Paper 159-2012

New Features in SAS/OR® 12.1

Ed Hughes and Manoj Chari, SAS Institute Inc., Cary, NC, USA

ABSTRACT

SAS/OR 12.1 debuts a broad range of new capabilities and enhanced features in operations research. This paper surveys these additions, emphasizing their benefits and their application to business problems, and especially to large business problems.

Optimization improvements include faster problem generation and support for additional specialized solution methods in the OPTMODEL procedure. SAS/OR 12.1 can also exploit multicore computing in an updated version of the nonlinear programming multistart method and in underlying methods for the LP, QP, and NLP solvers.

Other updates include an expanded set of constraint classes for the CLP procedure and enhanced SAS® Simulation Studio support for large models and large experiments (many factors and responses).

INTRODUCTION

This paper surveys many of the enhancements included with SAS/OR 12.1 and SAS Simulation Studio 12.1, which is a graphical discrete-event simulation application that is part of SAS/OR 12.1 on the Windows operating system. The practical implications of these improvements and additions are highlighted and selected examples illustrate some of the new features.

This paper assumes that you possess a basic understanding of operations research problems and methods and thus does not dwell on the basic concepts in any area, except as needed to establish the context for the new features. Instead the paper focuses on how the new SAS/OR capabilities should prove useful in your operations research modeling and analysis work.

This paper has been prepared in advance of the release of SAS/OR 12.1, so some details of the new features are not included since they are still evolving. A more extensive version of this paper will be made available later in the Knowledge Base area of the support.sas.com website.

NEW OPTIMIZATION CAPABILITIES

SAS/OR 12.1 enhances its capabilities in linear, mixed-integer linear, quadratic, and general nonlinear optimization. These improvements range from the introduction of new solver methods to enhanced controls and new implementations for current solvers to behind-the-scenes upgrades, all contributing to significant improvements in optimization performance. Overall, these enhancements enable you to solve more optimization problems, tackle larger and more complex problems, and spend less time solving them.

SUPPORTING TECHNOLOGIES

The SAS/OR 12.1 underlying improvements in optimization are chiefly related to multithreading, which denotes the use of multiple computational cores to enable computations to be executed in parallel rather than in serial fashion. Multithreading can provide dramatic performance improvements for optimization since these underlying computations are performed many times in the course of an optimization process.

The underlying linear algebra operations for the linear (LP), quadratic (QP), and nonlinear (NLP) interior point optimization algorithms are now multithreaded. The LP, QP, and NLP solvers can be used by PROC OPTMODEL, PROC OPTLP, and PROC OPTQP in SAS/OR. For nonlinear optimization with PROC OPTMODEL, the evaluation of nonlinear functions is multithreaded for improved performance.

Finally, PROC OPTMODEL itself benefits from multithreading in the process of creating an optimization model from PROC OPTMODEL statements. PROC OPTMODEL contains powerful declarative and programming statements and is adept at enabling data-driven definition of optimization models, with the result that a rather small section of PROC OPTMODEL code can create a very large optimization model when it is executed. Multithreading is an ideal improvement to this process and can dramatically shorten the time needed to create an optimization model.

SAS/OR users can use the NTHREADS= option in the PERFORMANCE statement in PROC OPTMODEL and other SAS/OR optimization procedures to specify the number of cores to be used. Otherwise, SAS detects the number of cores available and uses them.

NONLINEAR OPTIMIZATION

The nonlinear optimization solver used by PROC OPTMODEL builds on the introduction of multithreading for its two most significant improvements in SAS/OR 12.1. First, in addition to the nonlinear solver options SOLVER=ACTIVESET and SOLVER=INTERIORPOINT, SAS/OR 12.1 introduces the SOLVER=CONCURRENT option, with which you can invoke both the Active Set and Interior Point solvers for the specified problem, running in parallel on separate threads. The solution process terminates when either of the solvers terminates. If you are not sure which method will perform best for your nonlinear optimization problem, the SOLVER=CONCURRENT option should prove very useful and illuminating. If your task involves solving a large number of similarly structured problems, either as a group or as part of a larger analytic flow that is invoked repeatedly, using the SOLVER=CONCURRENT option with a representative problem can provide insights that are critical to more efficient operations.

Multithreading is also central to the nonlinear optimization solver's enhanced multistart capability, which now takes advantage of multiple threads to execute optimizations from multiple starting points in parallel. The multistart capability is essential for problems that feature nonconvex nonlinear functions in either or both of the objective and the constraints since such problems might have multiple locally optimal points. Starting optimization from several different starting points helps to overcome this difficulty, and multithreading this process helps to ensure that the overall optimization process is as quick as possible.

In SAS/OR 12.1, the nonlinear optimization solver also expands the irreducible infeasible set (IIS) capability (which is already present in the linear optimization solver) to include linear constraints that are specified for a nonlinear optimization problem. For an infeasible optimization problem, the IIS feature identifies an irreducible infeasible set of bounds, constraints, or both. For this set, removing any one of the constraints or bounds renders the remaining set feasible. This feature can be very useful for identifying sources of infeasibility, especially in large optimization problems. Since the concept of a set of constraints being collectively globally infeasible is valid only for linear constraints, the IIS capability is limited to linear constraints when applied to nonlinear optimization. The syntax is identical to that for linear optimization; the IIS= option in the SOLVE WITH NLP statement enables IIS detection if IIS=ON and disables it if IIS=OFF (the default value).

