Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

A TESTBED OF SIMULATION-OPTIMIZATION PROBLEMS

Raghu Pasupathy

Industrial and Systems Engineering
Virginia Tech
Blacksburg, VA 24060, U.S.A.

Shane G. Henderson

School of OR and IE
Cornell University
Ithaca, NY 14853, U.S.A.

ABSTRACT

We propose a testbed of simulation-optimization problems. The purpose of the testbed is to encourage development and constructive comparison of simulation-optimization techniques and algorithms. We are particularly interested in increasing attention to the finite-time performance of algorithms, rather than the asymptotic results that one often finds in the literature.

1 INTRODUCTION AND MOTIVATION

The Deterministic-Optimization (DO) problem is to

$$\begin{aligned} \text{minimize} \quad & g(x) \\ \text{subject to} \quad & h(x) \geq 0 \\ & x \in \mathfrak{D}, \end{aligned}$$

where $g : \mathfrak{D} \to \mathbb{R}$ and $h : \mathfrak{D} \to \mathbb{R}^d$ are the real-valued objective function and vector-valued constraint functions respectively, and $\mathfrak{D} \subseteq \mathbb{R}^q$ is an underlying set of potential solutions. In many cases h is vacuous, and the problem is then said to be unconstrained. The set $\mathfrak{D}$ is arbitrary, so it can represent any level of constraints. Typically, however, $\mathfrak{D}$ represents a natural and easily-defined set of potential solutions, such as the nonnegative orthant.

The Simulation-Optimization (SO) problem is a generalization of the DO problem where the objective is the same, i.e., a solution x^* is sought that minimizes g over $\mathfrak{D}$ while also satisfying $h(x) \geq 0$. The difference is that now g and/or h are only observable through a stochastic simulation, and therefore with error. We assume estimators $G_m(x)$ of $g(x)$, and $H_m(x)$ of $h(x)$, that are consistent, in the sense that $G_m(x) \Rightarrow g(x)$ and $H_m(x) \Rightarrow h(x)$ as $m \to \infty$, for any $x \in \mathfrak{D}$. Here $\Rightarrow$ denotes convergence in distribution, and m is some measure of simulation effort. For example, when h and g are performance measures associated with a finite-horizon simulation, m might represent the number of independent and identically distributed (i.i.d.) replications. We assume that the domain $\mathfrak{D}$ is deterministic and known.

In many cases, the SO problem possesses structure which can be used to assist in the search for a minimum. For example, if g is differentiable and one can obtain estimates of the gradients of g, then one can employ methods that exploit gradient information.

While one would like to identify a *global* minimum, this goal is, in general, exceedingly difficult to achieve, even in the case where g can be computed without simulation error. For example, if g is completely unstructured, then to guarantee that a global minimum has been found, one must compute g at every feasible solution. Accordingly, we tailor our discussion to the goal of finding *local* minima.

There has recently been a marked growth in the number and type of instances where SO problem formulations apply, often with only minor variations. There has also been a corresponding increase in the number and diversity of solutions to these SO problem variants. Recognizing this, a panel discussion at the 2000 Winter Simulation Conference (Fu et al. 2000) identified the need for a testbed of SO problems which, among other things, can be used to evaluate and compare competing SO algorithms, and to identify particular problem classes for further inquiry. This need was reiterated in Fu (2002) and in comments to that article in Glynn (2002). The use of testbeds is well-established in other fields. See Jackson et al. (1991) for examples, as part of an update to a set of guidelines (Crowder et al. 1979) on reporting computational results. For early discussion in the mathematical programming community, see Mulvey (1982).

A SO testbed is primarily motivated by the following considerations:

1. a testbed is useful in comparing the performance of competing algorithms through execution on the same problem instances;

2. a testbed helps to identify particular SO problem

classes that have defied efficient solution and in the process stimulates algorithm development for these classes;

3. a testbed increases the visibility of SO problem tools among researchers and practitioners thereby leading to their increased use.

In this paper, we describe the design and construction of a testbed of SO problems. Our aims in writing this paper are threefold:

(i) to inform the simulation community of our effort;

(ii) to obtain feedback from the simulation community on the testbed design; and

(iii) to solicit SO problems for inclusion in the testbed.

It is worth emphasizing that the testbed we propose is intended not as an archive of "unsolved" problems in SO, but instead as a forum that presents a range of SO problems to facilitate research and application. We recognize that a potential consequence of a testbed is the emergence of algorithms tailored for solving the particular instances appearing in the testbed. This negative consequence can be mitigated by ensuring that particular problem classes are not overly represented, that all problem classes contain a rich set of problems, and through the careful choice of performance measures for reporting algorithm performance. For further discussion of the merits and potential pitfalls of testbeds, see Jackson et al. (1991) and the references therein.

