Paper SAS1785-2015

Make Better Decisions with Optimization

David R. Duling, SAS Institute Inc.

ABSTRACT

Automated decision making systems are now found everywhere, from your bank to your government to your home. For example, when you inquire for a loan through a website, a complex decision process likely runs combinations of statistical models and business rules to make sure you are offered a set of options for tantalizing terms and conditions. To make that happen, analysts diligently strive to encode their complex business logic into these systems. But how do you know if you are making the best possible decisions? How do you know if your decisions conform to your business constraints? You might want to maximize the number of loans that you provide while balancing the risk among different customer categories. Welcome to the world of optimization. SAS® Business Rules Manager and SAS/OR® software can be used together to manage and optimize decisions. This presentation demonstrates how to build business rules and then optimize the rule parameters to maximize the effectiveness of those rules. The end result is more confidence that you are delivering an effective decision making process.

INTRODUCTION

In his 2012 book, 'Decision Management Systems', James Taylor identifies three types of decisions: strategic decisions, tactical decisions, and operational decisions. Strategic decisions are high value, low-volume decisions that guide the overall direction of the company, such as deciding what products to produce or what markets to enter. Tactical decisions are those focused on management and control, such as staffing and budgeting. Operational decisions are those of lower individual value and typically relate to a single customer or a single transaction, such as processing sales records, guiding call centers, delivering offers and recommendations, managing intelligent factories, and handling credit applications. (Taylor, 2012)

This paper focuses on optimization of operational decisions. Due to their high frequency, operational decisions can generate enough data to enable an optimization process. Small improvements in the decisions can lead to large improvements in the business objectives. These improvements are limited only by the imagination of the analyst. In our working example we will follow the process of optimizing a simple decision on accepting a credit application for further processing.

The general workflow is shown in Figure 1. Business analysts create business rules based on both analysis of data and implementation of business policies. These rules are deployed to production execution systems to make operational decisions. The results of these decisions generate new data on business performance such as revenue, recommendations, complaints, growth, and so on. This data is analyzed and the same analysts will then revise policies and update rules.

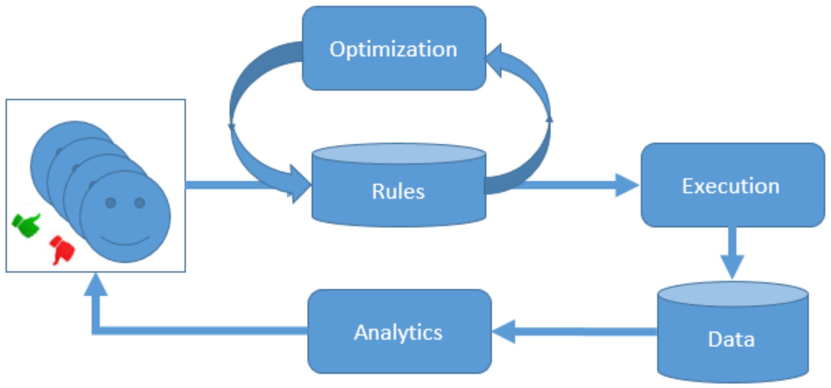

Figure 1. Business Workflow

The rules evolve over time during periods of different conditions and strategies. However, economic conditions and business policies can change rapidly. If inventory is increasing, we might need to decrease prices, reduce production, or improve deliveries. But, how do we know which actions to take? How do you know that your rules are producing the best results possible? In a complex system there can be many different possible actions and the right combination is not always obvious. In these cases, where the rules need to adapt to changes, or opportunities for improvement exist, optimization of business rules can be an excellent strategy.

BUSINESS RULES

Most businesses encode many of their operational decisions as business rules. Typically, many analysts collaborate on creating and maintaining rules that implement business strategy, regulations, taxation, logistics, and other factors into a comprehensive database. The main benefit of a rules system over a collection of code files is the ability to search, index, save and recall versions, enable lineage, and modify components of rules without editing code files. This makes the system more approachable to mainstream business analysts and more transparent to regulation and governance.

A rule set is a collection of similar rules that are used together and share a common set of input and output columns. The input columns are typically named conditions and the output columns are typically named actions. In this example rule set, we show a subset of rules needed to process credit applications. The most common display of a complex rules set is the decision table form as shown in Table 1.

Conditions			**Actions**
Age	***Income***	***Rating***	***Accept***
> 30	> 30000	1	1
> 35	> 30000	2	1
> 30	> 40000	2	1
> 40	> 70000	3	1

Table 1. Business Rules Decision Table.

