Paper 300-2009

New Features in Optimization with SAS/OR® Software

Ed Hughes and Trevor Kearney, SAS Institute Inc., Cary, NC

ABSTRACT

In its recent releases, SAS/OR software has made its optimization capabilities far more powerful and more accessible than ever before. New state-of-the-art optimization solvers and presolvers enable you to tackle larger and more complex optimization models and produce results more quickly. The OPTMODEL procedure, incorporating a rich, expressive optimization modeling language, provides a direct, transparent transition from the formulation of an optimization model to the SAS® statements needed to build, populate, and solve the model.

This paper provides background information about the role of optimization. It also surveys the new SAS/OR optimization advances, highlighting their importance for organizations that face larger and more diverse problems in resource allocation, supply chain planning, and many other domains in which optimization plays a key role.

INTRODUCTION

SAS software includes a diverse and well-established array of optimization capabilities designed to assist in building and solving many different types of optimization models. The greatest concentration of modeling, analysis, and problem-solving capabilities is found in SAS/OR software, but some optimization features are also present in SAS/STAT®, SAS/IML®, and SAS Enterprise Miner™ software. The SAS/ETS® MODEL procedure also provides considerable modeling capabilities. Users at a wide range of major U.S. and international corporations have found these capabilities to be indispensable in building decision-support and decision-guiding solutions aimed at identifying the best and most profitable strategies and action plans for their enterprises.

SAS/OR software includes a completely new generation of optimization modules (procedures) that make it easier to tackle a broad range of optimization problems. The foremost of these is the OPTMODEL procedure, which incorporates a powerful symbolic, interactive modeling language that enables you to work through the iterative process of building, solving, and refining optimization models with clarity and precision. New procedures for solving specific types of optimization problems (specified by using input data sets) are also provided: OPTLP for linear optimization, OPTMILP for mixed-integer optimization, and OPTQP for quadratic optimization. Supporting these procedures is a new suite of optimization solvers, which use state-of-the-art algorithms and deliver great improvements in performance and scalability.

Two additional procedures solve problems that are difficult to model and solve with PROC OPTMODEL and its associated procedures. The GA procedure uses genetic algorithms to solve optimization problems, and the experimental CLP procedure solves constraint satisfaction problems, which are closely related to optimization problems.

Complementing this optimization technology are several current and upcoming SAS solutions that include optimization as either a core function or a critical enabling technology, leveraging the decision-guiding power and insight added by SAS optimization. Examples include SAS Marketing Optimization, the SAS Revenue Optimization Suite, SAS Size Optimization, SAS Inventory Optimization, SAS Service Parts Optimization, and SAS Credit Risk Management.

This paper describes the nature and purpose of optimization and considers the various types of optimization problems that can be solved with SAS/OR software. We explore the value that optimization adds to enterprise data, to a range of analytical technologies, and to business intelligence. We look at the unique advantages that SAS software delivers to optimization modelers. We consider the new SAS/OR optimization features and see how they make the building and solving of optimization models easier, more transparent, and more scalable than ever before.

WHAT IS OPTIMIZATION? HOW ARE OPTIMIZATION MODELS BUILT?

Simply put, optimization is the process of choosing the permissible actions that result in the best outcome. This is the underlying concept, no matter how complex the means by which optimization is implemented. On these terms, virtually all activity—whether of enterprises, groups, individuals, or microorganisms—can be viewed as some form of optimization. Whether the goal is survival of the species through effective propagation, maximization of profits via

production decisions, minimization of risk with purchasing strategies, or something as mundane as finding the shortest route from one place to another, we are all striving to optimize something—or, more likely, several things at once.

Of course, not all attempts at optimization succeed; often success occurs in widely varying degrees of completeness. A number of factors can influence the degree to which optimization succeeds. On the most basic level, we might not have the freedom to take the actions that produce the best results. We might have incomplete or incorrect information about how various actions interact with each other, how they are limited by our circumstances, or how they influence the outcomes. There might be so many choices of actions that it is impossible to evaluate them all. The outcome itself might be inappropriately or inaccurately measured. We might not even have a reliable way to determine if a particular outcome is remotely close to being the best outcome.

These are the kinds of difficulties that plague informal, intuitive, or unstructured attempts at optimization. What is needed in most cases is a more precise description of the decision problem and a more disciplined approach to optimization. This more rigorous approach (termed *mathematical optimization* or, equivalently, *mathematical programming*) can mitigate or eliminate many of the problems we have described.

ELEMENTS AND CATEGORIES OF OPTIMIZATION PROBLEMS

A well-defined approach to optimization begins with a rigorous description of the three key elements of any optimization problem:

- **decision variables,** which are numerical representations of the available actions or choices. Examples include production levels, price settings, and capital or human resource allocations.
- **an objective** that is the goal of the optimization; something to be achieved. This goal must be measurable. Examples include maximizing profit, minimizing distance traveled, and minimizing unused raw materials.
- **constraints** which specify requirements or rules and place limits on how the objective can be pursued by limiting the permissible values of the decision variables. Some examples are machine processing capacity per hour, customer demand by sales territory, raw materials availability, bills of material in manufacturing or assembly, and budgetary restrictions.

Each type of optimization model is characterized by how its decision variables are defined and by the types of mathematical expressions that are used in defining its objective and constraints:

- **Linear programs (LPs)** use only linear expressions in the constraints and the objective (for example, 4x + 3.5y – 2z<=7), and the decision variables can take on any value in specified allowable ranges.
- **Mixed integer linear programs (MILPs)** are linear programs in which some of the decision variables must have integer values.
- **Integer programs (IPs)** are linear programs in which all of the decision variables must have integer values.
- **Nonlinear programs (NLPs)** use nonlinear expressions (such as x^2, cos (y), 1/x, and others) in the constraints or the objective (or both).
- **Quadratic programs (QPs)** are a specific type of NLP with a quadratic objective (involving squares of decision variables or the product of two decision variables) and typically linear constraints.

Within this framework of decision variables, objective, and constraints, the purpose of optimization is to maximize or minimize, as appropriate, the performance metric in the objective by assigning values to the decision variables that satisfy the constraints. Solution methods are tailored to suit each type of optimization model, but in general all begin with an initial solution and then seek to improve it until no further improvement can be found (or until some other predefined criterion for halting is satisfied). At this point the current solution is deemed “optimal.” If the problem is infeasible (the constraints cannot be satisfied) or unbounded (the objective function can be made infinitely positive or negative), the solution process will detect it.

BENEFITS OF OPTIMIZATION

Optimization conveys a number of direct and indirect benefits. This topic is covered at length in the SAS white paper, "Optimization with SAS/OR®: What It Is and How It Adds Value." See the "Recommended Reading" section in this paper for instructions about how to obtain this white paper. An earlier version appears as paper 171-2007 in the proceedings of SAS Global Forum 2007.