LINEAR AND MIXED-INTEGER LINEAR OPTIMIZATION

SAS/OR 12.1 delivers a great number of additions and upgrades to linear and mixed-integer linear optimization, improving performance on current features and expanding functionality. Mixed-integer linear optimization performance benefits not only from its own improvements but also from improvements in the simplex-based linear optimization algorithms, which are employed repeatedly as components of mixed-integer linear optimization.

Linear Optimization

Extensive improvements to the primal and dual simplex linear optimization solvers produce better performance and better integration with the crossover algorithm, which converts solutions found by the interior point solver into more usable basic optimal solutions. The crossover algorithm itself has undergone extensive enhancements that dramatically improve its speed and stability. It's also now possible to use the crossover algorithm with the network simplex solver, which might identify a solution to a partial network problem that isn't a basic optimal solution.

Paralleling developments in nonlinear optimization, SAS/OR 12.1 linear optimization introduces a concurrent solver, invoked with the SOLVER=CONCURRENT option, in the SOLVE WITH LP statement for PROC OPTMODEL or in the PROC OPTLP statement. The concurrent LP solver runs a selection of linear optimization solvers in parallel on different threads, with settings determined to suit the problem at hand. The optimization process terminates when the first solver identifies an optimal solution. As with nonlinear optimization, the concurrent LP solver has the potential to produce significant reductions in the time needed to solve challenging problems and to provide insights that are useful when you solve a large number of similarly structured problems.

In linear network optimization, improvements to the network simplex algorithm enhance its ability to identify networks embedded in linear optimization problems. The network extraction process is now able to identify more network elements and do so more quickly. A more fundamental change to network optimization is the introduction of PROC OPTNET, which includes state-of-the-art algorithms for analyzing network structures and solving network-oriented linear optimization problems. PROC OPTNET is covered in a later section of this paper.

Mixed-Integer Linear Optimization

Mixed-integer linear optimization in SAS/OR 12.1 builds on the advances in linear optimization and boasts its own improved and expanded capabilities. Overall, solver speed has increased by over 50% (on a library of test problems) compared to SAS/OR 9.3. The branch-and-bound algorithm, the central element of the mixed-integer optimization solver, has approximately doubled its ability to evaluate and solve component linear optimization problems (which are referred to as nodes in the branch-and-bound tree). These improvements have produced very significant reductions in solution time for difficult problems.

The Decomposition Algorithm

The most fundamental change to both linear and mixed-integer linear optimization in SAS/OR 12.1 is the addition of the decomposition (DECOMP) algorithm, which is invoked with a specialized set of options in the SOLVE WITH LP and SOLVE WITH MILP statements for PROC OPTMODEL or in the PROC statement for PROC OPTLP and PROC OPTMILP (with a modification of the MPS formatted input SAS data set). For many linear and mixed-integer linear optimization problems, most of the constraints apply only to a small set of decision variables; the constraints are thus “local” to these decision variables. Typically there are many such sets of local constraints, complemented by a small set of global or linking constraints that apply to all or most of the decision variables. Optimization problems with these characteristics are said to have a “block-angular” structure, since it’s easy to arrange the rows of the constraint matrix so that the nonzero values, which correspond to the local sets of constraints, appear as blocks along the main diagonal.

The DECOMP algorithm exploits this block-diagonal structure, decomposing the overall optimization problem into a set of component problems that can be solved in parallel on separate computational threads. The algorithm repeatedly solves these component problems and then cycles back to the overall problem to update key information that is used the next time the component problems are solved. This process continues iteratively and in the end produces a solution to the complete problem, with the linking constraints present. The combination of parallelized solving of the component problems and the iterative coordination with the solution of the overall problem can result in great reductions in solution time for problems that were previously regarded as time-consuming or practically impossible to solve successfully.

To use the DECOMP algorithm, the blocks of the constraint matrix, which correspond to component problems, must be identified either manually or automatically. The METHOD= option controls the means by which blocks are identified. METHOD=USER enables you specify the blocks yourself, using the .block suffix to declare blocks. This is by far the most common method for defining blocks. If your problem has a significant or dominant network structure, you can use METHOD=NETWORK to identify the blocks in the problem automatically. Finally, if no linking constraints are present in your problem, then METHOD=AUTO identifies the blocks automatically; in future releases METHOD=AUTO will expand its capabilities to deal with linking constraints.

The DECOMP algorithm uses a number of detailed options that specify how the solution processes for the component problems and the overall problem are configured and how they coordinate with each other. You can also specify the number of computational threads to make available for processing component problems and the level of detail in information that appears in the SAS log. Options specific to the linear and mixed-integer linear solvers used by the DECOMP algorithm are largely identical to those for the respective solvers.