When deployed to an operational system, the rule set might be expressed as SAS code:

```
ACCEPT=0;
IF AGE > 30 and INCOME > 30000 and RATING = 1 THEN ACCEPT=1;
IF AGE > 35 and INCOME > 30000 and RATING = 2 THEN ACCEPT=1;
IF AGE > 30 and INCOME > 40000 and RATING = 2 THEN ACCEPT=1;
IF AGE > 40 and INCOME > 70000 and RATING = 3 THEN ACCEPT=1;
```

In this example we see that we have several decision points for approving the application. *Condition Terms* represent all data needed to implement the business strategy. The *Rating* term is derived from analytical risk modeling processes and represents another large set of possible data points. *Action Terms* are the output of the rules and used by the operational application processing system to automate decisions.

A typical operational rules system can contain hundreds to thousands of these rules, referring to hundreds to thousands of terms. In addition, these rules might have evolved over time and reflect many generations of strategy and policy. With such a large number of rules and changing dynamics, it is not surprising that the combination of many simple rules produces a complex system that might become stale, contain inconsistencies and inefficiencies, and produce surprising or unintended results.

SAS Business Rules Manager provides a platform for creating, managing, and deploying business rules into operational systems. (SAS Institute Inc., 2014)

OPTIMIZATION

Mathematical optimization can be used to enforce constraints and find optimal combinations of parameters in well-defined systems of equations. Logistics systems typically use optimization to map out the most efficient routes for delivery of a large number packages to a large number of destinations using a minimal amount of human and capital resources. There are several requirements needed to implement an effective optimization. The first requirement is an objective function that represents the business metric such as loan approvals that should be optimized. The objective function is represented as SAS code. The next requirement is a set of parameters that can be adjusted to produce the best objective function value. The final requirement is a set of constraints. The constraints are rules that represent limits on the values of the parameters, or on combinations of values of parameters such as limits on age ranges and income ranges.

In our case, we want to optimize the rules for age and income. The first thing we need to do is to create parameters that can be used in the optimization. Table 2 shows the introduction of AGE1 and INC1 as *new parameters* and their values will be determined by the optimization process.

Conditions			**Actions**
Age	***Income***	***Rating***	***Accept***
> AGE1	> INC1	1	1
> AGE1+5	> INC1	2	1
> AGE1	> INC1+10000	2	1
> AGE1+10	> INC1+40000	3	1

Table 2. Business Rules Decision Table with Parameter Substitution.

We also need to establish some boundary constraints on our parameters. These are determined by the current business strategy and economic conditions. In our case, we want the minimum *AGE* threshold for application acceptance to be no less than 20 and no more than 30. And, for *INCOME* to be no less than 20000 and no more than 40000. Table 3 shows this parameterization.

Parameter	Min	Max
AGE1	20	30
INC1	20000	40000

Table 3. Optimization Parameters and Boundaries

The business strategy also tells us that we must establish constraints on how many applications we should accept with certain attributes. We want to ensure that no more than 50% of the applications that we accept have a credit RATING greater than 2. We also want to ensure that no more than 50% of the applications that we accept have a current MORTDUE value greater than $100,000. MORTDUE is the amount due on the applicant's current mortgage. Even though MORTDUE does not appear in the rule set, it can be part of the constraints and thus influence the rules. Table 4 shows this parameterization.

Parameter	Values	Limit
RATING	> 2	50%
MORTDUE	> $100,000	50%

Table 4. Nonlinear Constraints Configuration

We need to establish an objective function. This is described in Table 5.

Parameter	Function	Objective
ACCEPT	SUM	MAX

Table 5. Objective Function Definition

Finally, we need to select an optimization procedure for this task. The OPTLSO procedure (SAS Institute Inc., 2014) implements a derivative free optimizer (Griffin, 2010) that can operate on a wide variety of problems without extensive tuning. Business rules can be encoded in simple SAS DATA step code and can be included in both the objective function and constraint functions. Lower and upper bounds on variables and constraint functions are encoded in simple tables that can be easily modified and scripted. The optimization process combines properties of a Genetic Algorithm to search a wide area and a pattern search algorithm to accurately search small areas. This is known as a local search optimization technique and is not guaranteed to find a global optimal solution even if one exists.

Fortunately, for many business problems where design of experiments is useful, and the notion of best is well-defined, local search optimization can be used to iteratively improve the solution. In our simple example, it is not feasible to encode the business rules and the non-linear constraints into a problem that can be used with a general purpose global search technique. In such cases, the OPTLSO procedure is very appropriate for the problem. The OPTLSO procedure is part of SAS Operations Research (OR).