2 PROBLEM FORMAT

We propose three general classes of problem formats to cover the gamut of SO problems.

1. High-level Description (HD): In this format, the SO problem is specified through a detailed, verbal description of the problem. For example, the famous newsvendor problem as an SO problem in HD format is:

   A newsvendor orders a fixed quantity x of newspapers to be sold each day. The cost to the newsvendor, and the selling price respectively, of each newspaper is c and s. Unsold newspapers are salvaged each day at the unit price w. The daily demand D is Poisson distributed with mean λ. What is the quantity x that maximizes the expected profit for the newsvendor?

   Sets of values for λ, c, s and w would then be specified, corresponding to particular problem instances. An important feature of this problem is that the optimal solution is known. Another important feature is that x can be viewed as an integer-ordered variable as in the newspaper example, or as a continuous quantity if, for example, the problem relates to stocks of a perishable chemical.

2. Simulation Blackbox (SB): In SB format, one has some form of black box that, given a design point x and simulation runlength m (broadly interpreted), returns estimates of $g(x)$ and $h(x)$. The black box could be, for example, code in some general-purpose programming language, or a model encoded in a specific simulation language.

   The SB format may be useful in contexts where the model complexity defies easy problem description. As an example, in the newsvendor problem described above, suppose that the daily demand is not known to be Poisson but is instead the output of another module. Then the problem in SB format is a program that, for a given design x, uses the demand generation module to generate a random demand, and then delivers the corresponding realized profit for each day.

   An important weakness of this approach is that it is highly dependent on the black box being implementable on different platforms. This is especially important in view of the fast pace at which hardware and software platforms are evolving. Therefore, while we believe it is important to include such descriptions, they are likely to date rather quickly.

3. Objective Function–Feasible Space–Error (OFE): In the OFE format, the SO problem representation is more explicit and specified through an objective function, a set of inequalities that constitute the feasible space, and the error distribution, all provided in closed form. For example,

$$\begin{aligned} g(x) &= 2x_1^2 + x_2^2, \\ x_1x_2 &\geq 2, \\ x_1 &\geq 0, \\ x_2 &\geq 0, \\ G_m(x) &= g(x) + N(0, (x_1^2 + x_2^2)/m), \end{aligned}$$

   where $N(a, b)$ denotes a normal random variable with mean a and variance b. Here we have not specified the correlation structure of the error for different xs, or for different runlengths m. This is important, for example, in the setting of common random numbers. One such structure assumes that the errors are independent for different xs, and that for a given x, $G_m(x)$ is a sample average of m i.i.d. $N(g(x), x_1^2 + x_2^2)$ random variables, with the samples used for different runlengths being mutually independent. It is difficult to specify more general dependence structures in this format, and this is a weakness of the approach. However, problem descriptions such as this are needed in the library, because they represent problems for which much is known about the problem. Therefore, they can be used to identify particular strengths and weaknesses of algorithms.

Each problem also includes a brief description of what may be assumed about the problem. For example, in the newsvendor problem, one would clarify whether x should be viewed as continuous or integer-ordered. Furthermore, newsvendor problems are known to have certain convexity properties. One would specify whether this information can be assumed or not.

3 PROBLEM TAXONOMY

Algorithms are usually tailored to particular features of SO problems. For example, problems with discrete variables usually require quite different algorithms than problems involving continuous variables and a differentiable objective function. Accordingly, we provide a taxonomy of SO problems, similar in many respects to that provided in Barton and Meckesheimer (2006), based on important and identifiable properties of the feasible region, objective function and constraints. We plan to use the taxonomy to classify problems in the testbed.

In Figure 1 we first determine whether the set $\mathfrak{D}$ consists of categorical, integer-ordered, or continuous variables. Categorical variables are those that cannot be ordered in an appropriate way, such as variables corresponding to a choice of queue discipline. Integer-ordered variables usually represent a count of discrete entities, such as agents in a call center. Continuous variables take values in $\mathbb{R}$. If a problem contains a mix of variable types, then we classify it as categorical if it has categorical variables, or integer-ordered otherwise.

Integer-ordered problems and continuous problems are further classified as to whether they are constrained (h is non null) or unconstrained. Continuous-variable problems are also classified as to whether their objective function and constraint functions are known to be smooth (continuously differentiable) or non-smooth. In cases where it is not known whether the functions are smooth or non-smooth, the problems are classified as non-smooth.

