A PRACTICAL APPROACH TO SAMPLE-PATH SIMULATION OPTIMIZATION

Michael C. Ferris
Todd S. Munson
Krung Sinapiromsaran

Computer Sciences Department, University of Wisconsin
1210 West Dayton Street, Madison, WI 53706, U.S.A.

ABSTRACT

We propose solving continuous parametric simulation optimizations using a deterministic nonlinear optimization algorithm and sample-path simulations. The optimization problem is written in a modeling language with a simulation module accessed with an external function call. Since we allow no changes to the simulation code at all, we propose using a quadratic approximation of the simulation function to obtain derivatives. Results on three different queueing models are presented that show our method to be effective on a variety of practical problems.

KEYWORDS

Simulation optimization, nonlinear programming, quadratic approximation, modeling

1 INTRODUCTION

Simulation is a standard computational tool for understanding or predicting the behaviour of a complex system when exposed to a variety of realistic, stochastic input scenarios (Shannon, 1998). Analytic investigations of these systems are typically impossible due to the complexity of the underlying models. The simulation code can be very large, complicated, and difficult to understand, while in some cases, the source code might even be unavailable.

In many practical contexts, the simulation affords a few design parameters that can be modified to improve the performance of the system being modeled. Typically, these design parameters are constrained by other relationships, for example, budgetary or feasibility restrictions. Thus, an optimization model arises for which some of the defining relationships are the result of simulations. From the perspective of the optimization problem, the simulation is simply a function that takes the aforementioned input parameters and derives one or more output values from a simulation run.

This paper addresses a practical approach for solving such problems, allowing the optimization model to be formulated in the GAMS modeling system (Brooke et al., 1988) and the simulation to be provided essentially as a black-box routine. Our examples are drawn from problems whose parameters vary continuously, rather that discretely. The approach exploits state-of-the-art (gradient based) optimization approaches, rather than the stochastic neighbourhood search algorithms that are commonplace in the literature (Andradóttir, 1996; Andradóttir, 1995; Haddock and Mittenthal, 1992).

Two classes of methods have been commonly used to solve the continuous parametric simulation optimization, namely stochastic and deterministic optimization. Stochastic approaches estimate the optimal solution by generating a sequences $\{x_n\}$ where

$$x_{n+1} = c_\theta(x_n + a_n g(x_n, \theta))$$

for all $n \geq 1$, where $g(x_n, \theta)$ is an estimate of the Jacobian of the simulation function at x_n. The sequence a_n has infinite sum with a finite second moment. Examples include (Robbins and Monro, 1951) and (Keifer and Wolfowitz, 1952).

Another technique uses deterministic optimization to exploit a gradient evaluation in a sample-path method (Plambeck et al., 1993; Plambeck et al., 1996; Robinson, 1996). The gradient evaluation of the simulation function can be estimated using finite differences, infinitesimal perturbation analysis (Glasserman, 1991) or the derivative of a simple polynomial fitting model, as in this paper.

The first method, finite differences, is very general and easy to implement. It relies on the simulation computation of at least two proximal points. To increase the numerical accuracy of the derivative, both computed simulation points must be within a very small distance of each other. This closeness leads to difficulty of handling derivatives when the function is noisy (as can be the case for simulation runs).

The second method, infinitesimal perturbation analysis (IPA), uses one sample-path simulation run to collect gradient information and the simulation value. This method requires a modification of the simulation source code to incorporate the gradient computation during a simulation run. For this reason, and the difficulty of computing the actual gradient for each new simulation, we do not consider this approach here. Similar approaches based on automatic differentiation (AD) of the simulation code are also not used here, mainly due to the fact that source code of the simulation is required. Furthermore, in many cases, the AD codes do not provide meaningful derivatives since many simulation codes involve at least some integer variables, which are not differentiated, and complex logic.

Unlike approaches that require access to the source code, our method relies solely on executing the simulation repeatedly and building up a (small-scale) model that is fed into a standard (nonlinear) optimization code. While this approach may be inefficient from the standpoint of requiring many (potentially costly) simulation runs, we believe it is (in general) more reliable and efficient than other competing methods. Reliability derives from the fact that building the model does not require changes to the simulation code, and is carried out entirely automatically (and hence does not introduce programming error). Efficiency stems from the fact that the time to update a simulation for our approach is vastly shorter than, for example, IPA approaches, and affords the potential for parallelism in building the local model. For example, the local model can be built in parallel by executing the simulations on entirely separate processors.

This paper documents and explains our approach. We first describe the type of optimization model that we will address and explain how the simulation is incorporated into the optimization model. We then outline our procedure to generate a model of the simulation that can be used by an optimization code to solve the optimization problem. We pay particular attention to the treatment of noise in the simulation and introduce a statistical testing mechanism to determine when our model has captured the underlying simulation function excluding the noise. We detail how simulations are automatically reused in model building and justify several choices made in our experimentation. The strength of this work is to allow standard modeling and optimization tools to be easily and conveniently used to optimize existing simulation systems.