A simple example illustrates the use of the DECOMP algorithm with PROC OPTMODEL. The generalized assignment problem (GAP) involves the assignment of agents to tasks, with the resource usage and the cost for completing each task potentially differing according to the agent assigned. Each agent has a finite resource budget that cannot be exceeded, and each task must be assigned to exactly one agent. The goal is to minimize the total cost of the assignments.

The following SAS data sets indicate the assignment cost, assignment resource usage, and resource budgets for a GAP that involves 15 tasks and 5 agents:

```
/* for each agent i, cost of allocating job j to agent i */
data cost;
   input c1-c15;
   datalines;
17 21 22 18 24 15 20 18 19 18 16 22 24 24 16
23 16 21 16 17 16 19 25 18 21 17 15 25 17 24
16 20 16 25 24 16 17 19 19 18 20 16 17 21 24
19 19 22 22 20 16 19 17 21 19 25 23 25 25 25
18 19 15 15 21 25 16 16 23 15 22 17 19 22 24
;

/* for each agent i, the resource consumed in allocating job j to agent i */
data resource;
   input r1-r15;
   datalines;
8  15 14 23  8 16  8 25  9 17 25 15 10  8 24
15  7 23 22 11 11 12 10 17 16  7 16 10 18 22
21 20  6 22 24 10 24  9 21 14 11 14 11 19 16
```

```
  20 11  8 14  9  5  6 19 19  7  6  6 13  9 18
   8 13 13 13 10 20 25 16 16 17 10 10  5 12 23
  ;

  /* resource capacity of agent i */
  data capacity;
     input b @@;
     datalines;
  36 34 38 27 33
  ;
```

The following PROC OPTMODEL staetments declare index sets and parameters, read in data for the problem instance, and declare the constraints and objective for the problem:

```
  proc optmodel;
     num numJobs = 15;

     /* declare index sets */
     set AGENTS;
     set JOBS = 1..numJobs;

     /* declare parameters */
     num cost {AGENTS, JOBS};
     num resource {AGENTS, JOBS};
     num capacity {AGENTS};

     /* read data sets to populate data */
     read data capacity into AGENTS=[_n_] capacity=b;
     read data cost     into [i=_n_] {j in JOBS} <cost[i,j]=col('c'||j)>;
     read data resource into [i=_n_] {j in JOBS} <resource[i,j]=col('r'||j)>;

     /* x[i,j] = 1 if we assign agent i to job j */
     var x {AGENTS,JOBS} >= 0 binary;

     /* minimize the total cost of assignment */
     min TotalCost =
        sum {i in AGENTS, j in JOBS} cost[i,j] * x[i,j];

     /* each agent i has a capacity on the resources consumed for its assigned jobs */
     con Capacity {i in AGENTS}:
        sum {j in JOBS} resource[i,j] * x[i,j] <= capacity[i];

     /* each job j must be assigned to exactly one agent */
     con Assignment {j in JOBS}:
        sum {i in AGENTS} x[i,j] = 1;
```

Note that each of the Capacity constraints defined in the preceding statements involves only the decision variables that correspond to the resource for which it is defined; thus each Capacity constraint forms a distinct block of coefficients in the overall constraint matrix. These blocks are declared with the following OPTMODEL statements, in which the *i*th Capacity constraint becomes the *i*th block:

```
     /* each capacity constraint defines a block */
     for {i in AGENTS}
        Capacity[i].block = i;
```

Finally the SOLVE statement uses the DECOMP algorithm in solving this problem:

```
     /* solve with decomp (knapsack decomposition) */
     solve with milp /
       decomp =(method=user);
```

The CREATE DATA statement creates an output data set for the problem that is then printed with PROC PRINT:

```
     /* write solution to data set */
     create data soldata from [i j] =
       {i in AGENTS, j in JOBS: x[i,j] > 0.5} x=(round(x[i,j])) cost[i,j];
```

```
quit;

proc print data=soldata; sum cost; run;
```

Output 1 shows the SAS data set soldata, which contains the optimal assignment of agents to tasks. The total cost of the assignment is 261.

```
Obs     i      j     x     cost

  1     1      9     1       19
  2     1     15     1       16
  3     2      2     1       16
  4     2     11     1       17
  5     2     14     1       17
  6     3      3     1       16
  7     3      8     1       19
  8     3     13     1       17
  9     4      5     1       20
 10     4      6     1       16
 11     4      7     1       19
 12     4     10     1       19
 13     5      1     1       18
 14     5      4     1       15
 15     5     12     1       17
                           ====
                            261
```