4 ALGORITHM PERFORMANCE ON A SINGLE PROBLEM

The literature on simulation optimization is replete with proofs of convergence of algorithms. Such proofs are widely viewed as being an important feature of algorithms: given enough computational effort, one would hope that an algorithm converges to at least a locally optimal solution. Unfortunately, it is often the case, or perhaps usually the case, that the computational effort required to reach a point where such asymptotic results are informative is too large to be practical. In that case, the asymptotic results are not as useful as one might hope. In order to compare algorithms on a more appropriate time scale, it seems that we need to evaluate their finite-time performance. In this section we recommend and discuss a finite-time performance measure for use in reporting results of algorithm performance.

One often sees plots like that of Figure 2. Such plots show the estimated objective function value associated with the estimated best solution seen so far, as a function of time, where time can be measured by wall-clock time, number of objective function evaluations or otherwise.

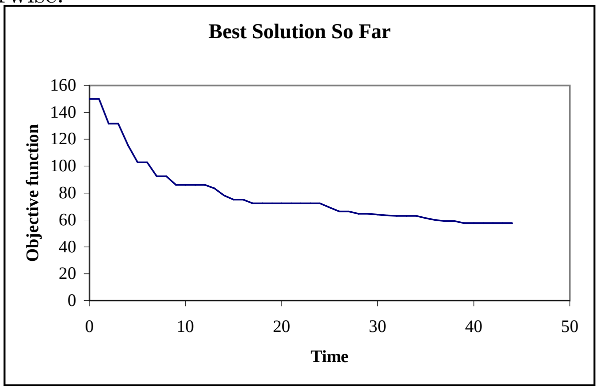

Figure 2: Estimated Objective of Estimated Best Solution

These plots are intuitive and easily constructed, but they have at least 2 important shortcomings. First, the objective function value is observed with noise, and so there is uncertainty in the curve height, given the series of estimated best solutions with time. The impact of noise is usually most clearly seen when the limiting height of the curve is below the true objective function value associated with the final estimated best solution (for a minimization problem). This is an example of the bias often associated with simulation optimization; see Mak, Morton, and Wood (1999) for an accessible overview.

Second, these curves hide the stochastic performance of the algorithm. If the optimization procedure is repeated then, due to a combination of the stochastic nature of the algorithms and the noise in the simulated function evaluations, performance can vary. One could overlay the curves resulting from multiple replications giving some idea of the true performance, but as the

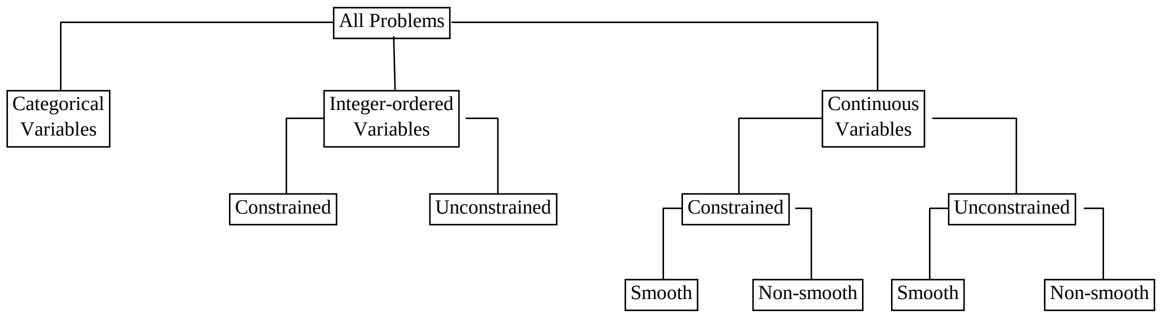

Figure 1: A SO Problem Taxonomy

optimization procedure is repeated, the curves tend to fill the plane, leading to a loss of visual information.

Another thorny issue occurs when there are constraints $h(x) \geq 0$, where h is also evaluated through simulation. In this case, the estimated best solution may, in fact, be infeasible. This is an issue even when h is deterministic, because of numerical inaccuracies. The mathematical programming community deals with this issue by specifying a tolerance, ϵ say, for each problem with constraints. Each component of h is then required to be greater than $-\epsilon$. The same tolerance is used for all constraints to keep things simple for problems with huge numbers of constraints. We adopt this approach.

For a performance measure, we propose an alternative to the above plot that more accurately reflects the stochastic performance of the algorithms. The alternative is related to the notion of goal softening as introduced in Ho, Sreenivas, and Vakili (1992).

