
          Node   Active   Sols    BestInteger      BestBound      Gap    Time
             0        1      3  33939.0000000         109181   68.91%       0
             0        1      3  33939.0000000  45862.9249030   26.00%       0
             0        1      7  39381.0000000  43471.6068311    9.41%       0
             0        1      7  39381.0000000  42707.3723942    7.79%       0
             0        1      7  39381.0000000  42240.0006312    6.77%       1
             0        1      7  39381.0000000  41971.7026684    6.17%       1
             0        1      7  39381.0000000  41737.8969132    5.65%       1
             0        1      7  39381.0000000  41603.2939984    5.34%       1
             0        1      7  39381.0000000  41478.6012550    5.06%       2
             0        1      7  39381.0000000  41401.0205505    4.88%       2
             0        1     10  40529.0000000  41352.9949434    1.99%       2
             0        1     10  40529.0000000  41307.5535959    1.88%       2
             0        1     10  40529.0000000  41278.9211791    1.82%       3
             0        1     10  40529.0000000  41248.9510126    1.75%       3
             0        1     10  40529.0000000  41217.0410780    1.67%       3
             0        1     10  40529.0000000  41194.1989774    1.61%       3
             0        1     10  40529.0000000  41174.5412859    1.57%       3
             0        1     10  40529.0000000  41161.0054242    1.54%       4
             0        1     10  40529.0000000  41144.3324825    1.50%       4
             0        1     10  40529.0000000  41130.2474275    1.46%       4
             0        1     10  40529.0000000  41122.9371818    1.44%       4
             0        1     10  40529.0000000  41113.9694765    1.42%       4
             0        1     10  40529.0000000  41104.8760183    1.40%       5
NOTE: The MILP solver added 814 cuts with 10849 cut coefficients at the root.
           796       42     11  40817.0000000  41070.6514203    0.62%       7
          1679      793     12  40852.0000000  41059.5824557    0.51%       8
          4286     2985     13  40868.0000000  41052.1191058    0.45%      11
          4895     3291     14  40900.0000000  41050.5324921    0.37%      11
          4962     3007     15  40933.0000000  41050.5324921    0.29%      11
         43207    27119     16  40959.0000000  41020.7899316    0.15%      59
```

Output 15.6.2 *continued*

```
         76301    36801     17  40968.0000000  41011.1154418    0.11%     109
        129192    17849     18  40980.0000000  41010.0647933    0.07%     199
NOTE: Real time limit reached.
NOTE: Objective of the best integer solution found = 40980.
```

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 that are 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, use the following PROC OPTMODEL statements to define the block-angular formulation and solve the problem by using the decomposition algorithm, the PRESOLVER=BASIC option, and **block_size=20**. 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=20);
```

The problem summary and solution summary are displayed in Output 15.6.3. Compared to the original formulation, the numbers of variables and constraints are increased by the number of duplicate variables.

Output 15.6.3 Problem Summary and Solution Summary

The OPTMODEL Procedure

Problem Summary	
Objective Sense	Maximization
Objective Function	TotalProfit
Objective Type	Linear
Number of Variables	3924
Bounded Above	0
Bounded Below	0
Bounded Below and Above	3924
Free	0
Fixed	0
Binary	3924
Integer	0
Number of Constraints	1776
Linear LE (<=)	768
Linear EQ (=)	1008
Linear GE (>=)	0
Linear Range	0
Constraint Coefficients	25252

Output 15.6.3 *continued*

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalProfit
Solution Status	Optimal within Relative Gap
Objective Value	40982
Relative Gap	0.0000244109
Absolute Gap	1.0004305891
Primal Infeasibility	2.664535E-15
Bound Infeasibility	4.218847E-15
Integer Infeasibility	4.218847E-15
Best Bound	40983.000431
Nodes	23
Iterations	308
Presolve Time	0.04
Solution Time	129.06

The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.6.4.