BEYOND TRADITIONAL OPTIMIZATION: GENETIC ALGORITHMS

Many compelling business problems do not fit neatly into any of the categories previously described. Some are optimization problems but might have an objective function that is not sufficiently "well-behaved" for traditional modeling and solution methods to work well—or at all. The objective function might not be continuous or might not be smooth, due to the nature of the real-world performance metric that it represents. Instead of a single "global" optimal value, the function might have many "local" optimal values (so that a graph of the objective resembles a mountain range with many small peaks instead of one large peak). There might be multiple objective functions. The objective function might not even be expressible as an algebraic function; perhaps you can determine its value only by consulting a table and finding the value that corresponds to a particular set of values for the decision variables.

Further difficulties could arise with decision variables; they might be constrained to take on only integer or sequence values, creating a new set of challenges to the use of traditional methods. These sorts of optimization problems can arise in areas such as securities management, production sequencing, and route planning.

In these cases, genetic algorithms can be enormously useful since they make no assumptions about the structure of the problem being solved. Genetic algorithms work with a set of solutions to the problem, treat them like a population of organisms, and use functions called "genetic operators" to evolve the population of solutions over successive "generations." These genetic operators are "crossover," which determines how solutions in one generation combine to create new solutions in the next generation, and "mutation," which introduces random variation into this process. Another important element of genetic algorithms is the selection process, which spells out the rules by which solutions from the current generation are chosen to either parent the next generation or to be passed on to the next generation directly.

The selection process tends to favor the "fittest" solutions as measured by their corresponding objective function values, and it is these solutions that are most likely to survive and to combine to form new solutions that inherit their characteristics. Selective pressure, favoring the fittest solutions, is balanced against genetic diversity, with the mutation operator introducing random variation in the solution population. The goal is for the algorithm to reach a point at which the population collectively takes on the characteristics of an optimal solution. This point might be difficult to reach in practice, but at the very least genetic algorithms can be used to find good solutions to difficult optimization problems in a reasonable amount of time.

One of the greatest strengths of genetic algorithms is their versatility, owing to the fact that they do not assume that the problem being solved has any special properties or any particular structure. Thus, for problems for which there is no optimization method that can exploit its specific characteristics, genetic algorithms might be the best approach. However, if such a method does exist, then it is probably quicker and less computationally intensive than genetic algorithms.

BEYOND TRADITIONAL OPTIMIZATION: CONSTRAINT PROGRAMMING

There are other types of business problems that do not, strictly speaking, manifest themselves as optimization problems because there is no compelling objective to maximize or minimize. Instead the focus is on finding one or more (or all) solutions that satisfy the stated constraints. Not surprisingly these are termed "constraint satisfaction problems (CSPs)."

A CSP is defined solely by its decision variables, their respective "domains" (or sets of permissible values), and the constraints. Therefore, in theory you could simply describe a CSP as an optimization problem in which the objective function is a constant, unchanged by changes in the values of the decision variables, and solve it with standard methods. Unfortunately, in CSPs both the decision variables and the constraints often have properties that create significant difficulties for the use of traditional optimization methods to describe and solve the problem.

One common difficulty concerns the domain for each decision variable. Usually, decision variables in CSPs are defined over finite domains (there is a finite number of permissible values) but these domains might not be something easy to depict such as ranges of consecutive integers (whole numbers). For example, a domain might contain all even numbers from 0 to 20 except for 14. For scheduling problems, which make up a large percentage of CSPs, the decision variables might be multidimensional. For example, the decision variables might represent the starting time of each activity in the schedule and the resource assigned to each activity at the indicated start time. Many such

domains cannot be represented effectively with traditional optimization models. Others require so much additional work that the modifications obscure the basic structure of the problem, making it more difficult to understand and potentially much more difficult to solve.

Constraints in CSPs present their own challenges to traditional optimization modeling and solution methods since they can comprise intricate logical restrictions or relationships that are cumbersome to represent in a traditional optimization model. In scheduling and sequencing problems, which constitute a substantial segment of CSPs, specialized constraints specifying that events must occur at different times or with a "lag" time of at least a certain duration elapsing between them can also place unreasonable burdens on the traditional optimization methods. The resulting optimization models might be possible to construct, but they are often far too large to solve effectively.

A class of solution methods that works well with CSPs is referred to as "constraint programming" and differs very significantly from the traditional optimization approach of improving a single solution iteratively. Constraint programming instead builds a solution element-by-element, assigning a value to one decision variable at a time and tracking the effect of each such assignment on the permissible values for the unassigned decision variables. This is accomplished via a technique known as "constraint propagation" since assigning a value to one decision variable affects only the decision variables that appear in one or more constraints with the just-assigned decision variable. For example, assume that x and y are decision variables and each can take on an integer value between 1 and 5. Also assume that the constraint $x + 2y \leq 10$ is part of the CSP. If you assign a value of $x = 4$, then the domain for y is reduced to integers between 1 and 3; any greater value for y violates the constraint.

For larger sets of decision variables and constraints, this is a much more complex process. It is also possible that at some point the values assigned so far for decision variables will eliminate, via constraint propagation, all of the permissible values for at least one of the unassigned variables. Accordingly, constraint programming also includes backtracking methods for dealing with such conflicts and consistency techniques, selection strategies, and assignment strategies for avoiding conflicts. Constraint programming also includes techniques for expressing constraints and domains that are attuned to the characteristics of these elements of CSPs. The end result is that a constraint programming formulation of a CSP is often much smaller and more direct than other formulations such as an equivalent formulation that uses integer programming methods. Constraint programming is adept at producing feasible solutions to complex CSPs.

OPTIMIZATION WITH SAS SOFTWARE

SAS/OR: OPERATIONS RESEARCH SOFTWARE

SAS/OR software, part of SAS software since 1983, brings together many of the analytical modeling and solution methods that are referred to collectively as "operations research." These techniques share an outlook that advocates understanding the details of a process and using that knowledge to improve decisions and performance. Major areas of operations research work addressed by SAS/OR software include the following:

- **mathematical optimization:** support for building and solving a broad range of optimization models
- **project and resource scheduling:** critical-path and resource-constrained project scheduling, which determines when to perform individual tasks that are linked by precedence or hierarchical relationships (or both). Resource requirements must be satisfied while working within limits on resource availability, and deadlines must be met.
- **discrete event simulation:** studying the performance of systems such as customer service operations, manufacturing processes, and traffic handling for which critical elements—such as arrival of customers or vehicles, processing or service times, or other elements—exhibit random variation. Simulation enables such systems to be studied in a modeling environment in which the long-term effects of alternative configurations and policies can be measured, analyzed statistically, and compared.

Optimization has been a key part of SAS/OR software from its inception. Long-standing SAS/OR optimization procedures include:

- **the LP procedure:** linear, integer, and mixed-integer optimization based on the simplex solution method
- **the INTPOINT procedure:** linear optimization using an interior point solution method

- **the NETFLOW procedure:** network flow optimization for problems that involve flows between nodes (locations) along arcs (node-to-node connections). This procedure solves shortest path, minimum-cost flow, and maximum flow problems.