Let Z_t be the *true* objective function value associated with the *estimated* (and therefore random) best solution seen by time t. (If a solution is infeasible, even taking into account the tolerance $-\epsilon$, then set its objective equal to some known poor value of the objective function z_{bad} say. Then Z_t could take on the value z_{bad} with positive probability, and this will show up in the plot described below. We considered setting the objective function for infeasible points equal to ∞ for a minimization problem, but this would cause difficulties in the summary statistics we describe in later sections.) Then Z_t is a real-valued random variable for each $t \geq 0$. It is random because the underlying estimated best solution is random. We are unlikely to know the true objective function values in practical simulation optimization problems, unless the objective function is not computed by simulation. However, this seems (to us) to be the right random variable to study. It is random due to random results of the simulations, and perhaps due to randomization in the optimization algorithm itself. This issue is revisited below when we discuss how to generate appropriate plots.

The distribution of Z_t, when viewed as a function of time, gives a tremendous amount of information about the performance of an algorithm. We advocate a plot that depicts this distribution as a function of time.

There are several plots which could show distributional information of Z_t as a function of time. Perhaps the simplest such plot gives the cumulative distribution function (c.d.f.) of Z_t as a function of time t. This can be depicted in a number of ways. For example, one can plot a function of t and r, where the height of the function is $P(Z_t \leq r)$, as in Figure 3. One could also plot a family of curves parameterized by r, where each curve is of the form $P(Z_t > r)$ as a function of t, for a fixed r, as in Figure 4.

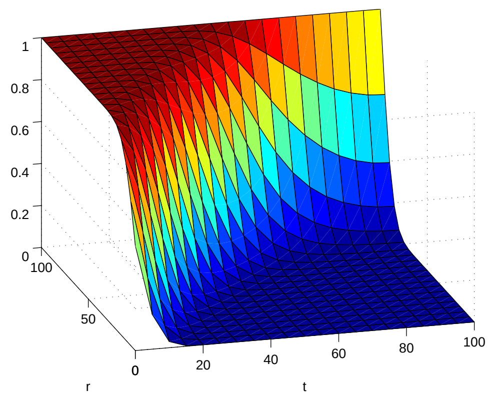

Figure 3: Hypothetical Plot of $P(Z_t \leq r)$ for a Maximization Problem

Such plots clearly showcase the performance of algorithms designed to find local versus global optima. Typically, algorithms designed to identify global optima converge more slowly than those designed to find local optima, owing to the need to search the feasible region. This slower rate typically exhibits itself through the distribution of Z_t being quite diffuse, and converging to the cdf of a point mass at a slow rate. In contrast, algorithms designed to identify local optima typically converge to the cdf of a point mass at a value that is strictly greater (for a minimization problem) than the globally optimal objective function value.

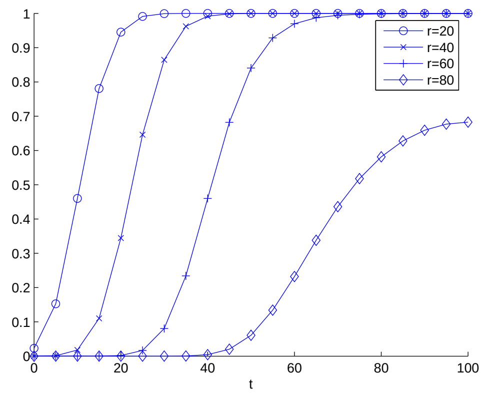

Figure 4: Hypothetical Plots of $P(Z_t > r)$ for Various r, for a Maximization Problem

Let us consider how to generate these plots. It is straightforward to run multiple replications of an optimization procedure in order to obtain the collection of solutions identified as best-so-far by time t as a function of t. However, we will not know the exact objective function value of these solutions if computing g involves simulation. One might then perform a post-processing step, where the objective function values of the estimated best-so-far solutions are estimated with great care, thereby obtaining an accurate estimate of the values that Z_t takes on, and therefore the cdf $P(Z_t \leq r)$.

A natural question arises as to how one should divide effort between the replications of the optimization procedure and the post-processing effort in order to best estimate the cdf of Z_t (for multiple t). This is a question that we will address elsewhere, perhaps adapting some of the ideas from a treatment of a similar problem (Lee 1998). For now we simply recommend that performance be reported through an estimate of the curve $(P(Z_t \leq r) : t \geq 0, r \in \mathbb{R})$, and that for best results, one should probably perform additional simulation replications after the simulation optimization experiment has finished, in order to compute the objective values needed more precisely.