2 SIMULATION OPTIMIZATION

The aim of our work is to use pre-existing algorithms available from within a modeling language to determine an optimal system design. In this section, we discuss the mechanisms used within GAMS to communicate information about the system being considered to the solver.

We will think of the (simulated) system as a function, $S : \mathbf{R}^n \to \mathbf{R}^m$, mapping design choices, $x \in \mathbf{R}^n$, to outputs, $y \in \mathbf{R}^m$. We are then interested in solving the optimization problem:

$$\begin{array}{ll} \min & f(x,y) \\ \text{s.t.} & y = S(x) \\ & (x,y) \in B \end{array} \tag{1}$$

where $f : \mathbf{R}^{n+m} \to \mathbf{R}$ models the design quality and B specifies additional constraints on the problem variables. As a simple motivating example, consider an M/M/1 queue with exponentially distributed inter-arrival and service times. We might want to improve the service rate by providing additional training to an employee, resulting in a decrease in the amount of time customers spend waiting in the queue. An optimization problem might be to determine the amount of training to provide that minimizes total costs; that is the fixed cost for training and an additional penalty for lost customer goodwill.

We will assume that S is not available analytically; the system function is provided as an oracle that is given the design choices and produces some outputs. In order to write the optimization problem in GAMS, we need to use the external function syntax as illustrated in the following example.

```
Variables obj,
          mu  'sim. input, service rate',
          w   'sim. output, waiting time';
Equations cost,
  extcall 'external function';

Scalar c 'constant' /4.0/;

cost..     obj =e= sqr(mu-c) + w;

* declare external equation: w - S(mu) = 0
extcall..  1*mu + 2*w =x= 1;

* Select CONOPTX as the solver
option nlp = conoptx;

* Set a lower bound on service rate
mu.lo = 3.0;
```

```
* Construct the model and solve
Model mm1 /expect,cost/;
Solve mm1 using nlp minimizing obj;
```

Most of the model consists of standard GAMS syntax. The key to our approach is the exploitation of the external function interface of the modeling language that is signified by the `=x=` notation. The optimization model identifies that equation, `extcall`, as a special nonlinear constraint to be implemented by the modeler. Some explanation is required concerning the format of the external function definition, in this case,

```
extcall.. 1*mu + 2*w =x= 1;
```

The coefficients on the variables, `mu` and `w`, determine the mapping of the GAMS variables to the order required for the external function. In this case, `mu` is passed as the first input variable and `w` as the second. The right hand side is a unique identifier for the function, as any model can have many external functions. For complete details of how to interface to external functions see the GAMS documentation available at `http://webster.gams.com/extfunc/extfunc.html`.

We have now informed GAMS that there is an external function. The modeling language then calls a nonlinear optimizer, such as CONOPT (Drud, 1985) that requires evaluations for each constraint and objective function. When the solver requests an evaluation of an external function, control is passed to a user defined function with the appropriate arguments. This function knows how to call the simulation with the correct input arguments. For the example above, a simulation is called with `mu` as input, producing an output $S(\mu)$. The external function returns the value $w - S(\mu)$, that the solver will subsequently attempt to drive to zero by modifying `w` and `mu`.

The diagram in Figure 1 shows how the various pieces of our optimization model are joined together. Note that each time the nonlinear programming solver requires a function evaluation, the simulation oracle is called at the given point. The nonlinear programming solvers we use also require gradient information about each of the constraints, including the externally defined ones. The algebraically defined functions have gradients that are generated by the modeling system using automatic differentiation. For the simulation function, we construct a quadratic model of the function and use this model to produce gradient information as detailed in the next section.

3 APPROXIMATING THE SIMULATION

Standard nonlinear programming software typically requires that a user provide at least first order derivatives for each of the functions appearing in the model (1). Automatic differentiation is used to construct the derivatives for constraints defined analytically in the GAMS model. However, the system function S, is not defined analytically, but is only available as an oracle. We must therefore construct a meaningful derivative using only function evaluations. Instead of using finite differences, we advocate fitting a low order polynomial to observed simulation output.

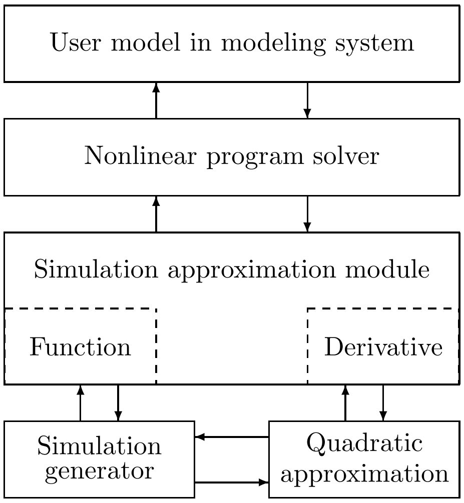

Figure 1: Overview of Simulation Optimization System

The nonlinear programming software makes a request for the derivative of the simulation function at a particular point, x_0. Our implementation evaluates the simulation at a number of random points chosen in a neighbourhood of x_0, and then fits a quadratic model, $A(x) := x^T Q x + c^T x + d$, in the least squares sense, to the observed function values. The derivative passed back to the nonlinear programming software, is $\frac{dA}{dx} := 2x^T Q + c^T$. Clearly, the appropriateness of the model depends on the error in the simulation and the size of the neighbourhood. We use a variety of statistical tests to check the validity of the model, while allowing some white noise. If the approximation is deemed poor, we reduce the size of the neighbourhood and construct a new model.