Output 15.6.4 Log

```
NOTE: There were 2916 observations read from the data set WORK.TASKDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 3924 variables (0 free, 0 fixed).
NOTE: The problem has 3924 binary and 0 integer variables.
NOTE: The problem has 1776 linear constraints (768 LE, 1008 EQ, 0 GE, 0 range).
NOTE: The problem has 25252 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 4 variables and 0 constraints.
NOTE: The MILP presolver removed 23 constraint coefficients.
NOTE: The MILP presolver modified 7297 constraint coefficients.
NOTE: The presolved problem has 3920 variables, 1776 constraints, and 25229 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 39 blocks. The largest block covers 1.126%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 3920 (100%) variables and 768 (43.24%) 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
         .   44109.0000   37886.0000   37886.0000    14.11%    14.11%    1    0
         5   44109.0000   37921.0000   37921.0000    14.03%    14.03%    3    1
         8   44109.0000   37975.0000   37975.0000    13.91%    13.91%    4    2
         .   44109.0000   37975.0000   37975.0000    13.91%    13.91%    4    2
        10   44109.0000   37975.0000   37975.0000    13.91%    13.91%    5    2
        12   44109.0000   38576.0000   38576.0000    12.54%    12.54%    5    2
        16   44109.0000   38857.0000   38857.0000    11.91%    11.91%    7    3
        18   44109.0000   39202.0000   39202.0000    11.12%    11.12%    8    4
         .   44109.0000   39202.0000   39202.0000    11.12%    11.12%   10    5
        20   44109.0000   39202.0000   39202.0000    11.12%    11.12%   10    5
        22   44109.0000   39590.0000   39590.0000    10.25%    10.25%   12    6
        28   44109.0000   40448.0000   40448.0000     8.30%     8.30%   18    8
        30   44109.0000   40448.0000   40699.0000     8.30%     7.73%   20    9
         .   44109.0000   40739.9167   40699.0000     7.64%     7.73%   38   16
        40   44109.0000   40739.9167   40699.0000     7.64%     7.73%   41   17
        41   43325.6713   40740.7619   40699.0000     5.97%     6.06%   44   18
        42   43129.3790   40758.2083   40699.0000     5.50%     5.64%   47   19
        43   43022.8997   40779.1667   40699.0000     5.22%     5.40%   49   19
        47   43022.8997   40881.8333   40823.0000     4.98%     5.11%   55   21
        50   43022.8997   40886.9583   40873.0000     4.96%     5.00%   57   23
        52   43022.8997   40892.5833   40877.0000     4.95%     4.99%   60   24
        58   43022.8997   40938.5833   40923.0000     4.84%     4.88%   72   28
        60   43022.8997   40938.5833   40923.0000     4.84%     4.88%   75   30
        61   41621.8616   40938.5833   40923.0000     1.64%     1.68%   77   31
        62   41450.8337   40938.5833   40923.0000     1.24%     1.27%   79   31
```

Output 15.6.4 *continued*

```
        63   41339.6881   40938.5833   40923.0000     0.97%     1.01%   81    32
        70   41339.6881   40964.0833   40923.0000     0.91%     1.01%   89    35
         .   41339.6881   40974.2500   40943.0000     0.88%     0.96%   94    37
        80   41339.6881   40974.2500   40943.0000     0.88%     0.96%   95    37
        81   41049.2500   40978.5833   40943.0000     0.17%     0.26%   97    38
        82   41009.0833   40978.5833   40943.0000     0.07%     0.16%   97    38
        90   41009.0833   40978.5833   40943.0000     0.07%     0.16%   99    40
       100   40997.5834   40997.5833   40943.0000     0.00%     0.13%  101    41
         .   40997.5834   40997.5833   40982.0000     0.00%     0.04%  101    41
NOTE: Starting branch and bound.
         Node  Active   Sols         Best         Best      Gap    CPU   Real
                                  Integer        Bound             Time   Time
            0       1     66   40982.0000   40997.5834    0.04%    101     41
           10       4     66   40982.0000   40987.8333    0.01%    233     90
           20       2     66   40982.0000   40987.5833    0.01%    317    121
           22       0     66   40982.0000   40983.0004    0.00%    337    128
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 128.95 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 40982.
```

The Trade-Off between Coverage and Subproblem Difficulty

The reformulation of this resource allocation problem provides a nice example of the potential trade-offs 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 the current 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 20. The following statement calls the decomposition algorithm and uses a block size of 80:

```
%ResourceAllocation_Decomp(task_data=TaskData, capacity=100, block_size=80);
```

The solution summary is displayed in Output 15.6.5.

Output 15.6.5 Solution Summary

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalProfit
Solution Status	Optimal
Objective Value	40982
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	0
Bound Infeasibility	0
Integer Infeasibility	4.440892E-16
Best Bound	40982
Nodes	1
Iterations	43
Presolve Time	0.04
Solution Time	55.36

The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.6.6.