How should we measure time t? One would like the results to be platform independent, while still giving some idea of the work involved in the optimization procedure. A method that is often adopted in deterministic, unconstrained, black-box optimization is to measure t in terms of evaluations of the function g. There are at least 3 issues that arise in adapting this approach to our context.

First, the evaluation of any constraints h may involve nontrivial effort. Second, for finite-horizon simulations one can simply compute the number of replications, while for infinite-horizon simulations there is no obvious analog of replications, e.g., as occurs in steady-state simulations where one typically generates a single sample path. Third, some algorithms extract additional information from the generated sample path, such as gradient-estimation schemes like infinitesimal perturbation analysis. Usually this entails a minimal amount of additional computational effort, but one can envisage situations where significant additional work is required.

In view of these issues, we recommend that the units in which t is measured be problem-specific, and specified as part of the specification of the problem.

5 ALGORITHM PERFORMANCE ON A TESTBED

In this section, we discuss summary statistics that facilitate reporting algorithm performance on a tesbed. The summary statistics we propose are derived from the trajectory of Z_t across time. Recall that Z_t is a random variable representing the true objective function value associated with the estimated best solution by time t.

5.1 Ideal Summary Statistics

Let C be a random variable representing the time at which an algorithm *achieves convergence*. Let us assume for now that the notion of *achieving convergence* has been defined in some suitable fashion. Then two natural summary statistics are the mean e^* and the variance v^* of the area under the Z_t trajectory generated by the algorithm. The measures e^* and v^* are computed as

$$e^* = E \int_0^C Z_t \, dt, \quad v^* = E \left(\int_0^C Z_t \, dt - e^* \right)^2 .$$

So, in minimization problems, algorithms with lower e^* values exhibit better performance on average. Also, low v^* values imply low variability in the performance of an algorithm.

The measure e^* has two advantages. First, it assesses algorithm performance in a hypothetical but *correct* interval, namely from time $t = 0$ to when the algorithm achieves convergence. Second, the measure naturally incorporates solution quality and convergence speed in assessing algorithm performance. In other words, through this measure, an algorithm will be assessed as performing well if it converges fast and/or produces solutions of high quality. It has the disadvantage that null algorithms that do nothing have $e^* = v^* = 0$, but such algorithms are not even contenders.

5.2 Estimable Summary Statistics

The measures e^* and v^* are very useful in concept but have an important drawback. They assume knowledge of the time C when an algorithm *attains convergence*, a notion that is difficult to precisely define. For example, for smooth problems, most definitions that one sees in the literature involve a condition such as $\|\nabla g\| \leq \epsilon$, where ∇g is the gradient of g and ϵ is a pre-specified tolerance. Since direct observations on the gradient are not available, procedures for checking the condition usually involve either a heuristic or some kind of a hypothesis testing procedure. In general, there seems to be little agreement on how to best define this notion.

For actual algorithm assessment, we therefore propose surrogate measures e and v for e^* and v^* respectively. The measures e and v are obtained by replacing C in the expressions for e^* and v^* by a user-specified problem-specific parameter t_C. The parameter t_C represents the time over which an algorithm needs to be assessed, and might represent, for instance, the available computing budget for solving the problem on hand. The measures e and v are computed as

$$e = E\int_0^{t_C} Z_t \, dt, \quad v = E\left(\int_0^{t_C} Z_t \, dt - e\right)^2.$$

Admittedly, the value of t_C is subjective, but we expect that as the testbed evolves and more algorithms are executed on a given problem, choosing t_C for any given problem will become more natural.

6 COMPARING SEVERAL ALGORITHMS ON A TESTBED

We are now in a position to compute a real-valued performance measure that describes the performance of an algorithm on a single problem. Suppose now that one wishes to compare the performance of *several* algorithms on a testbed of problems. One way to do this is via performance profiling, as introduced by Dolan and Moré (2002). Performance profiling is rapidly becoming a standard in the mathematical programming community. Unfortunately we only have space to sketch the key idea here.

Let $e(i, j)$ be the value of e for algorithm i on problem j. Let $e(j) = \min_i e(i, j)$ be the best value of e observed by any algorithm on problem j. Now compute $r(i, j)$, where

$$r(i, j) = \frac{e(i, j) - e(j)}{e_{\text{bad}} - e(j)}.$$

The value $e_{\text{bad}} = z_{\text{bad}}(j)t_C$ reflects bad performance over the interval $[0, t_C]$. The $r(i, j)$s are between 0 and 1, with smaller values being better. The $r(i, j)$s also have the nice property that they don't depend on the scaling of the objective values in the problems, although they *do* depend on the $z_{\text{bad}}(j)$s. Now plot, for each algorithm i, the empirical distribution function of $r(i, 1), r(i, 2), \ldots,$ all on the same graph. The $r(i, j)$ values were adopted from a proposal for global optimization in Montaz Ali et al. (2005).