Note that while the evaluation of $S(x_0)$ may be noisy, we always return this as the function value of the simulation, rather than the value of the quadratic model at x_0. The principal reason is that regardless of the path of the algorithm, this value will always be the same (if we use the same sample-path). However, if we were to choose different random points in the neighbourhood of x_0, the quadratic model could change. Thus, $A(x_0)$ might have a different value depending upon this choice of points.

One problem with this approach is that, by assumption, the simulation evaluations are expensive. Therefore, the code collects (and stores) all previously computed values for points in the neighborhood of x_0 to reduce the number of simulation evaluations performed. If the total number of points is not enough to fit the quadratic function, then an appropriate number of uniformly distributed random points in the neighborhood of x_0 within a radius r are generated. The simulation is called to compute each of the function values at the newly generated points.

3.1 Least Squares Problem

Once all of the simulation evaluations have been collected, we fit the quadratic model in a least squares sense. Let S be the simulation function from $\mathbf{R}^n \to \mathbf{R}^m$. The quadratic approximation of S for each component $l = 1, \ldots, m$ is

$$S_l(x) \approx A_l(x) := x^T Q^l x + {c^l}^T x + d^l.$$

Let $x^1, x^2, x^3, \ldots, x^{np}$ be the sampling points used to approximate S_l, where $np \geq \frac{n(n+1)}{2} + n + 1$. The least squares problem we solve is

$$\min_{Q,c,d} \sum_{k=1}^{np} \left((x^k)^T Q x^k + c^T x^k + d - s_l(x^k) \right)^2,$$

The coefficients, Q, c and d, are the variables of the problem and that x^k is given data. Since Q is symmetric, only the upper triangular elements of Q are needed. Therefore, we minimize

$$\sum_{k=1}^{np} \left(\sum_{i=1}^{n} \left(Q_{i,i}(x_i^k)^2 + 2 \sum_{j>i}^{n} Q_{i,j} x_i^k x_j^k + c_i x_i^k \right) + d - b_k \right)^2,$$

where $b_k = s_l(x^k)$ for $k = 1, \ldots, np$.

Let $z = (Q_{11}, Q_{12}, \ldots, Q_{nn}, c_1, \ldots, c_n, d)$ with $p = \frac{n(n+1)}{2} + n + 1$ and define C^T by

$$\begin{bmatrix} x_1^1 x_1^1 & x_1^2 x_1^2 & \ldots & x_1^{np} x_1^{np} \\ 2x_1^1 x_2^1 & 2x_1^2 x_2^2 & \ldots & 2x_1^{np} x_2^{np} \\ \vdots & \vdots & & \vdots \\ 2x_1^1 x_n^1 & 2x_1^2 x_n^2 & \ldots & 2x_1^{np} x_n^{np} \\ x_2^1 x_2^1 & x_2^2 x_2^2 & \ldots & x_2^{np} x_2^{np} \\ \vdots & \vdots & & \vdots \\ 2x_2^1 x_n^1 & 2x_2^2 x_n^2 & \ldots & 2x_2^{np} x_n^{np} \\ \vdots & \vdots & & \vdots \\ x_n^1 x_n^1 & x_n^2 x_n^2 & \ldots & x_n^{np} x_n^{np} \\ x_1^1 & x_1^2 & \ldots & x_1^{np} \\ \vdots & \vdots & & \vdots \\ x_n^1 & x_n^2 & \ldots & x_n^{np} \\ 1 & 1 & \ldots & 1 \end{bmatrix}$$

The optimality conditions of the least squares problem are the normal equations

$$C^T C z = C^T b. \tag{2}$$

We use the LAPACK library (Anderson et al., 1995) to solve this symmetric (positive-definite) system of linear equations. When $m > 1$, the system is repeatedly solved, once for each different value of b.

3.2 Statistical Tests

If R^2, the coefficient of determination (Allen, 1990), is small, then the approximation is poor. We then test a null hypothesis using the Cramér-von Mises statistic (Stephens, 1974; Stephens, 1976) (herein termed the W^2 statistic) to determine if the error is due to white noise, that is noise with normal distribution and unknown mean and variance.