This version of the model provides a stronger initial bound and solves to optimality in the root node.

Output 15.6.6 Log

```
NOTE: There were 2916 observations read from the data set WORK.TASKDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 3151 variables (0 free, 0 fixed).
NOTE: The problem has 3151 binary and 0 integer variables.
NOTE: The problem has 1003 linear constraints (768 LE, 235 EQ, 0 GE, 0 range).
NOTE: The problem has 23706 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 5 variables and 0 constraints.
NOTE: The MILP presolver removed 29 constraint coefficients.
NOTE: The MILP presolver modified 7295 constraint coefficients.
NOTE: The presolved problem has 3146 variables, 1003 constraints, and 23677 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 10 blocks. The largest block covers 7.976%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 3146 (100%) variables and 768 (76.57%) 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
         .   41762.0011   37733.0000   37733.0000    9.65%    9.65%    4    1
         6   41762.0011   37949.0000   37949.0000    9.13%    9.13%   17    6
         .   41762.0011   39051.0000   39051.0000    6.49%    6.49%   23    9
        10   41762.0011   39051.0000   39051.0000    6.49%    6.49%   26   10
        13   41762.0011   40689.0000   40689.0000    2.57%    2.57%   35   13
        18   41762.0011   40723.0000   40723.0000    2.49%    2.49%   51   19
        20   41762.0011   40723.0000   40742.0000    2.49%    2.44%   54   21
        21   41762.0011   40796.0000   40796.0000    2.31%    2.31%   59   23
        25   41762.0011   40910.0000   40910.0000    2.04%    2.04%   75   29
        26   41762.0011   40956.0000   40956.0000    1.93%    1.93%   81   31
        28   41566.7275   40956.0000   40956.0000    1.47%    1.47%   90   34
        29   41382.2500   40956.0000   40956.0000    1.03%    1.03%   96   36
        30   41279.9383   40956.0000   40956.0000    0.78%    0.78%  100   37
        37   41279.9383   40961.0000   40961.0000    0.77%    0.77%  128   47
         .   41279.9383   40961.0000   40961.0000    0.77%    0.77%  130   49
        40   41279.9383   40961.0000   40961.0000    0.77%    0.77%  131   50
        41   40982.0000   40961.0000   40961.0000    0.05%    0.05%  138   53
        43   40982.0000   40982.0000   40982.0000    0.00%    0.00%  141   55
         Node  Active   Sols         Best         Best      Gap    CPU   Real
                                   Integer        Bound             Time   Time
            0       0     38   40982.0000   40982.0000    0.00%    141     55
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 55.27 seconds.
NOTE: Optimal.
NOTE: Objective = 40982.
```

Example 15.7: 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 $C = \{2, \ldots, |C| + 1\}$ 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:

$$\text{minimize} \quad \sum_{(i,j)\in A} \sum_{k\in V} c_{ij} z_{ijk}$$

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

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 727, 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 PROC OPTMODEL statement creates plot data for the optimal routing:

```
   /* create solution data set */
   create data solution_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]
      function='line' drawspace='datavalue';
quit;
```

The solution summary is displayed in Output 15.7.1.

Output 15.7.1 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	1.364242E-11
Bound Infeasibility	1.364242E-12
Integer Infeasibility	5.107026E-15
Best Bound	603
Nodes	1
Iterations	68
Presolve Time	0.13
Solution Time	46.85

The iteration log is displayed in Output 15.7.2.