- **the NLP procedure:** general nonlinear optimization

These procedures accept optimization problems specified by the use of input SAS data sets, perform the desired optimization, and report the results in SAS data sets and output files. PROC NLP, due to the special nature of nonlinear optimization problems, also includes some optimization modeling capabilities. For the remainder of these procedures the optimization model is built as the input SAS data set is created, using the SAS DATA step.

RECENT TRENDS IN SAS/OR OPTIMIZATION

Over the past several years a number of patterns have emerged regarding the use of SAS/OR software for optimization. First, optimization is being used much more broadly than ever before. This is due to several causes, one being the general rise in competition among enterprises as consumers became better informed and less committed to single companies or brands. This leads to greater price competition, shrinking margins, and a greater need to extract better performance from less time and fewer resources. Thus, the user base for SAS/OR optimization capabilities is becoming much more diverse.

Second, optimization in SAS/OR software is being used more frequently in conjunction with other SAS analytics such as statistics, data mining, and forecasting. Data sets produced by these methods are used as input for optimization, which means that considerable processing must occur in order to extract the specific information needed by the optimization model and place it correctly in the input data sets for SAS/OR optimization. Within this cross-functional environment SAS/OR software is being used to solve larger and more complex optimization problems. This is a natural result of the benefits of optimization, as organizations hungry for better performance seek to expand their use of optimization in scope and scale.

More SAS/OR users are new—new to SAS/OR software itself and new to the SAS System as a whole. This means that we can no longer assume that SAS/OR users are experienced with the SAS language and adept at using the DATA step to manipulate data for input to the SAS/OR optimization procedures.

This creates several challenges for SAS/OR optimization. We must provide methods that find solutions more quickly so that larger and more complex models can be solved. We need to make SAS/OR optimization modeling less an artistic application of data manipulation techniques and more a direct translation from the conceptual models and the algebraic model formulations that optimization professionals build. We need to make it easier and more transparent to create the structure of an optimization model and populate it with the relevant data, whether the data comes from enterprise repositories, other SAS technologies, or other sources.

We also must work to shorten the “learning curve” for SAS/OR optimization by establishing greater consistency in the user interfaces provided by the various optimization procedures. We need to provide a simple basic syntax that applies, with only minor variations, to all SAS/OR optimization procedures.

WHAT’S NEW IN OPTIMIZATION WITH SAS/OR SOFTWARE

The SAS/OR research and development team has delivered a set of all-new optimization procedures that meet and exceed all of the preceding requirements. New state-of-the-art optimizers deliver excellent performance and are coupled with the powerful, accessible new OPTMODEL optimization modeling language that not only eases and shortens the initial modeling effort but also makes it far easier to reuse models. This is especially important if one person builds a model and then transfers it to someone else for ongoing implementation and support; by virtue of the transparent OPTMODEL language the transferred model is largely self-documenting.

The lineup of new procedures (several of which carry the “OPT” prefix to denote optimization) has been minimized to provide a one-to-one correspondence between a class of optimization problems and the corresponding SAS/OR procedure:

- **the OPTMODEL procedure:** optimization modeling language and access to all new optimization solvers

- **the OPTLP procedure:** linear optimization
- **the OPTQP procedure:** quadratic optimization
- **the OPTMILP procedure:** mixed integer linear optimization
- **the GA procedure:** genetic algorithms
- **the CLP procedure (experimental):** constraint programming

Among the SAS/OR procedures that carry the "OPT" prefix, the sole exception to the "one procedure, one problem type" rule is PROC OPTMODEL. This procedure is the flagship of the new family of optimization procedures and is intended as a single point of access for building and solving optimization models, regardless of the type of model. Thus, learning to use PROC OPTMODEL equips you to build and solve any type of traditional optimization model supported by SAS/OR. The remaining three "OPT" procedures (OPTLP, OPTQP, and OPTMILP) are useful if you specify your optimization model directly from an input data set and do not require the interactive modeling capabilities of PROC OPTMODEL.

The OPTMODEL, OPTLP, OPTMILP, and OPTQP procedures use new, accelerated optimization solvers that have been built using state-of-the-art optimization techniques. Their performance is excellent, enabling SAS/OR users to solve problems much more quickly and to tackle larger problems than ever before. Aside from PROC OPTMODEL (in which you can create your own optimization models), these new procedures receive optimization models specified in SAS data sets that adhere to industry-standard data formats (MPS for linear and mixed integer linear optimization, QPS for quadratic optimization). Users with models described using these formats can switch easily to using SAS/OR procedures to solve them.

The GA procedure and the experimental CLP procedure address specialized classes of problems. As such they use specialized modeling and solution methods, contrasting with the more broadly applicable methods used in the "OPT" procedures.

The next few sections take a closer look at each of the new SAS/OR optimization procedures.

PROC OPTMODEL AND THE OPTMODEL LANGUAGE

PROC OPTMODEL has been created to enable SAS/OR users to build optimization models easily, using modeling syntax and logical constructs that correspond directly to the algebraic statements used in symbolic formulations of optimization models. The OPTMODEL procedure also provides direct, targeted use of multiple input data sets, enabling much more precise use of data and close integration between optimization and SAS capabilities in data access and cleansing, predictive and descriptive analytics, business intelligence, and reporting.

The OPTMODEL procedure can access any of the new solvers created for SAS/OR software, and it can also be used as a modeling interface for the OPTLP, OPTQP, and OPTMILP procedures. This helps the new family of SAS/OR optimization procedures to deliver a consistent user interface in two ways: first by making all optimization modeling resident in one procedure, and second by structuring the other new procedures to receive input only in a well-defined, industry-standard format.

The OPTMODEL procedure includes an extensive set of commands for declaring all of the major elements of an optimization model. The most prominent elements, of course, are the decision variables, constraints, and objective. OPTMODEL also makes it easy to declare parameters, which define constant factors in the optimization model. Arrays and index sets for parameters and variables are also supported, enabling you to describe complex models with much greater clarity and simplicity. By establishing this structure in an optimization model, it is much easier to maintain a clear separation between the framework that the model establishes and the specific data that is inserted into this framework for any particular instance of the model.

PROC OPTMODEL supports a very broad range of programming statements and expressions. Input and output statements provide direct, precise, and flexible control over the use of input data in the model and enable you to output data on selected elements of the model or the optimal model solution with equal precision. Looping and control statements permit complex models to be created with few steps while preserving clarity. Expressions that include aggregation, set, and conditional operators make it far easier to define which model parameters and which decision variables are affected by a particular constraint or appear in the objective.

The OPTMODEL procedure runs interactively; each command is executed as soon as it is submitted. This enables you to explore the impact of changes to the model (for example, removing or adding constraints) on the optimal solution. You can also monitor the optimization process as it proceeds, pause at selected intervals, and make any needed adjustments to settings for the optimization solvers.