3.2.1 Cramér-von Mises Statistic

Given $e_1, \ldots, e_n$ from a continuous population distribution $G(\cdot)$, let $F_n(\cdot)$ be the empirical distribution function of e_i. The null hypothesis test is

$$H_0 : G(e) = F(e; \theta)$$

where $F(\cdot; \theta)$ is a given distribution (typically normal or exponential) with the parameter θ.

To test this null hypothesis, we use the Cramér-von Mises statistic computed as

$$W^2 = n \int_{-\infty}^{+\infty} \{F_n(e) - F(e;\theta)\}^2 dF(e;\theta).$$

For the normal distribution with unknown mean and variance, we have $F(\cdot;\theta) = N(\cdot; \bar{e}, s_e)$ where $\bar{e}$ is the sample mean and s_e is the sample standard deviation. We test the distribution of the difference between the simulation run and the quadratic model to determine whether it is white noise or not.

During the testing step we perform the following:

1. Sort the data (errors between simulation and quadratic model prediction) as $e_1 \leq e_2 \leq \ldots \leq e_n$.

2. Compute the sample mean $\bar{e}$ and sample standard deviation s_e.

3. Calculate $w_i = (e_i - \bar{e})/s_e$.

4. Compute $z_i = CDF(w_i)$ where $CDF(\cdot)$ is the cumulative probability of a standard normal distribution.

5. Calculate the W^2 statistic from these (sorted) z-values where
$$W^2 = \sum_{i=1}^{n}(z_i - \frac{2i-1}{2n})^2 + \frac{1}{12n}$$
6. Update W^2 to reflect the assumption of unknown mean and unknown variance, $W^2 = W^2(1 + \frac{0.5}{n})$.
7. Compare this value against the T^* statistical table (see (Stephens, 1974)). Note that $T^*_{0.15} = 0.091$, $T^*_{0.10} = 0.104$, $T^*_{0.05} = 0.126$, $T^*_{0.025} = 0.148$, and $T^*_{0.01} = 0.178$.

If W^2 is larger than T^*_α, then we reject the null hypothesis H_0 at the significance level α.

In our procedure, we fix the significance level $\alpha = 0.05$, then compute W^2 and compare it with $T^*_{0.05}$. If the null hypothesis is accepted, that is the error is due to white noise, the trend in the approximation is deemed to coincide with the actual trend of the simulation function. The quadratic approximation is then used.

If we reject this null hypothesis, i.e. $W^2 > T^*_{0.05}$, then we continue to remove any extreme values using the coefficient of skewness.

3.2.2 Coefficient of Skewness

The coefficient of skewness (Allen, 1990) is a measure of the lack of symmetry in data. If data is distributed symmetrically from the left and the right of the center point, then the skewness will have a zero value. For example, the normal distribution and uniform distribution have skewness equal to zero. When the skewness is negative, the distribution of data is more weighted to larger values than smaller values. If the skewness is positive, then data with smaller values are more prominent than data with larger values. The Chi-square distribution is one example that has positive skewness.

The skewness s_k is computed from $x_0, \ldots, x_n$ by
$$s_k = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^3}{(n-1) \times \sigma^3}$$
where σ is the sample standard deviation and $\bar{x}$ is the mean of $x_0, \ldots, x_n$.

We use the skewness to identify outliers or extreme values and use a histogram to group data into small separate groups for removal. We will consider removing the extreme values from our data set only when the skewness is outside the range of [-0.5, 0.5]. If the skewness is less than -0.5, most of the data has larger values. We reduce the radius of our problem to discard the smallest value block of our histogram which contains the extremely small values. We do the same for the largest value block of our histogram if the skewness is greater than 0.5.

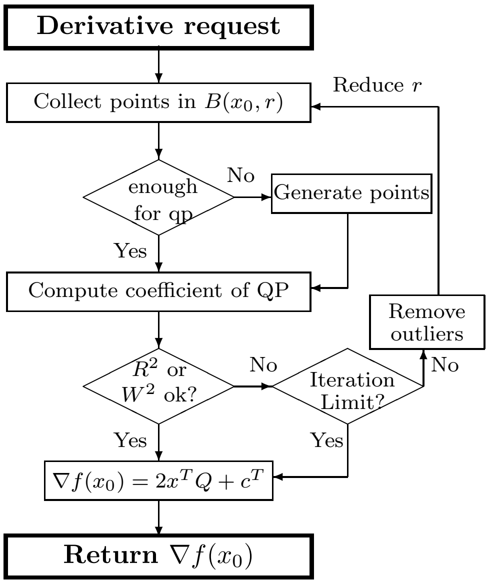

Figure 2: A Flow Chart for Computing the Derivative at a Point x_0

In the case that x_0 is one of the extreme points, the algorithm uses the current quadratic function to compute the derivative at x_0 and returns. However, the point is marked as a poor derivative value and we will attempt to construct a better quadratic model the next time an evaluation is requested at x_0.