Output 1. Optimal (Least-Cost) Assignment of Agents to Tasks

Setting the Cutting Plane Strategy

Cutting planes are a major component of the mixed-integer linear optimization solver; they help speed its progress by removing fractional (not integer feasible) solutions. SAS/OR 12.1 adds the CUTSTRATEGY= option in the PROC OPTMILP statement and in the SOLVE WITH MILP statement for PROC OPTMODEL, enabling you to determine the aggressiveness of your overall cutting plane strategy. This option complements the individual cut class controls (CUTCLQUE=, CUTGOMORY=, CUTMIR=, and so on), which enable you to enable or disable certain cut types, and the ALLCUTS= option, which enables or disables all cutting planes. In contrast, the CUTSTRATEGY= option controls cuts at a higher level, creating a profile for cutting plane use. As the cut strategy becomes more aggressive, more effort is directed toward creating cutting planes and more cutting planes are applied. The available values of the CUTSTRATEGY= option are AUTOMATIC, BASIC, MODERATE, and AGGRESSIVE, with AUTOMATIC as the default. The precise cutting plane strategy that corresponds to each of these settings can vary from problem to problem, because the strategy is tuned to suit not only the chosen aggressiveness level but also the problem at hand. For example, if a certain class of cutting planes is known to be inappropriate for the problem, then the strategy (at any level of aggressiveness) omits that class. The purpose of the CUTSTRATEGY= option is to provide a simple control that creates a sophisticated cutting plane approach.

Conflict Search

Another means of accelerating the solution process for mixed-integer linear optimization takes information from infeasible linear optimization problems that are encountered during an initial exploratory phase of the branch-and-bound process. This information is analyzed and ultimately is used to help the branch-and-bound process proceed more efficiently and avoid combinations of decision variable values that are known to lead to infeasibility. This approach, known as conflict analysis or conflict search, thus shortens the overall solution process by influencing presolve operations on branch-and-bound nodes, generation of cutting planes, computation of decision variable bounds, and branching. Although the approach is complex, its application in SAS/OR 12.1 is straightforward. The CONFLICTSEARCH= option in the PROC OPTMILP statement or the SOLVE WITH MILP statement in PROC OPTMODEL enables you to specify the level of conflict search to be performed. The available values for the CONFLICTSEARCH= option are NONE, AUTOMATIC, MODERATE, and AGGRESSIVE. A more aggressive strategy explores more branch-and-bound nodes initially before the branch-and-bound algorithm is restarted with information from infeasible nodes included. The default value is AUTOMATIC, which in general computes the best strategy based on the characteristics of the problem being solved.

Parameter Tuning

The final SAS/OR 12.1 improvement to the mixed-integer linear optimization solver is parameter tuning, which helps you determine the best option settings when you use PROC OPTMILP. There are many options available (and often many choices for each option) for mixed-integer linear optimization, including controls on the presolve process, branching, heuristics, and cutting planes. The TUNER statement for PROC OPTMILP enables you to investigate the effects of the many possible combinations of option settings on solver performance and determine which option values should produce the best performance. The PROBLEMS= option enables you to submit several problems for tuning at once. The OPTIONMODE= option specifies the parameters that are to be tuned. OPTIONMODE=USER indicates that you will supply a set of parameters and initial values via the OPTIONVALUES= data set, OPTIONMODE=AUTO (the default) tunes a small set of predetermined parameters, and OPTIONMODE=FULL tunes a much more extensive parameter set.

Parameter tuning starts with an initial set of parameter values and uses those settings to solve the problem. The problem is solved repeatedly with different parameter values, with a local search algorithm guiding the choices for parameter values. When the tuning process terminates, the best parameter values are output to a data set specified by the SUMMARY= option. You can control the amount of time used by this process with the MAXTIME= option. You can multithread this process by using the NTHREADS= option in the PERFORMANCE STATEMENT for PROC OPTMILP, permitting analysis of various settings to occur simultaneously.

NETWORK OPTIMIZATION WITH PROC OPTNET

PROC OPTNET, new in SAS/OR 12.1, provides a wealth of algorithms for investigating the characteristics of networks and solving network-oriented optimization problems. A network, sometimes also referred to as a graph, consists of a set of nodes that are connected by a set of arcs, edges, or links. There are many applications of network structures in real-world problems, including supply chain analysis, communications, transportation, and utilities problems. Table 1 shows the PROC OPTNET statement that corresponds to each class of network problem that it addresses.

Problem Type	**PROC OPNET Statement**
Minimum-cost network flow	NETFLOW_MINCOST
Shortest path	SHORTPATH
Minimum-cost linear assignment	LINEAR_ASSIGNMENT
Traveling salesman (TSP)	TSP
Minimum spanning tree	MINSPANTREE
Minimum cut	MINCUT
Cycle detection	CYCLE
Connected components	CONCOMP
Biconnected components	BICONCOMP
Transitive closure	TRANSITIVE_CLOSURE
Maximal cliques	CLIQUE

Table 1. Network Problem Types Addressed by PROC OPTNET

The formats of PROC OPTNET input data sets are designed to fit network-structured data, easing the process of specifying network-oriented problems. The underlying algorithms are highly efficient and can successfully address problems of varying levels of detail and scale. PROC OPTNET is a logical destination for users who are migrating from some of the legacy optimization procedures in SAS/OR. Former users of PROC NETFLOW can turn to OPTNET to solve shortest path and minimum-cost network flow problems, and former users of PROC ASSIGN can instead use the LINEAR_ASSIGNMENT statement in PROC OPTNET to solve assignment problems.