7 PROBLEM SPECIFICATION AND EXAMPLES

A simulation-optimization problem appearing in the testbed is specified through the following characteristics, and along the lines discussed in the preceding sections:

1. problem statement (in one of the recommended formats);
2. recommended parameter settings, including the tolerance ϵ and the value z_{bad};
3. recommended measurement of time;
4. a method for obtaining a starting solution for algorithms that require a single starting point, and a method for generating a collection of starting solutions for algorithms that require a family of solutions, e.g., genetic algorithms;
5. any information about the optimal solution(s);
6. comments on any known structure in the problem;
7. recommended r values to be used when reporting algorithm performance.

We now present a few examples.

7.1 The Newsvendor Problem

(Integer-ordered variables, unconstrained.)

Problem Statement: A newsvendor orders a fixed quantity x of newspapers to be sold each day. The cost to the newsvendor, and the selling price respectively, of each newspaper is c cents and s cents. Unsold newspapers are salvaged each day at the unit price w cents. The daily demand D is Poisson distributed with mean λ. A simulation that generates random variates from the specified demand distribution is available. What is the quantity x that maximizes the expected profit for the newsvendor? Assume that the mean demand (λ) and the fact that the demand is Poisson distributed are unknown to the solution procedure.

Recommended Parameter Settings: $c = 50, s = 90, w = 10, \lambda = 100$, $z_{\text{bad}} = 0$

Starting Solution(s): 0.

Measurement of time: Number of demand random variates generated. Take $t_C = 1000, 10000$.

Optimal Solution(s): Global minimum at $\inf\{x : F(x) \geq (s-c)/(s-w)\}$ where F is the Poisson cdf with parameter λ.

Known Structure: The objective function equals 0 for $x = 0$, and tends to $-\infty$ as $x \to \infty$. The objective function is only defined on the integers, but if one extends it to $\mathbb{R}$ through the usual piecewise-linear construction, then it is concave.

Recommended Plots: $0.1(s-c)\lambda, 0.2(s-c)\lambda, \ldots, (s-c)\lambda$.

7.2 Call Center Staffing

(Integer-ordered variables, constrained.)

Problem Statement: Calls arrive to a call center according to a non homogeneous Poisson process with rate function $(\lambda(t) : 0 \leq t \leq 16)$, where t is measured in hours. Call handling (service) times are gamma distributed with mean μ_h minutes and variance σ_h^2 minutes^2. Customers are willing to wait on hold for a limited amount of time before reaching a server. If this "patience time" expires before they reach a server then they hang up (abandon) and don't call back. Patience times are also gamma distributed with mean μ_p minutes and variance σ_p^2 minutes^2. Handle times, patience times and the call arrival process are all independent of one another. The call center has T trunk lines, so that it can accommodate at most T calls either being dealt with by agents or on hold; calls that arrive when this limit has been reached receive a busy signal (and do not call back). At time $t = 16$ the call center stops receiving calls, but any calls still in the system are answered by agents until all are served or abandoned.

Agents work shifts that are structured as follows: They work for x hours, take a 1/2 hour lunch break, and then work $8 - x$ hours, where x can be 3, 3.5, 4 or 4.5. Agents can start work on the hour or on the half-hour throughout the day, starting from $t = 0$ through to $t = 8$. Agents starting at $t = 8$ finish at $t = 16.5$. There are no part-time shifts. Agents are paid a flat rate of \$ r per hour. At the end of a shift, agents finish any call they are currently handling, and then leave. (Except at the end of the day, when any queued calls are also completed before the agents leave.)

Performance is measured in each hour as follows. Let N_i be the number of calls received (even if they are blocked) in the ith hour, $i = 1, \ldots, 16$. Let S_i be the number of calls that are answered within 20 seconds. (Calls that abandon or are blocked are not counted in S_i.) The service level constraint for the ith hour is that $ES_i \geq 0.8EN_i$, $i = 1, \ldots, 16$. Notice that EN_i can be analytically computed while ES_i cannot, so that simulation is used to estimate ES_i for $i = 1, \ldots, 16$.

Let x_j be the number of agents starting shift at time $j/2$, where $j = 0, 1, \ldots, 16$. We wish to choose the vector x that minimizes costs, subject to $ES_i - 0.8EN_i \geq 0$ for each $i = 1, \ldots, 16$.