Output 15.7.2 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 number of block threads has been reduced to 1 threads.
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%) variables and 4216 (99.5%) 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      20.0000             . 2.00e+01         .    1    1
        20       0.0000       0.5472             . 5.47e-01         .    3    3
        28       0.0000       0.0000             .    0.00%         .    4    4
NOTE: Starting phase 2.
        29     112.0000     918.8571             .  720.41%         .    5    5
        38     144.7353     709.5000             .  390.21%         .    9    9
         .     144.7353     676.5000     771.0000  367.40%  432.70%   10    9
        40     144.7353     676.5000     771.0000  367.40%  432.70%   10    9
        42     144.7353     676.5000     744.0000  367.40%  414.04%   11   10
        45     309.0418     659.3636     744.0000  113.36%  140.74%   13   12
        46     430.9536     658.0000     744.0000   52.68%   72.64%   14   13
        48     438.1162     654.2222     744.0000   49.33%   69.82%   16   14
        53     488.0473     629.6667     744.0000   29.02%   52.44%   21   18
        54     488.2217     629.6667     744.0000   28.97%   52.39%   24   21
        56     499.6802     624.0000     744.0000   24.88%   48.90%   27   23
        57     516.0000     624.0000     744.0000   20.93%   44.19%   30   26
        58     525.3810     615.6667     744.0000   17.18%   41.61%   32   28
        59     540.0000     612.0000     744.0000   13.33%   37.78%   33   29
```

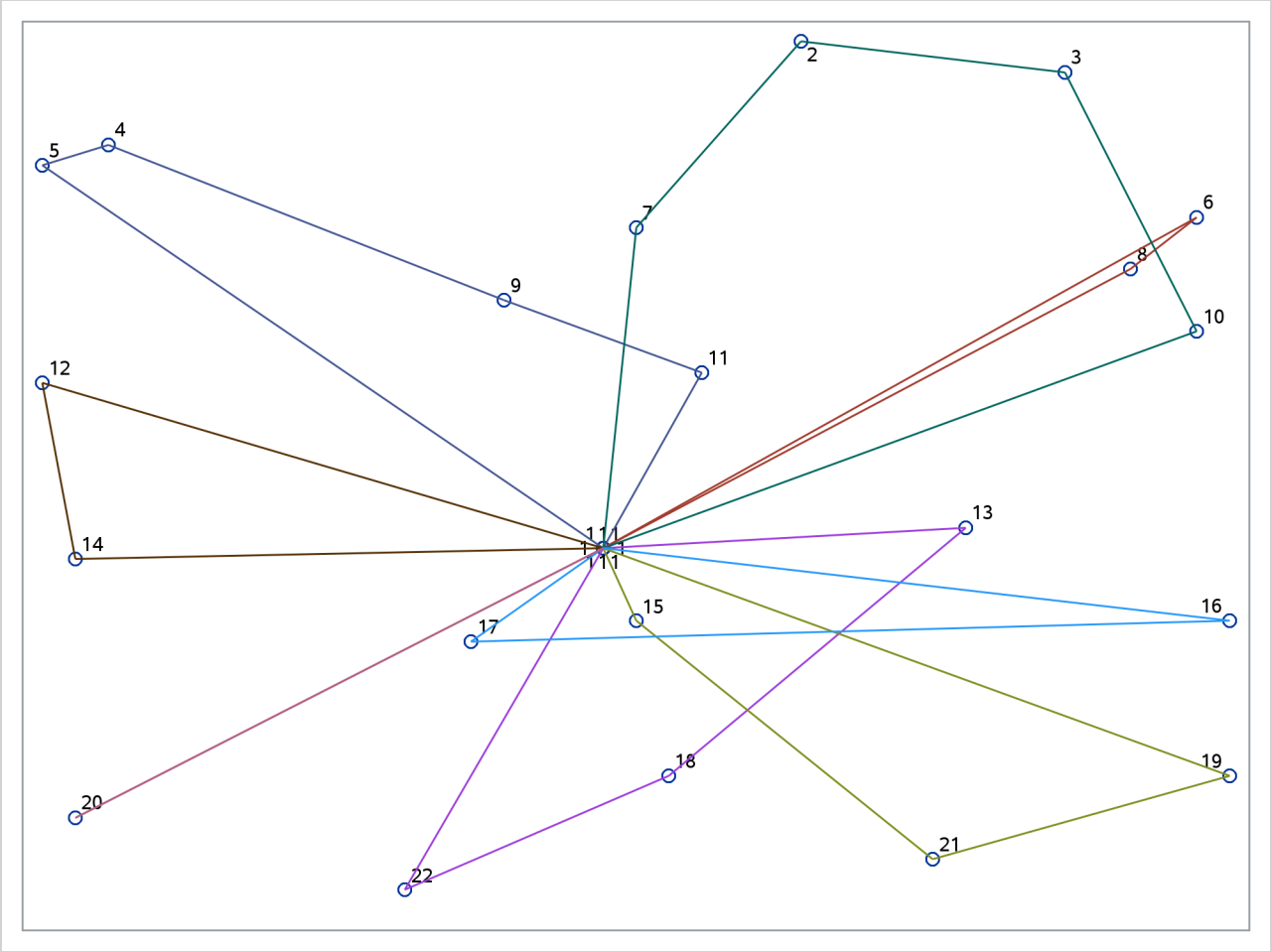
Output 15.7.2 *continued*