We then repeat the procedure of approximating and performing statistical tests using the smaller radius. We limit the number of times the radius can be reduced. If we exceed the iteration limit, then we use the last quadratic approximation to compute the derivative at x_0.

3.3 Complete Derivative Computation

A detailed flow chart of the complete implementation of a derivative evaluation is given as Figure 2. This chart summarizes the information contained in this section.

4 EXAMPLES AND RESULTS

The semantics for using external functions in GAMS dictate that the user writes a function and compiles it in such a manner that the code can be linked dynamically with the solver executable. We have implemented the quadratic approximation code as outlined in the previous section for this purpose. The remaining piece is the simulation routine which is incorporated into the quadratic approximation code. As the syntax for calling the simulation routine varies, we only require that a user writes a small interfacing function that calls their simulation for a given input and returns the outputs. The function can simply call the simulation if it is available in C or FORTRAN, or it can use system calls to run an external program.

We have written such routines for three different simulations. We have incorporated these simulations into optimization problems that are formulated within GAMS. Since this paper is illustrational, we have only used very simple optimization models. However, the strength of our approach is that sophisticated simulations can be linked into complex optimization models very easily. The remainder of this section details the simulation optimizations and the results obtained on them.

4.1 M/M/1 Queue

The first problem optimizes a stable M/M/1 queue to minimize average waiting time. This problem can be solved analytically, providing us with a mechanism to check the validity of our optimization approach. The exact simulation optimization is as follows:

$$\begin{array}{ll} \min & (\mu-4)^2+w \\ \text{s.t.} & w=S(\mu) \\ & \mu \geq \lambda \end{array}$$

where $S(\mu)$ returns the average waiting time for an M/M/1 queue with service rate μ. Analytically, the average waiting time is $w=\frac{1}{\mu-\lambda}$ for an inter-arrival rate λ. For our testing, we fix the inter-arrival rate at 3. Thus, our M/M/1 simulation optimization approaching the steady state is equivalent to the problem

$$\begin{array}{ll} \min & (\mu-4)^2+w \\ \text{s.t.} & w=\frac{1}{\mu-3} \\ & \mu \geq 3 \end{array}$$

The optimal solution is at $\mu=4.297$ with an objective value of 0.859.

To test our optimization approach, we used simulations with 10000, 100000, and 1000000 customers. The first 1% of customers were ignored to avoid initial bias. Tables 1 and 2 show details of the output from the M/M/1 simulation optimization problem based on the different number of sampling points. For all of the runs, a starting value of $\mu=3.0$ was used with an initial radius of 1.0 for the quadratic model neighbourhood and $R^2=0.99999$. We ran these results on a Pentium III 600 MHz machine running WINNT 4.0.

For the same simulation length, our algorithm achieves the same optimal solution independent of the

Table 1: Comparison of M/M/1 optimization parameterized by length of simulation run and number of points sampled in quadratic model without the W^2 statistic

Simulation length	*Sampling pts. (np)*	*Runs (sim.)*	*Time (sec.)*	*Obj. value*
10000	7	195	3	0.9015
	8	194	3	0.9015
	9	214	3	0.9015
	14	211	3	0.9015
	19	213	3	0.9015
	24	411	6	0.9015
100000	7	186	25	0.8536
	8	194	25	0.8536
	9	205	26	0.8536
	14	241	31	0.8536
	19	324	42	0.8536
	24	326	42	0.8536
1000000	7	134	170	0.8586
	8	179	226	0.8586
	9	208	263	0.8587
	14	220	278	0.8586
	19	237	507	0.8586
	24	241	553	0.8586

number of sampling points. As the length of the simulation increases, the sample-path optimization solution obtained by our method converges to the correct solution as predicted by the theory. The overall solution time depends heavily on the length of the simulation run. However, these tables give no indication that the use of the W^2 statistic is beneficial. The simulation runs are long enough that the function values perceived by the optimization code are not noisy and the overall simulation function $S(\mu)$ is smooth and well-behaved. The remainder of the examples use more realistic simulation codes that indicate more benefits of the W^2 statistic.

4.2 Telemarketing Example

A more interesting example comes from simulating a telemarketing system where we have a fixed number of operators answering calls. The number of customers on hold (waiting for service) is fixed. If a customer is denied entry into the queue, they are given a busy signal and there is a probability p that they will call back after waiting an exponentially distributed amount of time. Those that do not call back result in a lost sale. We want to choose the service rate on the operators to

Table 2: Comparison of M/M/1 optimization parameterized by length of simulation run and number of points sampled in quadratic model with the W^2 statistic

Simulation length	*Sampling pts. (np)*	*Runs (sim.)*	*Time (sec.)*	*Obj. value*
10000	7	175	3	0.9015
	8	153	2	0.9015
	9	81	1	0.9016
	14	200	3	0.9015
	19	205	3	0.9015
	24	447	7	0.9015
100000	7	205	35	0.8536
	8	165	41	0.8536
	9	205	54	0.8536
	14	212	53	0.8536
	19	337	66	0.8536
	24	335	81	0.8536
1000000	7	221	514	0.8586
	8	180	405	0.8586
	9	226	524	0.8586
	14	305	702	0.8586
	19	392	900	0.8586
	24	466	1059	0.8586

minimize some weighted sum of operator training costs and lost sales. A schematic overview of the simulation is given in Figure 3.