IMPLEMENTATION

The implementation is done entirely in SAS 9.4 code and the complete code listing is available directly from the author. The code is organized into 5 sections.

1. CREATE SAMPLE DATA

This code simply creates the data set named APPDATA used to guide the optimization. The code creates these variables: AGE, INCOME, MORTDUE, and RATING. The values of the first three variables are distributed continuously, and the value of the RATING variable is distributed as an integer ranging from 1 to 5, where 1 is the best rating and 5 is the worst. The objective function and the constraint functions used in the optimization are computed for each row of the data.

2. CREATE BUSINESS RULES

This code places the business rules into a macro for use both inside the optimization's objective function and directly inside a DATA step. The DATA step is used to deploy the rules into operational production systems. These rules could have been exported from the SAS Business Rules Manager system and then prepared for input to both the DATA step and optimization. The code used for optimization and operationalization has the parameters *AGE1* and *INC1* substituted for the original values:

```
%MACRO RULES;
  ACCEPT=0;
  IF AGE > AGE1 and INCOME > INC1 and RATING = 1 THEN ACCEPT=1;
  IF AGE > AGE1+10 and INCOME > INC1 and RATING = 2 THEN ACCEPT=1;
  IF AGE > AGE1 and INCOME > INC1+10000 and RATING = 2 THEN ACCEPT=1;
  IF AGE > AGE1+10 and INCOME > INC1+40000 and RATING = 3 THEN ACCEPT=1;
%MEND;
```

A DATA step can be used to test the initial values of the rules and produce output for comparison to the final values after optimization.

```
DATA SCORES; SET APPDATA;
  AGE1= 30; INC1= 30000;
  %RULES;
END;
```

3. CONFIGURE THE OPTIMIZATION

The OPTLSO procedure requires the creation of functions using the SAS Function Compiler (FCMP~~)~~) procedure. We can reuse the %RULES macro we defined earlier in the acceptance function definition. The OPTLSO procedure calls this function many times during the optimization process for each row of the data. Notice that the function signature includes both the optimization parameters *AGE1* and *INC1*, and also the values that are read from the data *RATING, AGE,* and *INCOME*. For more information about the FCMP procedure refer to SAS Customer Support at http://support.sas.com/. This snippet shows creation of the function for executing the business rules:

```
OPTIONS CMPLIB = WORK.BRM;
PROC FCMP OUTLIB=WORK.BRM.OPT;
   FUNCTION ACCEPTANCE (AGE1, INC1, RATING, AGE, INCOME);
      %RULES;
      RETURN (ACCEPT);
   ENDSUB;
QUIT;
```

We also need to define the objective and constraint functions called by the OPTLSO procedure. That code is similar to the code shown above and is available directly from the author.

Finally, we need to create tables that tell the procedure what to do. These data sets implement the business policies listed above. The OBJDATA table tells the procedure about the objective function that is used during the optimization. The VARDATA table tells the procedure that the solution should attempt to be constrained to the region where 20<AGE1<30 and 20000 < INC1 < 30000. The CONDATA table contains more sophisticated nonlinear constraints that are encoded as functions named RATINGC constraint and the MORTDUEC constraint:

```
DATA VARDATA;
   INPUT _ID_ $ _LB_ _UB_;
   DATALINES;
AGE1 20 30
INC1 20000 40000
; RUN;
DATA CONDATA;
   INPUT _ID_ $ _LB_ _UB_;
   DATALINES;
RATINGC 0 0.50
MORTDUEC 0 0.50
; RUN;
DATA OBJDATA;
   INPUT _ID_ $ _FUNCTION_ $ _SENSE_ $;
DATALINES;
ACCEPTS OBJFUNC MAX
RUN;
```

Now, whenever we want to change the optimization scenario, we only need to edit the code that creates those tables. To change the acceptable range of the *AGE* threshold, we edit the code that manages the *VARDATA* table. To change the allowable ratio of loans with poor ratings, we edit the code that manages the *CONDATA* table. This code makes it very easy to run many different scenarios quickly by editing or scripting the values in these tables.