```
        60     540.0000     612.0000     744.0000   13.33%   37.78%   34   30
        62     573.3846     605.3846     744.0000    5.58%   29.76%   37   32
        64     588.0000     604.0000     744.0000    2.72%   26.53%   41   35
        65     597.1667     603.8333     744.0000    1.12%   24.59%   45   38
        66     597.1667     603.0000     603.0000    0.98%    0.98%   49   41
        67     600.3333     603.0000     603.0000    0.44%    0.44%   53   43
        68     603.0000     603.0000     603.0000    0.00%    0.00%   56   46
         Node  Active   Sols         Best          Best      Gap     CPU   Real
                                  Integer         Bound             Time   Time
            0       0      3     603.0000     603.0000    0.00%     56     46
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 46.59 seconds.
NOTE: Optimal.
NOTE: Objective = 603.
NOTE: The data set WORK.SOLUTION_DATA has 29 observations and 9 variables.
```

The following call to PROC SGPLOT generates a plot of the optimal routing. The plot is displayed in Figure 15.7.3.

```
proc sgplot data=solution_data noautolegend;
   scatter x=x1 y=y1 / datalabel=i;
   vector x=x2 y=y2 / xorigin=x1 yorigin=y1 group=k noarrowheads;
   xaxis display=none;
   yaxis display=none;
run;
```

Output 15.7.3 Optimal Routing



Example 15.8: ATM Cash Management

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:

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

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:

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

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:

$$\begin{aligned}
\text{minimize} \quad & \sum_{a\in A}\sum_{d\in D}\left(f_{ad}^+ + f_{ad}^-\right) \\
\text{subject to} \quad & c_{ad}^x x_a + c_{ad}^y y_a + \\
& c_{ad}^z x_a y_a + c_{ad}^u u_a + c_{ad} - w_{ad} = f_{ad}^+ - f_{ad}^- && a \in A,\ d \in D \\
& \sum_{a\in A}\left(f_{ad}^+ - f_{ad}^- + w_{ad}\right) \le B_d && d \in D \\
& |\{d \in D \mid (f_{ad}^+ - f_{ad}^-) < 0\}| \le K_a && a \in A \\
& x_a, y_a \in [0,1] && a \in A \\
& u_a \ge 0 && a \in A \\
& f_{ad}^+ \ge 0 && a \in A,\ d \in D \\
& f_{ad}^- \in [0, w_{ad}] && a \in A,\ d \in D
\end{aligned}$$

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 $v_{ad} = 0 \Rightarrow f_{ad}^- \le 0$, by adding the constraint

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

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:

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

The MINLP model can now be written as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{a\in A}\sum_{d\in D}\left(f_{ad}^+ + f_{ad}^-\right) \\
\text{subject to} \quad & c_{ad}^x x_a + c_{ad}^y y_a + \\
& c_{ad}^z x_a y_a + c_{ad}^u u_a + c_{ad} - w_{ad} = f_{ad}^+ - f_{ad}^- && a \in A,\ d \in D \\
& \sum_{a\in A}\left(f_{ad}^+ - f_{ad}^- + w_{ad}\right) \le B_d && d \in D \\
& f_{ad}^- \le w_{ad} v_{ad} && a \in A,\ d \in D \\
& \sum_{d\in D} v_{ad} \le K_a && a \in A \\
& x_a, y_a \in [0,1] && a \in A \\
& u_a \ge 0 && a \in A \\
& f_{ad}^+ \ge 0 && a \in A,\ d \in D \\
& f_{ad}^- \in [0, w_{ad}] && a \in A,\ d \in D \\
& v_{ad} \in \{0,1\} && a \in A,\ d \in D
\end{aligned}$$

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:

$$\begin{aligned}
\text{minimize} \quad & \sum_{a\in A}\sum_{d\in D}\left(f_{ad}^{+}+f_{ad}^{-}\right) \\
\text{subject to} \quad & c_{ad}^{x}\sum_{t\in T}c_t x_{at}+c_{ad}^{y}y_a+ \\
& c_{ad}^{z}\sum_{t\in T}c_t x_{at}y_a+c_{ad}^{u}u_a+c_{ad}-w_{ad}=f_{ad}^{+}-f_{ad}^{-} && a\in A,\ d\in D \\
& \sum_{t\in T}x_{at}=1 && a\in A \\
& \sum_{a\in A}\left(f_{ad}^{+}-f_{ad}^{-}+w_{ad}\right)\le B_d && d\in D \\
& f_{ad}^{-}\le w_{ad}v_{ad} && a\in A,\ d\in D \\
& \sum_{d\in D}v_{ad}\le K_a && a\in A \\
& y_a\in[0,1] && a\in A \\
& u_a\ge 0 && a\in A \\
& f_{ad}^{+}\ge 0 && a\in A,\ d\in D \\
& f_{ad}^{-}\in[0,w_{ad}] && a\in A,\ d\in D \\
& v_{ad}\in\{0,1\} && a\in A,\ d\in D \\
& x_{at}\in\{0,1\} && a\in A,\ t\in T
\end{aligned}$$

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:

$$\begin{array}{lcll}
z_t & = & x_t y & t\in T \\
\sum_{t\in T}x_t & = & 1 & \\
x_t & \in & \{0,1\} & t\in T \\
y & \in & [0,1] &
\end{array}
\qquad \text{is equivalent to} \qquad
\begin{array}{lcll}
z_t & \ge & 0 & t\in T \\
z_t & \le & x_t & t\in T \\
\sum_{t\in T}x_t & = & 1 & \\
\sum_{t\in T}z_t & = & y & \\
x_t & \in & \{0,1\} & t\in T \\
y & \in & [0,1] &
\end{array}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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}
      Linear1Con[a,t].block = blockIndex[a];
end;
```