We will assume that we have 4 operators and a fixed queue size of $M = 100$. Initially, the operators have a service rate of $\mu_i = 0.5$. The inter-arrival times of customers first entering the system is exponentially distributed with an inter-arrival rate of $\lambda = 3$. The probability that a user will call back is $p = 0.1$ with an exponentially distributed waiting time of 0.4. The variables in the optimization are the service rates which are bounded below by 0.5 and the outputs from the simulation are the the percentage of customers lost and the average waiting time.

This simulation model is very noisy due to the probability of customers leaving the system without being served. Since the simulation is coded with a single random input stream, this can lead to significant changes in simulation outputs for small variations in the input parameters. Figure 4 shows how the percentage of calls lost change as the service rate for the first operator is increased. Note the output varies dramatically for small variations in inputs, but the overall shape of the function is clear. We expect the W^2 statistic to be beneficial in solving the optimization problem, since it attempts

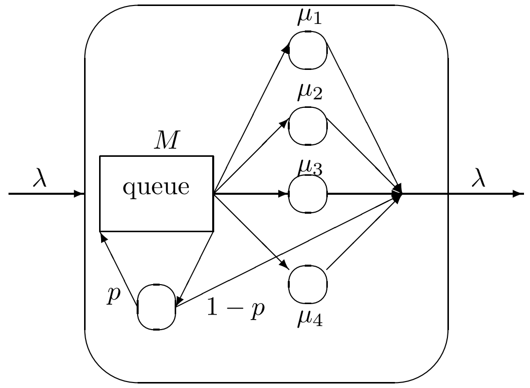

Figure 3: Telemarketing Simulation Structure

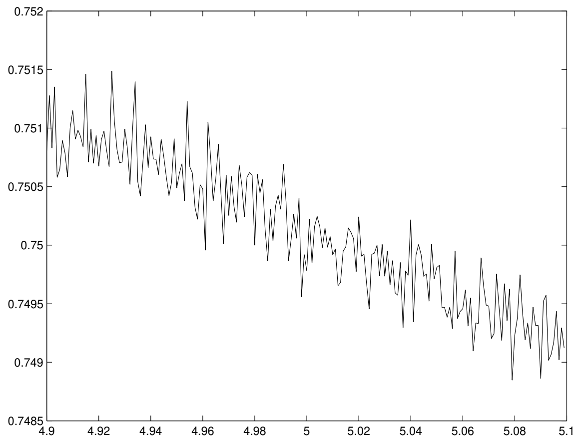

Figure 4: Percentage of Calls Lost against Service Rate for Telemarketing Simulation with 1 Million Customers

to reduce the effects of noise by generating quadratic models of the overall trend in the simulation functions.

The goal for optimizing this call-waiting simulation is to achieve the minimum number of customers lost due to the busy signal of the servers. Since the servers behave identically in the system, there are many solutions that satisfy our goal. To specify a reasonable optimization problem, we define an objective function as

$$obj = \sum_{i=1}^{4} w_i(\mu_i - l) + 100 \times lost$$

where l is the lower bound for all service rates, $lost$ is the percentage lost of customers in the system, μ_i is the service rates of i^{th} server and w_i is the weight on the i^{th} server. Different weights on each server correspond to different costs for training. For the runs presented $w_1 = 1$, $w_2 = 5$, $w_3 = 10$ and $w_4 = 15$, respectively.

Tables 3 and 4 show the details of our results based on different numbers of sampling points. Again the

Table 3: Comparison of call-waiting simulation optimization parameterized by length of simulation run and number of points sampled in quadratic model without the W^2 statistic

Simulation length	*Sampling pts. (np)*	*Runs (sim.)*	*Time (sec.)*	*Obj. value*
10000	35	582	54	1.4663
	50	887	45	1.0282
	65	885	71	1.2933
	80	1888	141	1.0277
	95	2144	160	1.0280
	110	1590	117	1.0279
100000	35	999	727	1.1303
	50	744	536	1.4641
	65	746	536	1.3358
	80	1013	722	1.1172
	95	1679	1181	1.1187
	110	1673	1201	1.1249
1000000	35	343	2993	1.5324
	50	533	4475	1.5331
	65	785	5009	1.1783
	80	501	2945	1.1707
	95	735	3940	1.1809
	110	1279	6337	1.2117

Table 4: Comparison of call-waiting simulation optimization parameterized by length of simulation run and number of points sampled in quadratic model with the W^2 statistic