4. RUN THE OPTIMIZATION

Now we can run the optimization using the OPTLSO procedure by executing the following code. The initial values of AGE1 and INC1 are generated within the range defined by the VARDATA boundaries. The final values of AGE1 and INC1 are determined by the optimization. Generations of possible solutions are managed by a custom Genetic Algorithm. At each one of these generations, a custom pattern search evaluates dozens of candidate solutions. The objective and constraint functions are evaluated multiple times for each candidate solution. The OPTLSO procedure contains a sophisticated scheme for scheduling the function evaluations in parallel in order to minimize run time.

```
PROC OPTLSO MAXGEN=100 LASTGEN=LASTGEN
   VARIABLES = VARDATA
   OBJECTIVE = OBJDATA
   NLINCON = CONDATA
   PRIMALOUT = SOLUTION
   FEASTOL = 0.00001
   ;
   READARRAY APPDATA;
   PERFORMANCE NTHREADS = 8;
RUN;
```

The procedure outputs a Solution Summary table that is shown in Figure 2. The output of the procedure shows that the final solution status indicates that the procedure terminated when the objective function converged to a single value without improvement. The solution ran 23 steps of the Genetic Algorithm and 1242 overall objective function evaluations. Infeasibility is the maximum amount of deviation of any constraint from its boundaries. A value of zero indicates that none of the final constraint evaluations returned a value outside the bounds previously established in the CONDATA table; thus indicating that our solution satisfies our business policies.

Solution Summary	
Solution Status	Function convergence
Objective	4002
Infeasibility	0
Iterations	23
Evaluations	1242
Cached Evaluations	36
Global Searches	1
Population Size	80
Seed	1

Figure 2. Final Optimization Details

Printing the solution data set shows the final values of the objective function and the constraints in Figure 3. The objective function is defined as the sum of the accepted applications and this output shows that the number is 4002. The optimal value of AGE1 is 26.95 years and the optimal value of INC1 is $37,234.94. The final value of the RATINGC constraint indicates that 50% of the acceptances have a credit rating of 1 or 2. The final value of the MORTDUEC constraint indicates that only 24% of the acceptances have a mortgage due value greater than $100,000.

Name	Value
obj	4002.00
inf	0.00
AGE1	26.95
INC1	37234.95
ACCEPTS	4002.00
RATINGC	0.50
MORTDUEC	0.24

Figure 3. Final Optimization Values

5. REPORT ON RESULTS

The final section of code prepares a report on the effectiveness of the optimization. The code scores the original data using the original business rules with the new values of AGE1 and INC1. The output table SCORES are run through steps to create plots and comparisons between the original naïve results and the optimized results. This snippet of code shows the substitution of the optimized parameters into the scoring step:

```
PROC SQL noprint;
   select _value_ into :AGE1 from solution where _id_ eq 'AGE1' ;
   select _value_ into :INC1 from solution where _id_ eq 'INC1' ;
   select _value_ into :RAT  from solution where _id_ eq 'RATINGC' ;
   select _value_ into :MORT from solution where _id_ eq 'MORTDUEC' ;
QUIT;
DATA SCORES ; set APPDATA ;
   AGE1= &AGE1. ;
   INC1= &INC1. ;
   %RULES;
RUN;
```

RESULTS

The original naïve solution sets AGE1 to 30 and INC1 to $40,000. The optimization process shows that we can move those values to 26.95 and $37,243.95. For business purposes, we can round those values to their nearest integers. Figure 4 shows the distributions of the input variables versus the accept output as shown in the four plots below. The histogram of AGE shows vertical reference lines at the optimal rule thresholds based on AGE1 (27, 32, and 37). The histogram of INCOME shows reference lines at the optimal rule threshold values based on INC1 ($37,244, $47,234, and $77,244). The histogram of MORTDUE shows a reference line at the constraint value, and the bar plot of ratings shows the number accepted in each bin. We can see a large number of loans both accepted and denied across all age groups above the value of AGE1. However, most loans above the INC1 threshold were accepted. Combined with the credit RATING, it is more important to have a higher income than to have a higher age. We can also see that most denials have a relatively low MORTDUE value, indicating that applicants with higher mortgages also tended to have higher risk.