The budget constraint links all the ATMs, and it remains in the master problem. Finally, the following statements use the decomposition algorithm to solve the problem to within 1% of proven optimality:

```
/* solve with the decomposition algorithm */
solve with milp / nthreads=4 relobjgap=0.01 decomp;
```

The solution summary is displayed in Output 15.8.1.

Output 15.8.1 Solution Summary Table

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	CashFlowDiff
Solution Status	Optimal within Relative Gap
Objective Value	2465540.9823
Relative Gap	0.0052611142
Absolute Gap	12903.605401
Primal Infeasibility	6.7044766E-8
Bound Infeasibility	0
Integer Infeasibility	0
Best Bound	2452637.3769
Nodes	1
Iterations	8
Presolve Time	4.81
Solution Time	74.27

The iteration log, which contains the problem statistics, the progress of the solution, and the optimal objective value, is shown in Output 15.8.2.

Output 15.8.2 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 561 variables and 390 constraints.
NOTE: The MILP presolver removed 21279 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 5919 variables, 3990 constraints, and 37599 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 5.138%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 5919 (100%) variables and 3890 (97.49%) 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         .   61   23
         2       0.0000       0.0000             .    0.00%         .   61   23
NOTE: Starting phase 2.
         3   2.4432e+06   2.7489e+06             .   12.51%         .  105   40
         5   2.4526e+06   2.4948e+06   2.4948e+06    1.72%    1.72%  149   60
NOTE: The Decomposition algorithm stopped on the integer RELOBJGAP= option.
         8   2.4526e+06   2.4642e+06   2.4655e+06    0.47%    0.53%  157   69
         Node  Active   Sols         Best          Best      Gap    CPU   Real
                                   Integer         Bound             Time   Time
            0       0      5   2.4655e+06   2.4526e+06    0.53%    157     69
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 69.28 seconds.
NOTE: Optimal within relative gap.
NOTE: Objective = 2465540.9823.
```

Example 15.9: Kidney Donor Exchange and METHOD=SET

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 be a donor. If the donor and recipient are incompatible (because of blood type, tissue mismatch, and so on), the transplant cannot happen. Now suppose two donor-recipient pairs, i and j, are in this situation, but donor i is compatible with recipient j and donor j is compatible with recipient i. Then two transplants can take place in a two-way swap, shown in Figure 15.8. More generally, an n-way swap can be performed involving n donors and n recipients (Willingham 2009).

Figure 15.8 Kidney Donor Exchange Two-Way Swap

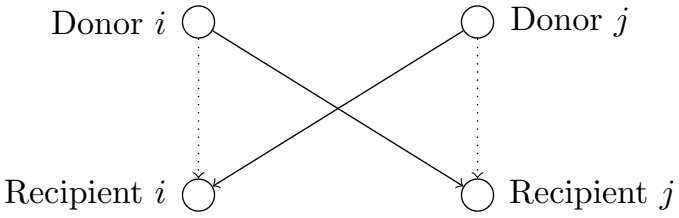
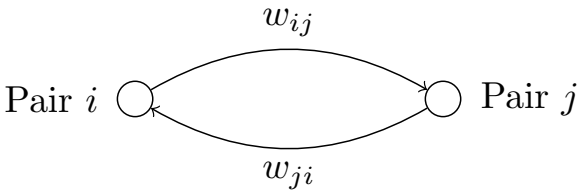


Figure 15.9 Kidney Donor Exchange Network



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, as shown in Figure 15.9. 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 donates 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. For 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 that, 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:

$$
\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{(i,j)\in A}\sum_{m\in M} w_{ij}x_{ijm} & \\
\text{subject to} & \displaystyle\sum_{m\in M} y_{im} + s_i = 1 & i \in N \quad \text{(Packing)} \\
& \displaystyle\sum_{(i,j)\in A} x_{ijm} = y_{im} & i \in N,\ m \in M \quad \text{(Donate)} \\
& \displaystyle\sum_{(i,j)\in A} x_{ijm} = y_{jm} & j \in N,\ m \in M \quad \text{(Receive)} \\
& \displaystyle\sum_{(i,j)\in A} x_{ijm} \le L & m \in M \quad \text{(Cardinality)} \\
& x_{ijm} \in \{0,1\} & (i,j) \in A,\ m \in M \\
& y_{im} \in \{0,1\} & i \in N,\ m \in M \\
& s_i \in \{0,1\} & i \in N
\end{array}
$$

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 727, 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 by 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;
   call streaminit(1);
   do i = 0 to &n - 1;
      do j = 0 to &n - 1;
         if i eq j then continue;
         else if rand('UNIFORM') < &p then do;
            weight = rand('UNIFORM');
            output;
         end;
      end;
   end;
run;
```