Recommended Parameter Settings: $\lambda(t) = 500 + 500\sin((3\pi t - 16\pi)/32), \mu_h = 6, \sigma_h^2 = 2, \mu_p = 2, \sigma_p^2 = 1, T = 150, r = 18$. The tolerance for the constraints is $\epsilon = 0.5$, and we take $z_{\text{bad}} = 16rT$, which would be the cost if the number of agents equalled the number of trunk lines in all periods.

Starting Solution(s): Let $x_i = 8$, for all i. If multiple random solutions are required, then let each x_j be uniformly distributed on $\{0, 1, 2, \ldots, 10\}$. (Note that x_j represents the number of agents *starting* their shift at a certain time, rather than the number of agents actually *working* at that time.)

Measurement of time: Number of simulated days of call center operation. Take $t_C = 1000, 10000$.

Optimal Solution(s): Unknown.

Known Structure: None.

Recommended Plots: Unknown.

7.3 Ambulance Bases

(Continuous variables, constrained, unknown if it is smooth or not.)

Problem Statement: Calls (i.e., requests for ambulances) arrive according to a Poisson process at constant rate λ per hour. The calls are located within the unit square $[0, 1]^2$, where distances are measured in units of 30 kilometers so that the square's area is 900 km^2. Call locations are i.i.d. with density function $(f(x, y) : 0 \leq x, y \leq 1)$ and independent of the Poisson arrival process. Scene times (time that an ambulance spends at the location of the call) are gamma distributed with mean μ_s minutes and standard deviation σ_s minutes.

There are a number $d \geq 1$ ambulances. Each ambulance travels at a constant rate of v_f km/hr on the way to a call, and at a constant rate of v_s km/hr otherwise. (These rates reflect the fact that ambulances have to slow down when going through intersections to avoid creating further accidents.) All travel is in Manhattan fashion, in the sense that when traveling from (x_1, y_1) to (x_2, y_2), the ambulance first travels from (x_1, y_1) to (x_1, y_2), i.e., vertically, and then on to (x_2, y_2), i.e., horizontally. Ambulance i has a base located at the point $b(i)$, $i = 1, \ldots, d$.

When a call arrives, the closest free ambulance travels to the call, spends some time at the scene, and is then freed for further work. If there are no available ambulances when a call is received, the call is added to a queue of calls that is answered in FIFO order. After attending a call, if there is no further work, the ambulance proceeds back to its base.

The goal is to choose the base locations that minimize the (long run) average response time (time from when a call is received until when an ambulance arrives at the scene).

Recommended Parameter Settings: f is proportional to $1.6-(|x-0.8|+|y-0.8|), \mu_s = 45, \sigma_s = 15, v_f = 60, v_s = 40$. The arrival rate λ may be selected at will, and the number of ambulances d chosen accordingly. Obviously, d has to be large enough that the system is stable. Set $z_{\text{bad}} = 60/v_f$, which is the time it takes the ambulance to travel between opposite corners of the square.

Starting Solution(s): All ambulance bases located at $(0.5, 0.5)$. If multiple initial solutions are required, then select base locations uniformly at random from within the square, independently of one another.

Measurement of time: Number of simulated hours of operation. Take $t_C = 10000$.

Optimal Solution(s): Unknown.

Known Structure: None.

Recommended Plots: Unknown.

7.4 Parameter Estimation

(Continuous variables, unconstrained, unknown if it is smooth or not.)

Problem Statement: Say a simulation generates output data $\{Y_j\}, Y_j \in [0,\infty] \times [0,\infty]$, that are i.i.d and known to come from a distribution with the two-dimensional density function

$$f(y_1, y_2; x^*) = \frac{e^{-y_1} y_1^{x_1^* y_2 - 1}}{\Gamma(x_1^* y_2)} \frac{e^{-y_2} y_2^{x_2^* - 1}}{\Gamma(x_2^*)}, \quad y_1, y_2 > 0,$$

where $x^* \equiv (x_1^*, x_2^*)$ is the unknown vector of parameters.

Noting that x^* maximizes the function

$$\begin{aligned} g(x) &= \mathrm{E}\left[\log\left(f(Y;x)\right)\right] \\ &= \int_0^\infty \log\left(f(y;x)\right) f(y;x^*)dy, \end{aligned}$$

and that

$$G_m(x) = \frac{\sum_{j=1}^m \log(f(Y_j;x))}{m}$$

is a consistent estimator of $g(x)$, find x^*.

Recommended Parameter Settings: Use $x^* = (2,5)$ to generate the data. Take z_{bad} as the true objective function value associated with taking $x = (1,1)$.