Simulation length	*Sampling pts. (np)*	*Runs (sim.)*	*Time (sec.)*	*Obj. value*
10000	35	556	45	1.0281
	50	760	58	1.0280
	65	1224	93	1.0278
	80	1775	134	1.0277
	95	2517	192	1.0276
	110	1773	138	1.0278
100000	35	958	698	1.1191
	50	978	705	1.1186
	65	993	712	1.1354
	80	1013	722	1.1431
	95	1049	745	1.1214
	110	2127	1524	1.1216
1000000	35	881	7168	1.1760
	50	1696	12908	1.1655
	65	848	4698	1.2480
	80	1511	10478	1.1666
	95	1031	5580	1.1636
	110	2261	13293	1.2301

benefit of using additional sampling points to fit the quadratic model is unclear - the results with small values of np are similar to those with large values (except the smaller values execute more quickly). It appears that the results using the W^2 statistic are significantly more robust than those without. This robustness comes at some cost in terms of computing and time. While the precise solution is unknown, the optimization of the longer length simulation appears to give more accurate values for the objective value under independent simulation runs of even greater length.

4.3 Tandem Production Line

The final simulation we attempted to optimize is a tandem production line composed of m machines and $m-1$ buffers which hold the excess product between two machines arranged in series. The product arrives from an external source to the first machine and is then processed by each machine. Progress is blocked when the number of products in a buffer exceeds the maximum buffer size. The machine then waits until there is an available slot in the buffer. There is also a probability that a machine may fail at an exponentially distributed time. The time to repair a failed machine is also exponentially distributed. The input parameters to the simulation are the machine processing rates, the probability of machine failure, and the rate of repair for each machine. The output parameter is the reciprocal of throughput where the throughput is the average processing rate for the entire line. Figure 5 gives an overview of the system.

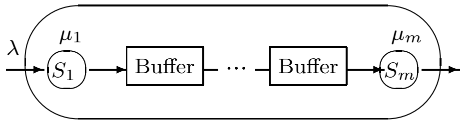

Figure 5: Tandem Production Line Simulation Structure

The actual simulation we use was provided by Erica Plambeck and is based on the Tandem production problems from (Plambeck et al., 1993). We use this paper as a basis for comparison here, and hence fix the probability of machine failure and the rate of repair to the values given in that paper. The paper contains 7 cases each with two different starting points for a total of 14 problems. Problems 1 and 2 involve 2 machines, problem 3 involves 4, problems 4 and 5 involve 6, problem 6 involves 5 machines and problem 7 involves 15. Two methods were used to obtain the results reported

Table 5: Objective comparisons between the SRO, BSO, and QSO methods

Case	*SRO*	*BSO*	*QSO*	
			without W^2	*with* W^2
1a	7.6899	7.6895	7.6899	7.6899
1b	7.7010	7.7008	7.7008	7.7008
2a	0.9638	0.9638	0.9637	0.9637
2b	1.0070	1.0070	1.0070	1.0070
3a	0.7404	0.7404	0.7404	0.7404
3b	0.7358	0.7404	0.7356	0.7357
4a	0.3956	0.3957	0.3955	0.3955
4b	0.3960	0.3960	0.3960	0.3960
5a	0.3485	0.3482	0.3465	0.3465
5b	0.3450	0.3446	0.3413	0.3413
6a		3.3956	3.3950	3.3951
6b		3.3977	3.3928	3.3928
7a		3.4065	3.4107	3.4107
7b		3.4061	3.4043	3.4054

Table 6: Tandem production line comparison for the QSO method with and without the W^2 statistic

Case	*Without* W^2 *statistics*		*With* W^2 *statistics*	
	Runs (sim.)	*Time (sec.)*	*Runs (sim.)*	*Time (sec.)*
1a	223	87	235	91
1b	240	94	351	140
2a	129	5	130	5
2b	40	2	73	3
3a	570	411	495	361
3b	737	521	666	461
4a	3074	4476	2524	3641
4b	3322	4879	1870	2733
5a	2778	244	2151	184
5b	3481	304	1962	166
6a	1827	106	2430	140
6b	1610	92	906	53
7a	****	50000	****	50000
7b	24210	28468	14443	17636

in the paper. The first method is Bundle-based stochastic optimization (BSO) which is applicable to all of the problems. The second method, single run optimization (SRO), only applies to cases 1 through 5. We obtained the simulation code from the author and used it with our optimization methodology. We use the label QSO to indicate our method.

Each simulation run uses 49500 units with 500 units to remove bias, except that 7a and 7b uses 90000 units. The starting radius for fitting the quadratic was set to be equal to the total number of machines and R^2 was set to 0.99999. Table 5 compares the optimal solutions found among three methods, BSO, SRO and QSO. We can see that the solutions found by QSO with or without the W^2 statistic are virtually indistinguishable and are all comparable to those found by SRO and BSO. In fact, on problems 5a, 5b, 6a, 6b and 7b, QSO seems to provide the best solutions of all codes. On problem 7a, QSO had more difficulties and we terminated it after it hit a time limit at a slightly worse objective value. By adding an extra constraint that constrains the sum of all the machine rates, we were able to solve 7a to optimality as well. Table 6 shows the total simulation runs and the total time used by our algorithm, with and without the W^2 statistic. These results seem to indicate that for the larger dimension problems the use of the W^2 statistic is preferable.