PROC OPTMODEL, via its SOLVE command, enables you to use any of the new SAS/OR optimization solvers; see the sections for PROC OPTLP, PROC OPTMILP, and PROC OPTQP for details about available linear, mixed integer, and quadratic optimization solvers that are available. OPTMODEL can itself determine an appropriate solver to use, or you can specify precisely which solver to apply, also choosing fine-tuning controls on the selected solver if you want. In fact, by using OPTMODEL's programming statements you can even program a completely customized solver, possibly using combinations of the built-in SAS/OR solvers, a custom-programmed solution method, or a combination of these.

For general nonlinear programming, PROC OPTMODEL can access a number of solvers depending upon the characteristics of the model. Three solvers (Fletcher-Reeves, limited-memory BFGS, and Polak-Ribière) are available for unconstrained nonlinear optimization, three (conjugate gradient, Newton-Raphson, and trust region) for linearly constrained nonlinear optimization, and one (sequential quadratic programming) for nonlinearly constrained nonlinear optimization.

The OPTMODEL procedure can be used as a modeling interface for the other new SAS/OR optimization procedures; this is useful if you want to use OPTMODEL to create a baseline model but prefer to make further model revisions by modifying the model data directly. The SAVE MPS and SAVE QPS commands in OPTMODEL save the current optimization model in a SAS data set that adheres to the indicated standard format, which can be input directly to the OPTLP, OPTMILP, and OPTQP procedures.

Two brief examples illustrate the intuitive nature and flexibility of the OPTMODEL language.

PROC OPTMODEL: PRODUCTION PLANNING EXAMPLE

In this example, various grades of cloth are to be produced on multiple machines for several customers. We know how many yards of each grade of cloth each customer demands along with the corresponding profit per yard (which varies according to the machine on which the cloth is made). We also have data on the available capacity of each machine and on the amount of time needed for each machine to produce one yard of each grade of cloth. The goal is to fill all of the customers' orders at maximum profit, while not exceeding the capacity of any machine.

For this problem the decision variable y_{cgm} represents the yards of cloth of grade g to produce for customer c on machine m. The parameter p_{cgm} represents the corresponding profit per yard, d_{cg} represents the demand of customer c for grade g cloth, t_{gm} represents the time needed on machine m to produce one yard of grade g cloth, and a_m represents the available capacity of machine m. With these definitions the optimization problem can be formulated symbolically as follows:

$$\begin{array}{lll} \max & \sum_{c,g,m} p_{cgm} y_{cgm} & \text{(profit)} \\ \text{subject to} & \sum_{m} y_{cgm} = d_{cg} \quad \text{for all } c,g & \text{(demand)} \\ & \sum_{c,g} t_{gm} y_{cgm} \le a_m \quad \text{for all } m & \text{(availability)} \\ & y_{cgm} \ge 0 \quad \text{for all } c,g,m & \end{array}$$

This formulation states that we seek to maximize profit, satisfy all demands, and stay within each machine's capacity.

Now let's look at the equivalent PROC OPTMODEL code for this problem. After some statements that declare the customers, grades, and machines and read in the data described above, PROC OPTMODEL describes the core of the optimization model as follows (the full program appears in Appendix 1):

```
var y{CUSTOMERS, GRADES, MACHINES} >= 0;

max obj = sum{c in CUSTOMERS, g in GRADES, m in MACHINES} profit[c,g,m]*y[c,g,m];

con req_demand{c in CUSTOMERS, g in GRADES}:
sum{m in MACHINES} y[c,g,m] = demand[c,g];

con req_avail{m in MACHINES}:
sum{c in CUSTOMERS, g in GRADES} time[g,m]*y[c,g,m] <= avail[m];
```

It is easy to see that these statements correspond directly to the symbolic formulation and are in many ways more "readable" than the formulation; after all, it is easier to understand `CUSTOMERS,GRADES,MACHINES` than `c,g,m`. The breadth, simplicity, and power of the OPTMODEL language mean that you can define optimization models clearly and compactly, with a one-to-one correspondence between its syntax and your symbolic formulation.

Supporting this transparent optimization code are the READ DATA and CREATE DATA statements of PROC OPTMODEL, which enable you to work easily with SAS data sets to provide input to optimization models and to receive output after an optimal solution has been reached. In the PROC OPTMODEL syntax for the production planning problem, we must read in data on:

- the profit associated with each combination of customer, cloth grade, and machine
- the demand by each customer for each grade of cloth
- the production time required per yard of cloth on each machine
- the available production time for each machine

The READ DATA statement in PROC OPTMODEL accomplishes this with precision and clarity. Consider the data on customer demand, which looks like this:

```
data demand;
input customer grade1 grade2 grade3 grade4 grade5 grade6;
datalines;
1 100 100 150 150 175 250
2 300 125 300 275 310 325
3 400 0 400 500 340 0
4 250 0 750 750 0 0
5 0 600 300 0 210 360
;
```

The interpretation of this data is straightforward. For example, the second observation has a value of 2 for the variable `customer` and a value of `310` for the variable `grade5`. This means that customer 2 demands 310 yards of grade 5 cloth. The READ DATA statement syntax that makes this data available to the optimization model is equally direct:

```
read data demand into CUSTOMERS=[customer]
       {g in GRADES} <demand[customer,g]=col("grade"||g)>;
```

This READ DATA statement reads the SAS data set named `demand`. The first part of the statement indicates that the values of the variable `customer` are being used to populate a set, `CUSTOMERS`, that is used by PROC OPTMODEL to store customer identifiers. This makes the set `CUSTOMERS` completely data-driven, which is useful since for any period of time the list of customers is likely to change. The second part of the statement imports demand information into the optimization model by populating the array `demand` and is really just a formal statement of the explanation of the values in the `demand` data set given in the preceding paragraph.

We use the SOLVE command in PROC OPTMODEL to invoke the primal simplex solver and identify an optimal solution, and we use the CREATE statement in PROC OPTMODEL to output that solution:

```
create data solution from [customer grade machine]
       ={c in CUSTOMERS, g in GRADES, m in MACHINES: y[c,g,m]^=0} amount=y;
```

This CREATE DATA statement creates a new SAS data set called `solution` from the optimal values of the y_{cgm} decision variables. The statement indicates first that the values for the variables `customer`, `grade`, and `machine` are to be copied directly from the corresponding values in the optimal solution. The second section of the statement creates a new variable, `amount`, which takes on the value of y_{cgm} whenever y_{cgm} has a nonzero value. Thus, we use the CREATE DATA statement to filter information about the optimal solution so that we can focus on the nonzero decision variable values. This limits the output to the actions recommended by the optimization process (the decision variables with values of zero at optimality correspond to actions that should *not* be taken).