A simple example illustrates the use of PROC OPTNET in solving a shortest path problem. In this problem a SAS employee wants to find the shortest path (in time traveled) from SAS in Cary, NC (SAS Campus Drive) to her home in Raleigh, NC (614 Capital Boulevard) at 5:00 p.m. The following SAS data set details the approximate travel distances and speed limits between these two points and several intermediate locations during the evening rush hour:

```
data LinkSetInRoadNC5pm;
   input start_inter $1-20 end_inter $20-40 miles miles_per_hour;
   datalines;
614CapitalBlvd      Capital/WadeAve      0.6  25
614CapitalBlvd      Capital/US70W        0.6  25
614CapitalBlvd      Capital/US440W       3.0  45
Capital/WadeAve     WadeAve/RaleighExpy  3.0  25 /*high traffic*/
Capital/US70W       US70W/US440W         3.2  60
US70W/US440W        US440W/RaleighExpy   2.7  60
Capital/US440W      US440W/RaleighExpy   6.7  60
US440W/RaleighExpy  RaleighExpy/US40W    3.0  60
WadeAve/RaleighExpy RaleighExpy/US40W    3.0  60
RaleighExpy/US40W   US40W/HarrisonAve    1.3  55
US40W/HarrisonAve   SASCampusDrive       0.5  25
;
```

A simple DATA step transforms speed limit and distance data into traversal time for each link:

```
data LinkSetInRoadNC5pm;
   set LinkSetInRoadNC5pm;
   time_to_travel = miles * 1/miles_per_hour * 60;
run;
```

With the SAS data set LinkSetInRoadNC5pm as input, the PROC OPTNET code for solving this problem is as follows:

```
proc optnet
   data_links = LinkSetInRoadNC5pm;
   data_links_var
      from = start_inter
      to = end_inter
      weight = time_to_travel;
   shortpath
      outpaths = ShortPath
      source = "SASCampusDrive"
      sink = "614CapitalBlvd";
run;
```

The DATA_LINKS statement specifies the input SAS data set, and the options in the DATA_LINKS_VAR statement indicate the variables in the input data set that carry the start location, end location, and weight (in this case, travel time) information for each link. The SHORTPATH statement and its options indicate the output SAS data set and the start and finish points for the desired shortest path. The output SAS data set, which indicates the shortest path, is shown in Output 2.

```
                                                     time_to_
order    start_inter            end_inter            travel

  1      SASCampusDrive         US40W/HarrisonAve     1.2000
  2      US40W/HarrisonAve      RaleighExpy/US40W     1.4182
  3      RaleighExpy/US40W      US440W/RaleighExpy    3.0000
  4      US440W/RaleighExpy     US70W/US440W          2.7000
  5      US70W/US440W           Capital/US70W         3.2000
  6      Capital/US70W          614CapitalBlvd        1.4400
                                                    ========
                                                     12.9582
```

Output 2. Shortest Path for Road Network at 5:00 p.m.

The diverse network optimization and analysis capabilities of PROC OPTNET provide value in their own right and can also serve as components in larger analytic and optimization processes that possess one or more significant network elements.

NEW FEATURES IN CONSTRAINT PROGRAMMING

The CLP procedure in SAS/OR provides finite-domain constraint programming for solving constraint satisfaction problems (CSPs). The procedure has the ability to address both general and scheduling-oriented CSPs, and with the recent addition of the ability to specify an objective function, also provides an alternate method for solving optimization problems.

In SAS/OR 12.1, PROC CLP adds two classes of constraints that expand its capabilities and can accelerate its solution process. The LEXICO statement is used to impose a lexicographic ordering between pairs of variable lists. Lexicographic order is essentially analogous to alphabetic order but expands the concept to include numeric values. One vector (list) of values is lexicographically less than another if the corresponding elements are equal up to a certain point and immediately after that point the next element of the first vector is numerically less than the second. Thus the vector (1, 2, 4, 7, 3) is lexicographically less than the vector (1, 2, 4, 8,1). Lexicographic ordering can be very useful in eliminating certain types of symmetry that can arise among solutions to constraint satisfaction problems. Imposing a lexicographic ordering eliminates many of the mutually symmetric solutions, reducing the number of permissible solutions to the problem and in turn shortening the solution process.

Another constraint class added to PROC CLP for SAS/OR 12.1 is the bin-packing constraint, imposed via the PACK statement. A bin-packing constraint directs that a specified number of items must be placed into a specified number of bins, subject to the capacities (expressed in numbers of items) of the bins. The PACK statement provides a compact way to express such constraints, which can often be useful components of larger CSPs or optimization problems.