5 CONCLUSIONS

We have shown how to perform sample-path simulation optimization from within a modeling language. We have developed a mechanism to automatically compute a quadratic approximation (a local model) to an existing simulation using only function evaluations. The derivatives passed to the nonlinear programming software are based on this quadratic approximation. We have tested this approach on several problems and the results show this approach is useful in determining an optimal system design. We have also shown how to determine reasonable choices for the algorithmic parameters.

Furthermore, we have experimented with the use of the W^2 statistic to reduce the effects of noise in the simulation values on the optimization. The statistic is used to accept local models that are inaccurate only due to noisy simulations. This appears to give more robustness in the optimization at the cost of some computing time. Since the modeling framework also allows additional constraints to be specified, this can also be used to increase robustness of the overall procedure.

Any existing simulation can be optimized easily and effectively using the strategy developed in this paper. Complex optimization problems can be set up within existing modeling languages that incorporate simulations as an integral part of the model. The only requirement on a user is that a very small function be written that interfaces between our approximation code and the simulation. This interface routine can directly invoke the simulation if it is available in source code, or can use system calls to run the simulation as an external program.

ACKNOWLEDGMENTS

This research was partially supported by Air Force Office of Scientific Research Grant F49620-98-1-0417, National Science Foundation Grant CCR-9972372 and Microsoft Corporation.

REFERENCES

Allen, A. O. 1990. *Probability, Statistics and Queueing Theory*. Academic Press, London, second edition.

Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Cros, J. D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D. 1995. *LAPACK User's Guide*. SIAM, Philadelphia, Pennsylvania, second edition.

Andradóttir, S. 1995. A method of discrete stochastic optimization. *Management Science*, 41:1946–1961.

Andradóttir, S. 1996. A global search method for discrete stochastic optimization. *SIAM Journal on Optimization*, 6:513–530.

Brooke, A., Kendrick, D., and Meeraus, A. 1988. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, CA.

Drud, A. 1985. CONOPT: A GRG code for large sparse dynamic nonlinear optimization problems. *Mathematical Programming*, 31:153–191.

Glasserman, P. 1991. *Gradient Estimation via Perturbation Analysis*. Kluwer, Norwell, MA.

Haddock, J. and Mittenthal, J. 1992. Simulation optimization using simulated annealing. *Comput. Ind. Eng.*, 22:387–395.

Keifer, J. and Wolfowitz, J. 1952. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23:462–466.

Plambeck, E. L., Fu, B. R., Robinson, S., and Suri, R. 1993. Throughput optimization in tandem production lines via nonsmooth programming. In Schoen, J., editor, *Proceedings of the 1993 Summer Computer Simulation Conference*, pages 70–75, San Diego, CA. Society for Computer Simulation.

Plambeck, E. L., Fu, B. R., Robinson, S., and Suri, R. 1996. Sample-path optimization of convex stochastic performance functions. *Mathematical Programming*, 75:137–176.

Robbins, H. and Monro, S. 1951. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407.

Robinson, S. M. 1996. Analysis of sample-path optimization. *Mathematics of Operations Research*, 21:513–528.

Shannon, R. E. 1998. Introduction to the art and science of simulation. *WSC98 1998 Winter simulation conference proceedings*, 1:7–14.

Stephens, M. A. 1974. EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69:730–737.

Stephens, M. A. 1976. Asymptotic results for goodness-of-fit statistics with unknown parameters. *Annals of Statistics*, 4:357–369.

AUTHOR BIOGRAPHIES

MICHAEL C. FERRIS is Professor of Computer Sciences and Industrial Engineering at the University of Wisconsin–Madison. His research interests are in computational algorithms and modeling, coupled with applications in economics, structural and mechanical engineering and medicine. His work emphasizes practical solution of large scale optimization and complementarity problems. He is a member of the Mathematical Programming Society, SIAM and INFORMS. His email and web addresses are `<ferris@cs.wisc.edu>` and `<www.cs.wisc.edu/∼ferris>`.

TODD S. MUNSON is a graduate student in the Computer Sciences department at the University of Wisconsin–Madison where he researches computational methods for large scale optimization. He received a B.S. degree in computer science from the University of Nebraska–Omaha. He is a member of the Mathematical Programming Society, SIAM, and INFORMS.

KRUNG SINAPIROMSARAN is a graduate student in the Computer Sciences department at the University of Wisconsin–Madison. His research interests are in optimization of problems using modeling language systems and developing applications on the Internet. He was a lecturer at the Chulalongkorn University, Bangkok, Thailand.