The data set `solution`, which contains information about the optimal production plan, can now be passed to any SAS procedure for tabular or graphical reporting. This data set can be published in any format suitable for SAS data sets (the TABULATE procedure is used in this case; see Appendix 1). In short, any of the information delivery vehicles supported by SAS can be used at this point. Thus, SAS/OR optimization can integrate smoothly with SAS reporting and business intelligence.

PROC OPTMODEL: FACILITY LOCATION EXAMPLE

In this example the challenge is to determine the best locations and customer assignments for facilities in a new distribution network. Information is available about the locations and projected demand levels of the customers that will be served by the new facilities; information is also available about the locations of the possible sites for new facilities. All such facilities will have the same capacity (maximum number of units that can be supplied to customers).

For each candidate location there is a specific fixed charge that we will incur if we place a new facility at that location. This fixed charge could represent some or all of land acquisition costs, site preparation, and building costs. We must also decide which customers will be served by each facility that is built. Serving each customer incurs a per-unit variable charge equal to the distance between the facility site and the customer's location. We must satisfy each customer's demand in full, and each customer is to be served by only one facility. We must not overallocate the capacity of any facility to serve customers. The objective is to minimize total cost. We consider this problem both with and without the fixed charges so that we can gauge their effect on the optimal decisions.

For this problem there are two sets of decision variables. The decision variable b_s takes a value of 1 if a facility is built at site s and takes a value of 0 otherwise. Similarly, a_{cs} takes a value of 1 if customer c is assigned to site s and takes a value of 0 otherwise (decision variables that can be set only to 0 or 1 are referred to as *binary variables*). The parameter $dist_{cs}$ represents the distance between the location of customer c and the location of site s; the parameter dem_c represents the number of units demanded by customer c. The capacity of any facility is represented by the parameter cap, and the fixed charge for site s is f_s. The formulation of this optimization problem follows; note that we provide two versions of the objective, one including fixed charges and one excluding them.

min	$\sum_{c,s} dist_{cs} a_{cs}$		*(variable charges only)*
min	$\sum_{c,s} dist_{cs} a_{cs} + \sum_{s} f_s b_s$		*(variable charges + fixed charges)*
subject to	$\sum_{s} a_{cs} = 1$	for all c	*(one site per customer)*
	$a_{cs} \le b_s$	for all c, s	*(an assigned facility must be built)*
	$\sum_{c} dem_c a_{cs} \le cap \cdot b_s$	for all s	*(site capacities)*
	$a_{cs}, b_s \in \{0,1\}$	for all c, s	

As in the previous example, the PROC OPTMODEL statements resemble the symbolic formulation very closely. After the customary declaration statements and some statements that compute the Euclidean (straight-line) distances between customer locations and candidate sites, the following statements define the model (the full OPTMODEL program appears in Appendix 2):

```
   var Assign {CUSTOMERS, SITES} binary;
   var Build {SITES} binary;

   min CostNoFixedCharge
       = sum {c in CUSTOMERS, s in SITES} dist[c,j] * Assign[c,s];
   min CostFixedCharge
       = CostNoFixedCharge + sum {s in SITES} fixed_charge[s] * Build[s];

   /* each customer assigned to exactly one site */
   con assign_def {c in CUSTOMERS}:
       sum {s in SITES} Assign[c,s] = 1;

   /* if customer c assigned to site s, a facility must be built at site s */
   con link {c in CUSTOMERS, s in SITES}:
       Assign[c,s] <= Build[s];

   /* each site can handle at most &SiteCapacity demand */
   con capacity {s in SITES}:
       sum {c in CUSTOMERS} demand[c] * Assign[c,s] <= &SiteCapacity *
Build[s];
```

Decision variables, two versions of the objective function, and constraints are all clearly defined in the PROC OPTMODEL statements. Note especially that the definition of the total cost objective including fixed charges is able to use the previously defined “no-fixed-charges” objective in its equation.

You can run this optimization model with and without fixed charges simply by issuing a SOLVE command in PROC OPTMODEL that specifies your desired objective function. In each case the results of the optimization are written to a SAS data set using the CREATE DATA statement and are then passed to the GPLOT procedure from SAS/GRAPH® software for display. If you issue the following command, then you produce a solution in which facilities are built at all candidate sites.

```
   solve obj CostNoFixedCharge;
```

This is not surprising: since the only goal is to limit the distance-related costs of serving customers, it is advantageous to build as many facilities as possible.

The PROC GPLOT output is shown in Figure 1.

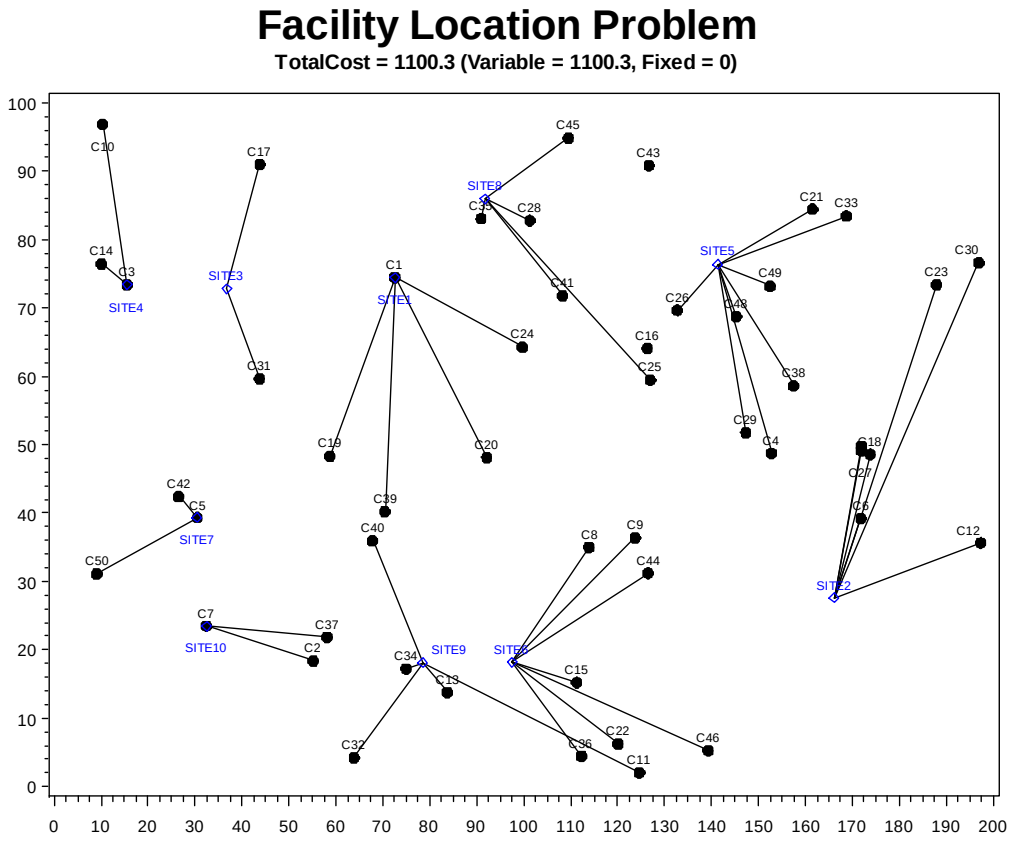

Figure 1. Facility Location Problem Solution without Fixed Charges

In Figure 1 the candidate sites are shown as blue squares and the customer locations as black dots; the line segments indicate the assignments of customers to sites.