In this case, you can specify METHOD=SET and let the decomposition algorithm automatically detect the set partitioning master constraints (Packing) and each independent matching subproblem. 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;

   /* automatically decompose using METHOD=SET */
   solve with milp / presolver=basic decomp=(method=set);

   /* 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 15.9.1.

Output 15.9.1 Solution Summary

The OPTMODEL Procedure

Solution Summary	
Solver	MILP
Algorithm	Decomposition
Objective Function	TotalWeight
Solution Status	Optimal
Objective Value	24.850855395
Relative Gap	0
Absolute Gap	0
Primal Infeasibility	8.881784E-16
Bound Infeasibility	8.881784E-16
Integer Infeasibility	8.881784E-16
Best Bound	24.850855395
Nodes	9
Iterations	93
Presolve Time	0.02
Solution Time	61.20

The iteration log is displayed in Output 15.9.2.

Output 15.9.2 Log

```
NOTE: There were 208 observations read from the data set WORK.ARCDATA.
NOTE: Problem generation will use 4 threads.
NOTE: The problem has 15092 variables (0 free, 0 fixed).
NOTE: The problem has 15092 binary and 0 integer variables.
NOTE: The problem has 9751 linear constraints (49 LE, 9702 EQ, 0 GE, 0 range).
NOTE: The problem has 45080 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 5081 variables and 3366 constraints.
NOTE: The MILP presolver removed 15175 constraint coefficients.
NOTE: The MILP presolver modified 0 constraint coefficients.
NOTE: The presolved problem has 10011 variables, 6385 constraints, and 29905 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 SET 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 number of block threads has been reduced to 1 threads.
NOTE: The problem has a decomposable structure with 49 blocks. The largest block covers 2.02%
      of the constraints in the problem.
NOTE: The decomposition subproblems cover 9947 (99.36%) variables and 6321 (99%) 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
.	350.7353	10.9864	10.9864	96.87%	96.87%	0	0
1	350.7353	10.9864	10.9864	96.87%	96.87%	0	0
2	332.8644	10.9864	10.9864	96.70%	96.70%	0	0
3	330.4272	10.9864	10.9864	96.68%	96.68%	0	0
5	322.4758	14.8476	14.8476	95.40%	95.40%	0	0
7	322.4758	16.3175	16.3175	94.94%	94.94%	1	1
8	285.0281	16.3175	16.3175	94.28%	94.28%	1	1
.	285.0281	18.6410	18.2683	93.46%	93.59%	2	2
10	285.0281	18.6410	18.2683	93.46%	93.59%	2	2
12	246.0827	20.1977	20.1977	91.79%	91.79%	3	3
13	231.1332	20.1977	20.1977	91.26%	91.26%	3	3
14	211.6250	20.2621	20.1977	90.43%	90.46%	3	3
16	188.7069	21.7075	20.1977	88.50%	89.30%	3	3
17	185.9179	21.7075	20.1977	88.32%	89.14%	3	3
18	155.7681	22.1348	20.1977	85.79%	87.03%	3	4
.	155.7681	23.1832	20.4068	85.12%	86.90%	3	4
20	144.9324	23.1832	20.4068	84.00%	85.92%	3	4
22	144.9324	24.5652	24.5652	83.05%	83.05%	7	7
23	121.5242	24.5652	24.5652	79.79%	79.79%	7	7
25	99.4004	24.5652	24.5652	75.29%	75.29%	7	7
29	93.5107	24.5652	24.5652	73.73%	73.73%	7	7
30	93.5107	24.6129	24.5652	73.68%	73.73%	7	7

Output 15.9.2 *continued*