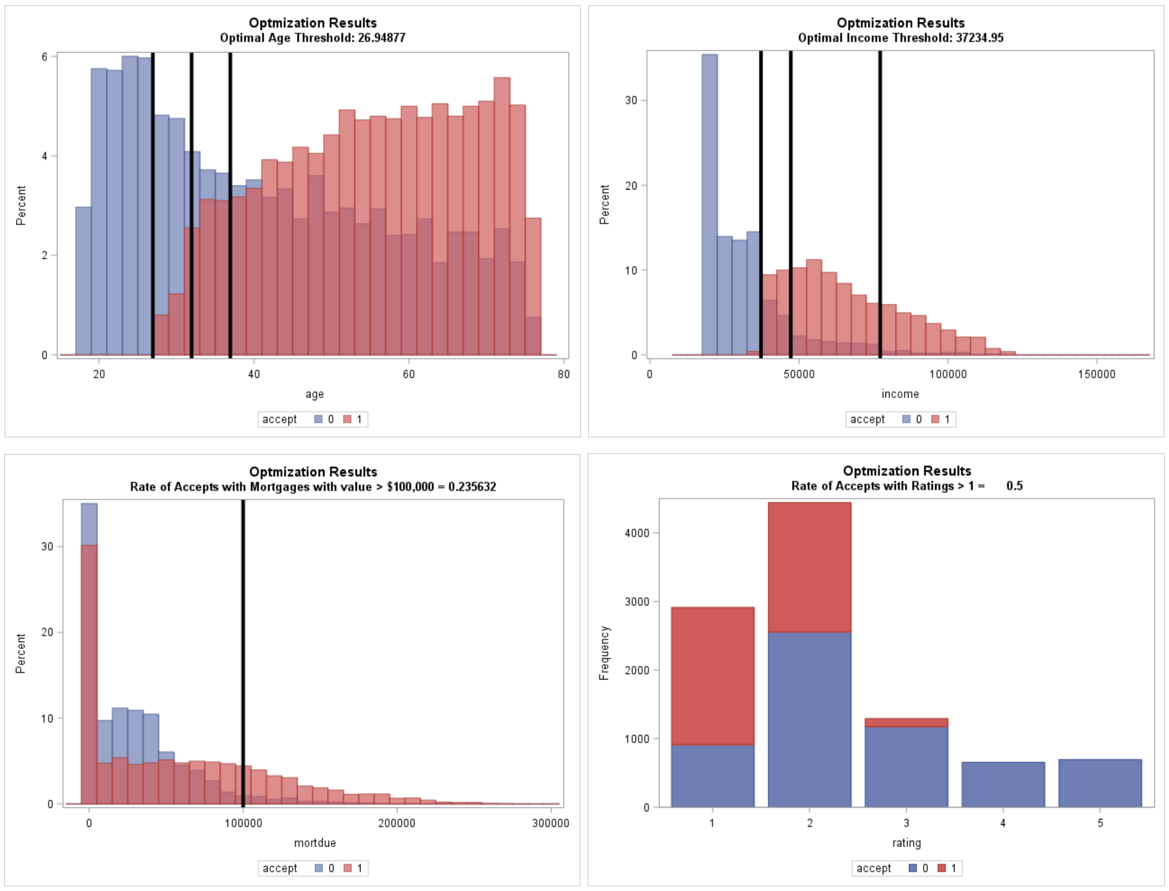

Figure 4. Plots of Input Variables versus Accepts after Optimization

Figure 5 shows the number of applications that were accepted across each credit rating category before and after the optimization. The number of accepted applications increases in categories one, two, and three after the optimization. The business rules do not permit any applications from customers with a credit rating of four or five. The total number accepted applications rose from 3643 to 4002. Accepting 359 more loans with no increased risk can have a large impact on profitability. In our case, the optimization is a success.

Accept	Rating	Before	After	change
1	1	1875	2001	+126
1	2	1668	1885	+217
1	3	100	116	+16
		3643	**4002**	**+359**

Figure 5. Accepted Applications before and after Optimization

Now, look at the tradeoff between objective function and infeasibility. Figure 6 shows a point for all 1242 solutions that were attempted during the optimization. Infeasibility is plotted on the X axis and objective function is plotted on the Y axis. Each solution generates new values of AGE1 and INC1 and then computes new values of the objective function and the constraints. The objective function is the number of applications accepted. As the solution becomes more infeasible, the objective value goes up. Our final solution shows the one that generates the greatest objective function (4002) with zero infeasibility, meaning that all constraints are satisfied. We can see how much we can improve the objective function if we change our business policy to relax some constraints. The optimization algorithm only generates candidate values of AGE1 and INC1 within their bounds. Thus, the infeasibility violations are in the nonlinear constraints RATINGC and MORTDUEC. Expanding either the age and income ranges, or the acceptable rate of high credit ratings, or the ratio or definition of high mortgage due values can lead to a higher objective function.