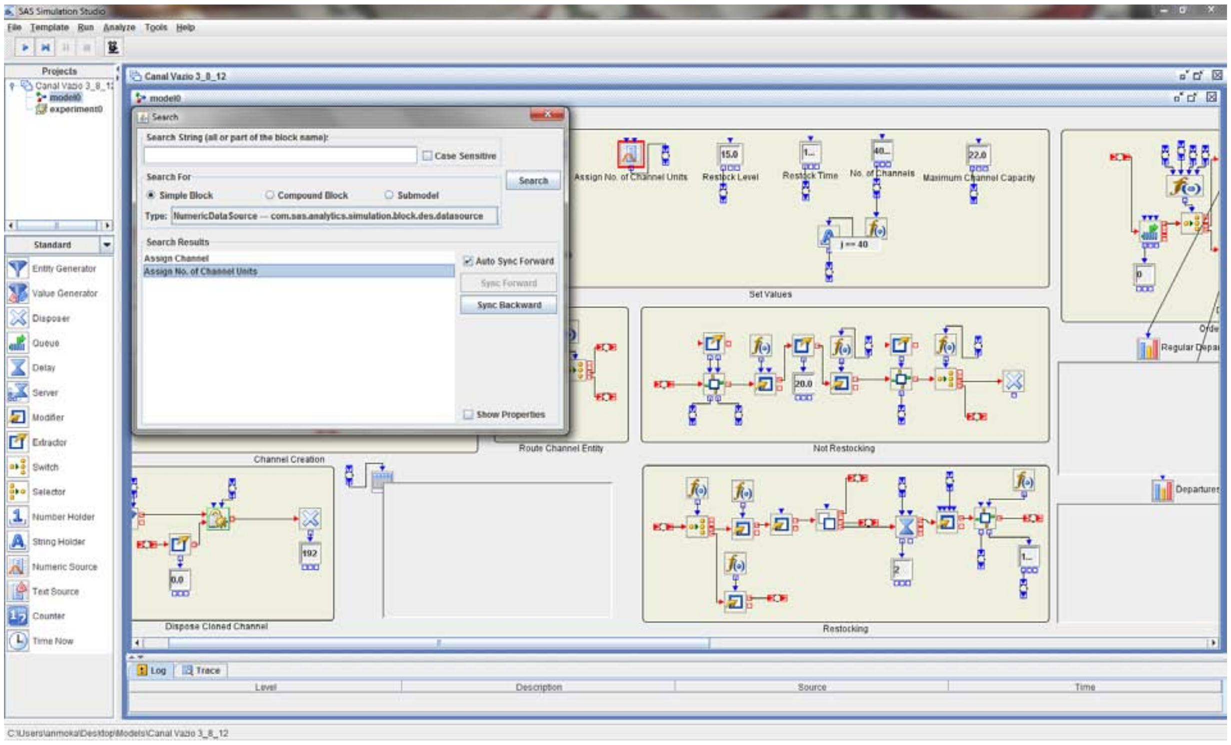
ADVANCES IN SAS SIMULATION STUDIO

SAS Simulation Studio 12.1, a component of SAS/OR 12.1 for Windows environments, adds several features that improve your ability to build, explore, and work with large, complex discrete-event simulation models. Large models present a number of challenges to a graphical user interface such as that of SAS Simulation Studio. Connection of model components, navigation within a model, identification of objects or areas of interest, and management of different levels of modeling are all tasks that can become more difficult as the model size grows significantly beyond what can be displayed on one screen. An indirect effect of model growth is an increased number of factors and responses that are needed to parameterize and investigate the performance of the system being modeled.

Improvements in SAS Simulation Studio 12.1 address each of these issues. In SAS Simulation Studio, you connect blocks by dragging the cursor to create links between output and input ports on regular blocks and Connector blocks. SAS Simulation Studio 12.1 automatically scrolls the display of the Model window as you drag the link being created from its origin to its destination, thus enabling you to create a link between two blocks that are located far apart. Automatic scrolling also enables you to navigate a large model more easily. Instead of using the horizontal and vertical scroll bars of the Model window to move to a new area, in SAS Simulation Studio 12.1 you can simply hold the left mouse button and drag the visible region of the model to the desired area. This works for simple navigation and for moving a block to a new, remote location in the model.

In addition, SAS Simulation Studio 12.1 enables you to search among the blocks in a model and identify the blocks that have a specified type and a certain character string in their label (or both). From the listing of identified blocks you can open the Properties Dialog box for each identified block and edit its settings. Thus, if you can identify a set of blocks that need similar updates, then you can execute these updates without manually searching through the model for qualifying blocks and editing them individually. For very large models this capability not only makes the update process easier but also makes it more thorough because you can identify qualifying blocks centrally.

Figure 1 shows the results of a search for block types. In this case Numeric Source blocks are identified.



Search

Search String (all or part of the block name):