```
        33      80.5044      25.1072      24.5652   68.81%   69.49%    7    7
        35      80.4668      25.3238      24.5652   68.53%   69.47%    7    8
        37      66.2531      25.3737      24.5652   61.70%   62.92%    7    8
        38      57.0024      25.3737      24.5652   55.49%   56.90%    7    8
        39      48.6031      25.4025      24.5652   47.73%   49.46%    7    8
         .      48.6031      25.4025      24.5652   47.73%   49.46%    7    8
        40      38.7784      25.4025      24.5652   34.49%   36.65%    7    8
        41      38.2701      25.4025      24.5652   33.62%   35.81%    7    8
        42      33.5632      25.4058      24.5652   24.30%   26.81%    7    8
        43      33.3733      25.4080      24.5652   23.87%   26.39%    7    8
        44      26.6513      25.4146      24.5652    4.64%    7.83%    7    8
        46      26.2248      25.4177      24.5652    3.08%    6.33%    7    8
        47      25.8597      25.4177      24.5652    1.71%    5.01%    7    8
        48      25.4194      25.4194      24.5652    0.00%    3.36%    8    8
NOTE: Starting branch and bound.
         Node  Active   Sols         Best         Best      Gap     CPU   Real
                                  Integer        Bound             Time   Time
            0       1     13      24.5652      25.4194    3.36%      8      8
            1       3     14      24.8509      25.4194    2.24%     39     40
            8       0     14      24.8509      24.8509    0.00%     59     61
NOTE: The Decomposition algorithm used 4 threads.
NOTE: The Decomposition algorithm time is 61.10 seconds.
NOTE: Optimal.
NOTE: Objective = 24.850855395.
NOTE: The data set WORK.SOLUTION has 47 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.

```
data Solution;
   set Solution;
run;
proc optnet
   direction = directed
   links     = Solution;
   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 15.10.

Figure 15.10 Optimal Donor Exchanges

cycle=1

order	node
1	2
2	18
3	90
4	26
5	84
6	53
7	62
8	2

cycle=2

order	node
1	6
2	96
3	27
4	93
5	23
6	51
7	87
8	78
9	43
10	41
11	6

cycle=3

order	node
1	3
2	56
3	5
4	83
5	45
6	63
7	14
8	64
9	69
10	92
11	3

cycle=4

order	node
1	37
2	39
3	89
4	77
5	37

Figure 15.10 *continued*

cycle=5

order	node
1	21
2	85
3	82
4	73
5	21

cycle=6

order	node
1	38
2	79
3	71
4	38

cycle=7

order	node
1	0
2	99
3	33
4	29
5	20
6	24
7	97
8	31
9	46
10	0

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, Department of Electronics, Computer Science, and Systems, University of Bologna.

Dantzig, G. B., and Wolfe, P. (1960). "Decomposition Principle for Linear Programs." *Operations Research* 8:101–111. 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., 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: Mixed Integer Programming Library Version 5." *Mathematical Programming Computation* 3:103–163. http://dx.doi.org/10.1007/s12532-011-0025-9.

Ralphs, T. K., and Galati, M. V. (2006). "Decomposition and Dynamic Cut Generation in Integer Linear Programming." *Mathematical Programming* 106:261–285. http://dx.doi.org/10.1007/S10107-005-0606-3.

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. http://dx.doi.org/10.1016/j.orl.2005.05.009.

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

Subject Index

Syntax Index