If instead you issue the following command, then you produce an optimal solution in which the reduction in variable charges from building more facilities is balanced against the increased fixed charges from doing so.

```
solve obj CostFixedCharge;
```

The net result is that facilities are built at fewer sites, as shown in Figure 2.

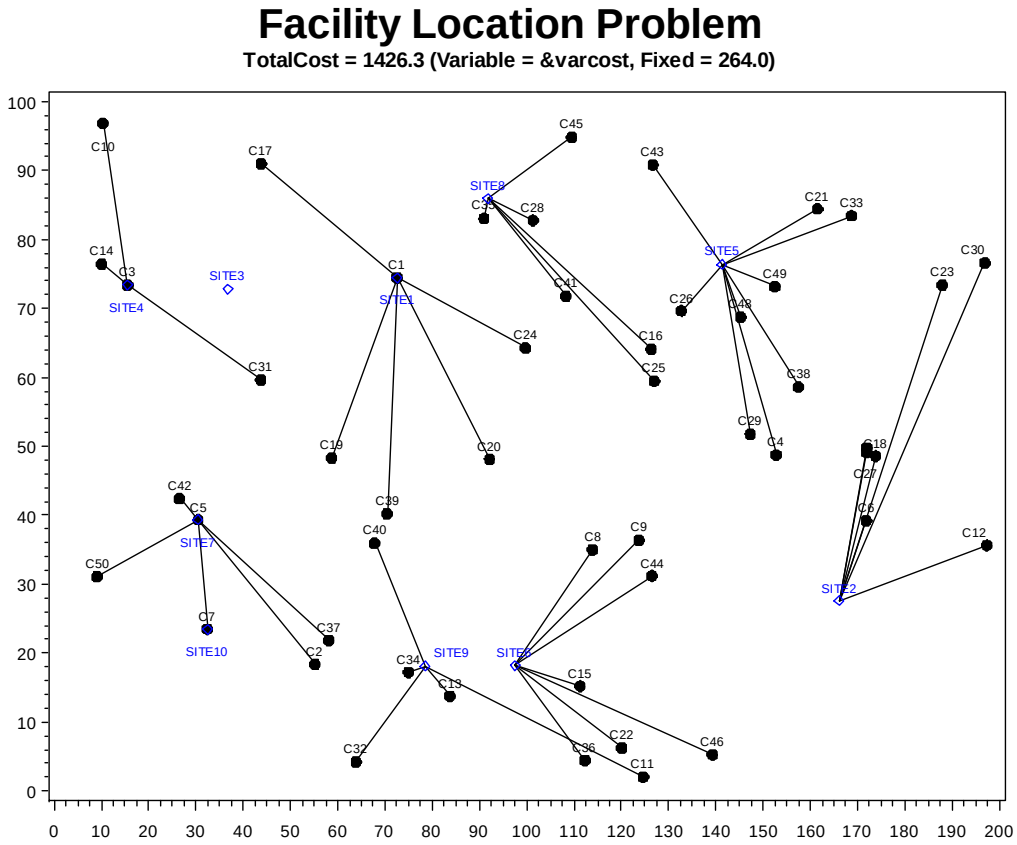

Figure 2. Facility Location Problem Solution with Fixed Charges

Note that sites 3 and 10 no longer host facilities and that the assignment of customers has been adjusted accordingly.

PROC OPTMODEL IN CONTEXT

It is important to remember that, while the OPTMODEL procedure incorporates a powerful, full-fledged optimization modeling language, it is still a SAS procedure. This means that a call of the OPTMODEL procedure can include SAS macro variables for further parameterization and can itself be included in a SAS macro. PROC OPTMODEL can use virtually any of the dozens of available SAS functions in building an optimization model. It can be combined in a SAS program with other procedure calls and SAS commands that handle the data and analytic work that precedes and supports optimization, and the entire program can be registered as a stored process and called from SAS Enterprise Guide® software or the SAS Add-In for Microsoft Office. In short, in offering an optimization modeling language within a SAS procedure that is part of the SAS language, PROC OPTMODEL gives you the best of both worlds.

The remaining new “OPT” optimization procedures in SAS/OR software have been designed to be compact and straightforward since all of the modeling capabilities have been concentrated in the OPTMODEL procedure. These procedures provide targeted access to specific classes of solvers. They are useful if you prefer to work with a single input data set that describes the entire optimization model or if you want to separate the creation of your model from the solution process.

PROC OPTLP: LINEAR OPTIMIZATION

PROC OPTLP concentrates on the solution of linear programs, offering three solvers: primal simplex, dual simplex, and an experimental interior point algorithm. It includes an aggressive presolver that can work to reduce the effective size of the optimization model (and thus the time required to solve it) before the solver begins its work. The OPTLP procedure accepts models specified in a SAS data set that uses the MPS data format, which has become a standard in linear optimization.

PROC OPTQP: QUADRATIC OPTIMIZATION

PROC OPTQP solves quadratic programs with an interior point solver. This solver is especially designed to handle large-scale problems since many quadratic programming problems originate in large corporate settings and are broad in scope. The OPTQP procedure handles both sparse and dense problems well. In a sparse problem each constraint tends to involve relatively few of the decision variables; in a dense problem many decision variables are involved in most of the constraints. For the OPTQP procedure the problem should be defined in a SAS data set that adheres to the QPS data format, an extension of the MPS format.

PROC OPTMILP: MIXED INTEGER LINEAR OPTIMIZATION

PROC OPTMILP solves mixed integer linear programs by using a branch-and-bound technique based on the simplex method. This technique involves the sequential creation and solution of a series of related linear programs, with new, modified linear programs potentially being generated at each step. The OPTMILP procedure includes a presolver for reducing the effective size of the model and also uses cutting planes and primal heuristics, two techniques that help to speed the progress of the branch-and-bound method.

PROC GA: GENETIC ALGORITHMS FOR OPTIMIZATION

PROC GA enables you to specify and solve optimization problems with a genetic algorithms approach. It supports all of the following steps in defining the problem and its solution method:

1. Specify the supporting data for the problem.
2. Specify the basic parameters of the genetic algorithm:
    a. Encoding: determining how solutions to the problem are represented (integers, real numbers, or character strings, or a combination of these).
    b. Objective function.
    c. Selection process: determines how members of the current generation of solutions are chosen to move unchanged or combine to form the next generation.
    d. Crossover operator: determines how the attributes of “parent” solutions in one generation are combined to produce “offspring” solutions in the next generation.
    e. Mutation operator: determines how random variation is introduced into offspring solutions in order to maintain genetic diversity in the solution population.
3. Generate an initial population of solutions.
4. Control the execution of the genetic algorithm and record the results.