[ ] Case Sensitive

Search For

(●) Simple Block ( ) Compound Block ( ) Submodel

Type: NumericDataSource — com.sas.analytics.simulation.block.des.datasource

Search Results

Assign Channel
Assign No. of Channel Units

[✓] Auto Sync Forward

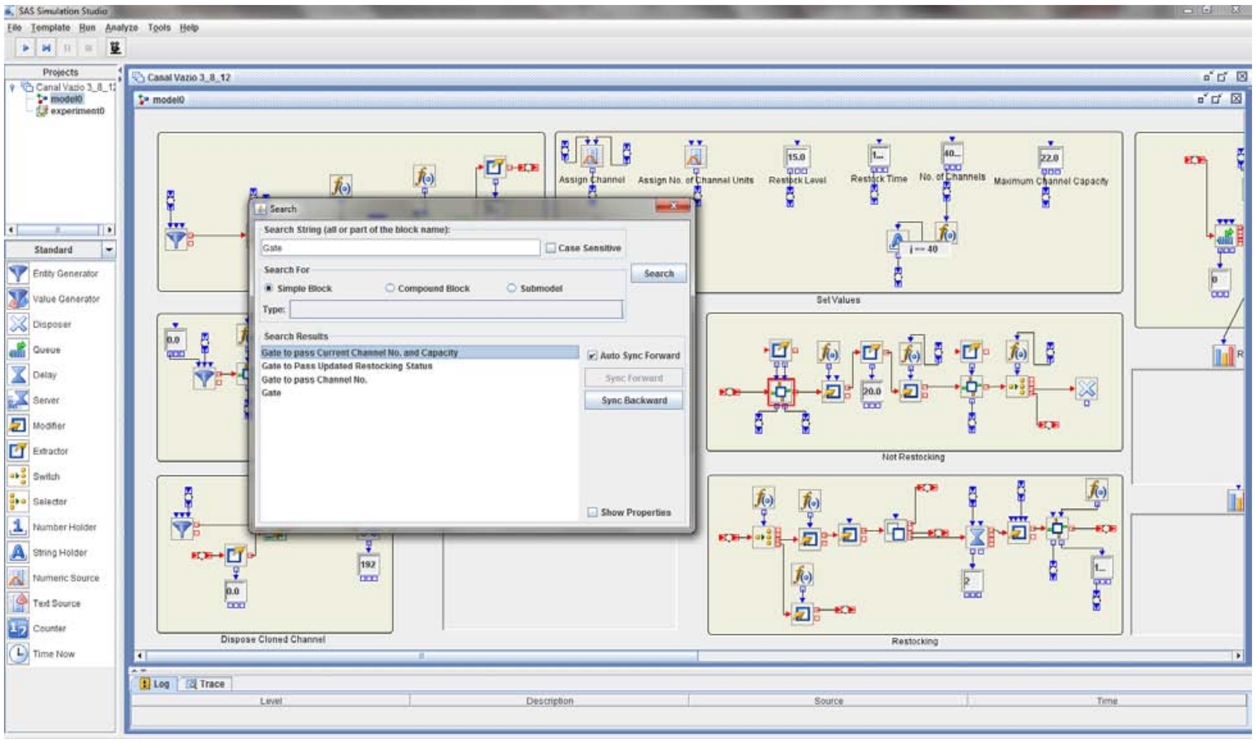
Sync Forward

Sync Backward

[ ] Show Properties

Figure 1. Searching for Block Types

Figure 2 shows the results of a search for a character string “Gate” in block labels.



Search

Search String (all or part of the block name):

Gate

[ ] Case Sensitive

Search For

(●) Simple Block ( ) Compound Block ( ) Submodel

Type:

Search Results

Gate to pass Current Channel No. and Capacity
Gate to Pass Updated Restocking Status
Gate to pass Channel No.
Gate