Starting Solution(s): Take $x = (1,1)$. If multiple solutions are required then select them as i.i.d. uniform in the open square $(0,10) \times (0,10)$.

Measurement of time: Number of output data points generated. Take $t_C = 1000, 10000$.

Optimal Solution(s): Global maximum at $x^* = (2,5)$.

Known Structure: None.

Recommended Plots: Unknown.

7.5 Rosenbrock's Function

(Continuous variables, constrained, smooth.)

Problem Statement:

$$\begin{aligned} \text{Min } g(x) &= \sum_{j=1,3,\ldots,2q-1} \left[(1-x_j)^2 + 100(x_{j+1} - x_j^2)^2\right], \\ |x_j| &\leq 10 \text{ for } j = 1,2,\ldots,2k, \\ G_m(x) &= g(x) + \sum_i = 1^m N_i(0, (1 + \sqrt{g(x)})), \end{aligned}$$

where the N_is are independent normal random variables that are independent at different x values.

Recommended Parameter Settings: Choose q at will. Take z_{bad} as $Eg(U)$, where U is a $2q$-dimensional random vector uniformly distributed as described next for starting solutions.

Starting Solution(s): Uniformly distributed in the hypercube $[-10,10] \times [-10,10] \times \cdots \times [-10,10]$.

Measurement of time: Number of random variates generated. Take $t_C = 10000$.

Optimal Solution(s): Global minimum at $(1,1,\ldots,1)$.

Known Structure: Objective function is nonnegative and infinitely differentiable everywhere.

Recommended Plots: $0, q, 10q, 100q, 1000q, 10000q$.

ACKNOWLEDGMENTS

We would like to thank Michael Saunders for helpful discussions. This work was partially supported by National Science Foundation Grant DMI 0400287.

REFERENCES

Barton, R. R., and M. Meckesheimer. 2006. Metamodel-based simulation optimization. In *Handbook of Simulation*, ed. S. G. Henderson and B. L. Nelson, 535–574. Elsevier.

Crowder, H. P., R. S. Dembo, and J. M. Mulvey. 1979. On reporting computational experiments with mathematical software. *ACM Transactions on Mathematical Software* 5:193–203.

Dolan, E. D., and J. J. Moré. 2002. Benchmarking optimization software with performance profiles. *Mathematical Programming, Series A* 91:201–213.

Fu, M. C. 2002. Optimization for simulation: theory vs. practice. *INFORMS Journal on Computing* 14:192–215.

Fu, M. C., S. Andradóttir, J. S. Carson, F. G. J. P. Kelly, and S. M. Robinson. 2000. Integrating optimization and simulation:research and practice. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 610–616. Piscataway NJ: IEEE.

Glynn, P. W. 2002. Additional perspectives on simulation for optimization. *INFORMS Journal on Computing* 14:220–222.

Ho, Y. C., R. Sreenivas, and P. Vakili. 1992. Ordinal optimization of discrete event dynamic systems. *Journal of Discrete-Event Dynamic Systems* 2 (2): 61–88.

Jackson, R. H. F., P. T. Boggs, S. G. Nash, and S. Powell. 1991. Guidelines for reporting results of computational experiments. report of the ad hoc committee. *Mathematical Programming* 49 (3): 413–425.

Lee, S. H. 1998. *Monte Carlo computation of conditional expectation quantiles*. Ph. D. thesis, Stanford University, Stanford, CA.

Mak, W.-K., D. P. Morton, and R. K. Wood. 1999. Monte Carlo bounding techniques for determining solution quality in stochastic programs. *Operations Research Letters* 24:47–56.

Montaz Ali, M., C. Khompatraporn, and Z. B. Zabinsky. 2005. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization* 31:635–672.

Mulvey, J. M. (Ed.) 1982. *Evaluating mathematical programming techniques*, Volume 199 of *Lecture Notes in Economics and Mathematical Systems*. New York: Springer-Verlag.

AUTHOR BIOGRAPHIES

RAGHU PASUPATHY is an assistant professor in the Industrial and Systems Engineering department at Virginia Tech. His current research interests include simulation optimization, stochastic root finding and simulation methods. He is a member of INFORMS and IIE.

SHANE G. HENDERSON is an associate professor in the School of Operations Research and Industrial Engineering at Cornell University. He has previously held positions at the University of Michigan (Ann Arbor) and the University of Auckland. He is the simulation area editor at *Operations Research*, and an associate editor for the *ACM Transactions on Modeling and Computer Simulation* and *Operations Research Letters*. He likes cats but is allergic to them. His research interests include discrete-event simulation and simulation optimization.