For the selection process and the genetic operators crossover and mutation, PROC GA enables you to select from predefined methods (with parameters available for tuning) or to specify a completely customized method. The procedure offers similar flexibility in defining the encoding, specifying the objective function, creating the initial population, and setting controls on the execution of the algorithm,

PROC CLP: CONSTRAINT PROGRAMMING FOR CONSTRAINT SATISFACTION PROBLEMS

PROC CLP is an experimental finite-domain constraint programming procedure for use with constraint satisfaction problems (CSPs) featuring linear, logical, and/or scheduling constraints. It includes a compact and expressive syntax for specifying the key elements of CSPs—decision variables, decision variable domains, and constraints. It also enables you to set several parameters that control the methods used in building a solution to the CSP. PROC CLP is equipped to solve general CSPs and also includes the ability to model and solve scheduling CSPs, in which the goal is to determine a feasible schedule for a group of interrelated activities and associated resources. You can direct PROC CLP to find one solution, a specified number of solutions, or all solutions to a CSP.

MIGRATION FROM OLDER SAS/OR OPTIMIZATION PROCEDURES

New in SAS/OR 9.2, the MPSOUT= option enables you to save a linear or mixed integer linear optimization model built with PROC LP or a linear optimization model built with PROC INTPOINT or PROC NETFLOW as a SAS data set that conforms to the MPS format requirements. The data set can then be used by PROC OPTLP or PROC OPTMILP, as appropriate. This enables you to take advantage of the newest SAS/OR optimization solvers without the need to recreate prior modeling work.

CONCLUSION: THE SAS ADVANTAGE

The new SAS/OR optimization procedures provide accelerated, accessible optimization modeling and solution capabilities for a broad range of optimization problems. With more and more SAS users reaching the point of readiness (in data/analytic sufficiency and organizational maturity) for optimization and seeking to solve larger and more complex optimization problems, the advances in SAS/OR arrive at just the right time. With OPTMODEL and the other new "OPT" procedures, it is easier than ever for SAS/OR users to build optimization models from enterprise data, find optimal solutions, and report the key elements of those solutions so that they can be implemented successfully. The GA procedure enables the use of genetic algorithms, increasing the range of optimization problems that can be modeled and solved using SAS/OR software. The CLP procedure extends the reach of SAS/OR users in both optimization and project scheduling since it enables you to model and solve both general and scheduling-oriented constraint satisfaction problems.

As noted, optimization is far from being a standalone technology. Nevertheless, SAS stands alone in providing the full range of data, analytic, and business intelligence capabilities that are essential to building, solving, and applying relevant optimization models. Moreover, SAS/OR software offers the fullest range of optimization modeling and solution capabilities. With SAS software you can be confident that your optimization modeling environment has a firm analytic foundation, the simplicity and flexibility that enable you to adapt easily to changing needs, and the power not only to inform but to add insight, innovation, and credibility to the decision-making process.

APPENDIX 1: OPTMODEL PROGRAM FOR PRODUCTION PLANNING PROBLEM

```
/* Revenue per unit of each grade of cloth, per machine, per customer */
data object;
input machine customer
grade1 grade2 grade3 grade4 grade5 grade6;
datalines;
1 1 102 140 105 105 125 148
1 2 115 133 118 118 143 166
1 3 70 108 83 83 88 86
1 4 79 117 87 87 107 105
1 5 77 115 90 90 105 148
2 1 123 150 125 124 154 .
2 2 130 157 132 131 166 .
2 3 103 130 115 114 129 .
2 4 101 128 108 107 137 .
2 5 118 145 130 129 154 .
3 1 83 . . 97 122 147
3 2 119 . . 133 163 180
3 3 67 . . 91 101 101
3 4 85 . . 104 129 129
3 5 90 . . 114 134 179
4 1 108 121 79 . 112 132
4 2 121 132 92 . 130 150
4 3 78 91 59 . 77 72
4 4 100 113 76 . 109 104
4 5 96 109 77 . 105 145
;

/* Demand by each customer for each grade of cloth */
data demand;
input customer
grade1 grade2 grade3 grade4 grade5 grade6;
datalines;
1 100 100 150 150 175 250
2 300 125 300 275 310 325
3 400 0 400 500 340 0
4 250 0 750 750 0 0
5 0 600 300 0 210 360
;
```

```
/* Machine time capacities and consumption per unit of each grade */
data resource;
input machine
grade1 grade2 grade3 grade4 grade5 grade6 avail;
datalines;
1 .250 .275 .300 .350 .310 .295 744
2 .300 .300 .305 .315 .320 . 244
3 .350 . . .320 .315 .300 790
4 .280 .275 .260 . .250 .295 672
;

proc optmodel;
set CUSTOMERS;
set GRADES = 1..6;
set MACHINES;

/* parameters */
number profit{CUSTOMERS, GRADES, MACHINES} init 0;
number demand{CUSTOMERS, GRADES};
number time{GRADES,MACHINES} init 0;
number avail{MACHINES} init 0;

/* load the customer set and demands */
read data demand
       into CUSTOMERS=[customer]
       {g in GRADES} <demand[customer,g]=col("grade"||g)>;

/* load the machine set, time costs, and availability */
read data resource nomiss
       into MACHINES=[machine]
       {g in GRADES} <time[g,machine]=col("grade"||g)> avail;

/* load objective data */
read data object nomiss
       into [machine customer]
       {g in GRADES} <profit[customer,g,machine]=col("grade"||g)>;

/* the model */
var y{CUSTOMERS, GRADES, MACHINES} >= 0;

max obj = sum{c in CUSTOMERS, g in GRADES, m in MACHINES} profit[c,g,m]*y[c,g,m];

con req_demand{c in CUSTOMERS, g in GRADES}:
sum{m in MACHINES} y[c,g,m] = demand[c,g];

con req_avail{m in MACHINES}:
sum{c in CUSTOMERS, g in GRADES} time[g,m]*y[c,g,m] <= avail[m];

/* call the solver and save the results */
solve with lp/solver=primal;

create data solution
       from [customer grade machine]
       ={c in CUSTOMERS, g in GRADES, m in MACHINES: y[c,g,m]^=0}
       amount=y;
quit;

proc print data=solution; run;

proc tabulate data=solution;
class customer grade machine;
var amount;
table (machine*customer), (grade*amount);
```

```
run;
```