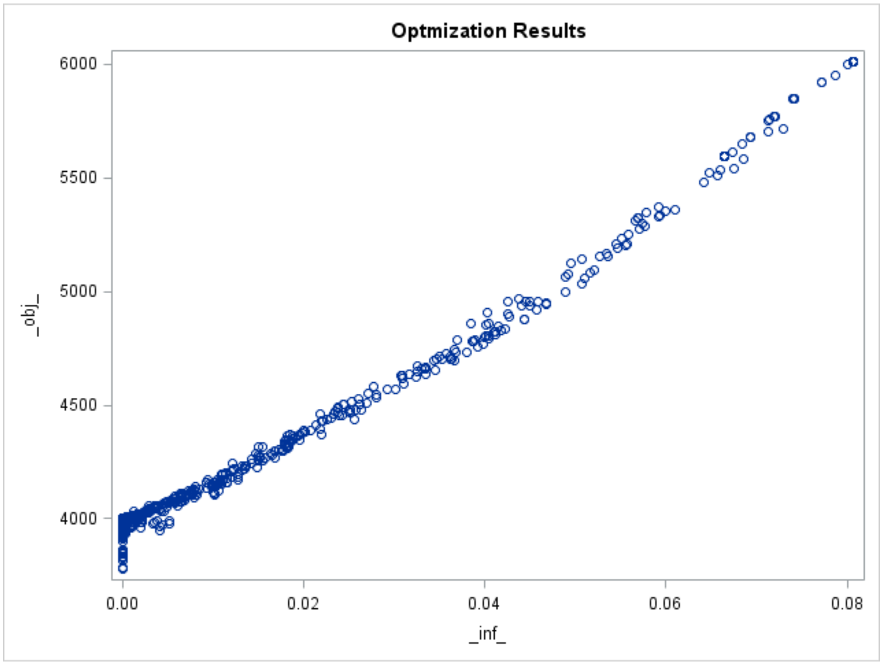

Figure 6 Objective Function (_obj_) versus infeasibility (_inf_)

CONCERNS

While the optimization in this example was a rousing success, not all optimizations will achieve that goal. Even when our initial objectives are not reached we can still use the analysis to improve our business process. Two of these scenarios are described below.

Not all problems reach a feasible solution. The OPTLSO procedure output shows this as a significant value in the infeasibility measure (_inf_). If this value had been significantly greater than zero or the number of Accepts had been disappointing, then we need to reconsider our business policies as expressed in the business rules, or reconsider the wisdom of our constraints. This is not a bad result. Infeasibility shows us that our business problem is not realistic and needs to be re-examined.

Not all problems yield improvement in the objective function from the original expectation. This can happen when the original expectation is not feasible and the optimization process finds a feasible solution with a lower objective function value. This is not a bad result. Moving a business rule set from an infeasible solution to feasible solution is extremely beneficial and produces a more robust objective function estimate that is more useful in business planning.

These concerns highlight an important point. The process of optimization can yield important knowledge about your decision making systems even if the optimization does not reach a satisfying result.

CONCLUSION

Business Rules are often the product of a mixture of data driven systems and expert systems. While optimization is a mathematical approach to improving processes. The two dissimilar systems can work together to improve operational decision making systems. SAS Business Rules Manager provides a system for analysts to collaborate on business rule development and create sequential rule sets that execute in the SAS system. SAS OR provides local search optimization that is well suited for the problems described by business rules. Analysts who understand the business needs and are able to quantify both objectives and constraints are now able to analyze and improve their processes using sophisticated optimization tools.

REFERENCES

Taylor, J. (2012). *Decision Management Systems.* Boston, MA: IBM Press.

SAS Institute Inc. (2014). *SAS Business Rules Manager 2.2: User's Guide.* Cary, NC: SAS Institute Inc.

SAS Institute Inc. (2014). *SAS/OR® 13.2 User's Guide: Local Search Optimization.* Cary, N.C.: SAS Institue Inc.

Griffin, J. D, et al. (2010). *Derivative-Free Optimization Via Evolutionary Algorithms Guiding Local Search.* Retrieved from Sandia National Labs: https://www.sim.informatik.tu-darmstadt.de/publ/download/2010-griffin_etal.pdf

ACKNOWLEDGMENTS

The author wishes to acknowledge the kind help from Joshua Griffin and Michelle Opp of SAS Institute.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. You can also request the complete code file that implements this analysis. Contact the author at:

David R. Duling
SAS Institute Inc.
Cary, North Carolina, 27513
Email: david.duling@sas.com
Phone: 919-531-5267