[✓] Auto Sync Forward

Sync Forward

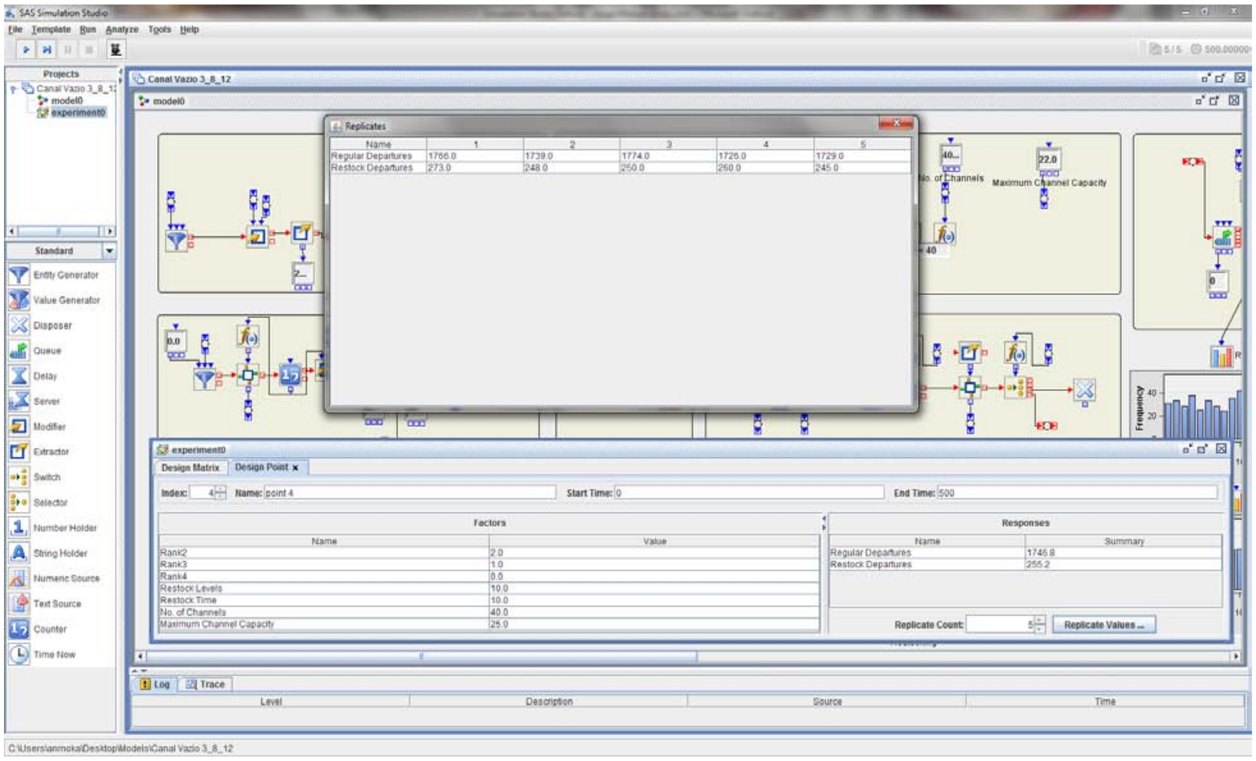
Sync Backward

[ ] Show Properties

Figure 2. Searching for Character Strings in Block Labels

When you design experiments for large simulation models, it's often true that you need a large number of factors to parameterize the model and a large number of responses to track system performance with sufficient detail. This was a challenge prior to SAS Simulation Studio 12.1 because the Experiment window displayed factors and responses in the header row of a table, with design points and their replications' results displayed in the rows below. A very large number of factors and responses did not fit on one screen in this display scheme and you had to scroll across the Experiment window to view all of them.

SAS Simulation Studio 12.1 provides you with two alternate configurations for the Experiment window. The Design Matrix tab presents the tabular layout described above. The Design Point tab presents each design point in its own display. Factors and responses (summarized over replications) are displayed in separate tables, each with the factor or response names appearing in one column and the respective values in a second column. This layout enables a large number of factors and responses to be displayed. Response values for each replication of the design point can be displayed in a separate window. Figure 3 shows the Design Point layout of the Experiment window for SAS Simulation Studio 12.1, including the Replicates window for individual response values.



Name	1	2	3	4	5
Regular Departures	1766.0	1739.0	1774.0	1726.0	1729.0
Restock Departures	273.0	248.0	250.0	260.0	245.0

Index: 4 Name: point 4 Start Time: 0 End Time: 500

Factors

Name	Value
Rank2	2.0
Rank3	1.0
Rank4	0.0
Restock Levels	10.0
Restock Time	10.0
No. of Channels	40.0
Maximum Channel Capacity	25.0

Responses

Name	Summary
Regular Departures	1746.8
Restock Departures	255.2

Replicate Count: 5

Level	Description	Source	Time

Figure 3. Design Point View of the SAS Simulation Studio 12.1 Experiment Window

SAS Simulation Studio 12.1 enhances its multilevel model management features by introducing the submodel component. Like the compound block, the submodel encapsulates a group of SAS Simulation Studio blocks and their connections, but the submodel outpaces the compound block in some important ways. The submodel, when expanded, opens in its own window. This means that a submodel in its collapsed form can be placed close to other blocks in the Model window without allowing space for its expanded form (as must be done for compound blocks). The most important property of the submodel is the fact that it can be copied and instantiated in several locations simultaneously, whether in the same model, in different models in the same project, or across different projects. Each such instance is a direct reference to the original submodel, not a disconnected copy. This means that you can edit the submodel by editing any of its instances; changes made to any instance are propagated to all current and future instances of the submodel. This feature enables you to maintain consistency across your models and projects.

CONCLUSION

SAS/OR 12.1, including SAS Simulation Studio 12.1, introduces many new features and enhancements of current capabilities in optimization, constraint programming, and discrete-event simulation. Optimization improvements and additions expand the range of problem types that you can model, take advantage of multithreading and other advances for better performance and scalability, and give you more options in choosing solution methods. New constraint classes for constraint programming enhance its performance and versatility. Additions to SAS Simulation Studio enable you to build and manage larger simulation models much more easily and to reuse and update model elements more easily and consistently. Overall, SAS/OR 12.1 provides you with better, more diverse tools to address the strategic, tactical, and operational planning problems that are central to operations research.

ACKNOWLEDGMENTS

The authors thank Matthew Galati, Rob Pratt, Melanie Gratton, Ben-Hao Wang, Yan Xu, Phillip Meanor, Emily Lada, Anup Mokashi, Imre Polik, Philipp Christophel, Leo Lopes, Tao Huang, and Joshua Griffin for their assistance with source material, examples, and screen shots. The authors also express their gratitude to Anne Baxter for her invaluable contributions as editor.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Ed Hughes
SAS Institute, Inc.
SAS Campus Drive
Cary, NC 27513
Ed.Hughes@sas.com

Manoj Chari
SAS Institute, Inc.
SAS Campus Drive
Cary, NC 27513
Manoj.Chari@sas.com