APPENDIX 2: OPTMODEL PROGRAM FOR FACILITY LOCATION PROBLEM

```
/* ============================================================ */
/* ======== Example - Facility Location    =================== */
/* ============================================================ */
    %let NumCustomers  = 50;
    %let NumSites      = 10;
    %let SiteCapacity  = 35;
    %let MaxDemand     = 10;
    %let xmax          = 200;
    %let ymax          = 100;
    %let seed          = 12345;

/* generate random customer locations */
data cdata(drop=c);
   length name $8;
   do c = 1 to &NumCustomers;
      name = compress('C'||put(c,best.));
      x = ranuni(&seed) * &xmax;
      y = ranuni(&seed) * &ymax;
      demand = ranuni(&seed) * &MaxDemand;
      output;
   end;
run;

/* generate random site locations and fixed charge */
data sdata(drop=s);
   length name $8;
   do s = 1 to &NumSites;
      name = compress('SITE'||put(s,best.));
      x = ranuni(&seed) * &xmax;
      y = ranuni(&seed) * &ymax;
      fixed_charge = (abs(&xmax/2-x) + abs(&ymax/2-y)) / 2;
      output;
   end;
run;


proc optmodel presolver=none;
   set <str> CUSTOMERS;
   set <str> SITES;

   /* x and y coordinates of CUSTOMERS and SITES */
   num x {CUSTOMERS union SITES};
   num y {CUSTOMERS union SITES};
   num demand{CUSTOMERS};
   num fixed_charge {SITES};

   /* distance from customer c to site s */
   num dist {c in CUSTOMERS, s in SITES}
       = sqrt((x[c] - x[s])^2 + (y[c] - y[s])^2);

   read data cdata into CUSTOMERS=[name] x y demand;
   read data sdata into SITES=[name] x y fixed_charge;

   var Assign {CUSTOMERS, SITES} binary;
   var Build {SITES} binary;

   min CostNoFixedCharge
       = sum {c in CUSTOMERS, s in SITES} dist[c,s] * Assign[c,s];
```

```
   min CostFixedCharge
       = CostNoFixedCharge + sum {s in SITES} fixed_charge[s] * Build[s];

   /* each customer assigned to exactly one site */
   con assign_def {c in CUSTOMERS}:
      sum {s in SITES} Assign[c,s] = 1;

   /* if customer c assigned to site s, then facility is built at site s */
   con link {c in CUSTOMERS, s in SITES}:
       Assign[c,s] <= Build[s];

   /* each site can handle at most &SiteCapacity demand */
   con capacity {s in SITES}:
       sum {c in CUSTOMERS} demand[c] * Assign[c,s]
       <= &SiteCapacity * Build[s];

   /* solve the milp with fixed charges */
   solve obj CostFixedCharge;


   /* clean up the solution */
   for {c in CUSTOMERS, s in SITES} Assign[c,s] = round(Assign[c,s],1E-6);
   for {s in SITES} Build[s] = round(Build[s],1E-6);

   num varcost = sum {i in CUSTOMERS, s in SITES} dist[c,s] * Assign[c,s].sol;
   num fixcost = sum {s in SITES} fixed_charge[s] * Build[s].sol;
   call symput('varcost',compress(put(varcost,12.1)));
   call symput('fixcost',compress(put(fixcost,12.1)));
   call symput('totalcost',compress(put(CostFixedCharge,12.1)));

   /* create a data set for use by gplot */
   create data CostFixedCharge_Data from
       [customer site]={c in CUSTOMERS, s in SITES: Assign[c,s] = 1}
       xc=x[c] yc=y[c] xs=x[s] ys=y[s];
   ************************************************;

   /* solve the milp with no fixed charges */
   solve obj CostNoFixedCharge;

   /* clean up the solution */
   for {c in CUSTOMERS, s in SITES} Assign[c,s] = round(Assign[c,s],1E-6);
   for {s in SITES} Build[s] = round(Build[s],1E-6);

   call symput('varcostNo',compress(put(CostNoFixedCharge,12.1)));

   /* create a data set for use by gplot */
   create data CostNoFixedCharge_Data from
       [customer site]={c in CUSTOMERS, s in SITES: Assign[c,s] = 1}
       xc=x[c] yc=y[c] xs=x[s] ys=y[s];

   quit;


title1 "Facility Location Problem";
title2 "TotalCost = &varcostNo (Variable = &varcostNo, Fixed = 0)";
data csdata;
    set cdata(rename=(y=cy)) sdata(rename=(y=sy));
run;
/* create annotate data set to draw line between customer and assigned site */
%annomac;
data anno(drop=xc yc xs ys);
    %SYSTEM(2, 2, 2);
    set CostNoFixedCharge_Data(keep=xc yc xs ys);
```

```
    %LINE(xc, yc, xs, ys, *, 1, 1);
 run;

proc gplot data=csdata anno=anno;
    axis1 label=none order=(0 to &xmax by 10);
    axis2 label=none order=(0 to &ymax by 10);
    symbol1 value=dot interpol=none
        pointlabel=("#name" height=0.7) cv=black;

    symbol2 value=diamond interpol=none
        pointlabel=("#name" color=blue height=0.7) cv=blue;

    plot cy*x sy*x / overlay haxis=axis1 vaxis=axis2;
run;
quit;

title1 "Facility Location Problem";
title2 "TotalCost = &totalcost (Variable = &varcost, Fixed = &fixcost)";
/* create annotate data set to draw line between customer and assigned site */
data anno(drop=xc yc xs ys);
    %SYSTEM(2, 2, 2);
    set CostFixedCharge_Data(keep=xc yc xs ys);
    %LINE(xc, yc, xs, ys, *, 1, 1);
run;

proc gplot data=csdata anno=anno;
    axis1 label=none order=(0 to &xmax by 10);
    axis2 label=none order=(0 to &ymax by 10);
    symbol1 value=dot interpol=none
        pointlabel=("#name" height=0.7) cv=black;

    symbol2 value=diamond interpol=none
        pointlabel=("#name" color=blue height=0.7) cv=blue;

    plot cy*x sy*x / overlay haxis=axis1 vaxis=axis2;
run;
quit;

title;
```

REFERENCES

SAS Institute Inc. 2008. *SAS/OR User's Guide: Mathematical Programming.* Cary, NC: SAS Institute Inc.
SAS Institute Inc. 2008. *SAS/OR User's Guide: Local Search Optimization.* Cary, NC: SAS Institute Inc.
SAS Institute Inc. 2008. *SAS/OR User's Guide: Constraint Programming.* Cary, NC: SAS Institute Inc.

RECOMMENDED READING

"Optimization with SAS/OR®: What It Is and How It Adds Value," Ed Hughes and Trevor Kearney, SAS Institute Inc., 2007; available from http://www.sas.com/technologies/analytics/optimization/index.html. Select "White Papers" on the right side of the page under the "How Can We Help?" header.

CONTACT INFORMATION

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

Ed Hughes, SAS/OR Product Manager
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Work Phone: 919-531-6916
E-mail: Ed.Hughes@sas.com

Trevor Kearney, Manager of Numerical Optimization
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Work Phone: 919-531-7726
E-mail: Trevor.Kearney@sas.com