Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

SIMULATION MODELING, EXPERIMENTING, ANALYSIS, AND IMPLEMENTATION

Lee Schruben

University of California, Berkeley
Industrial Engineering and Operations Research,
Berkeley, CA, 94720 USA

ABSTRACT

Textbooks sometimes describe building models, running experiments, analyzing outputs, and implementing results as distinct activities in a simulation project. This paper demonstrates advantages of combining these activities in the context of system performance optimization. Simulation optimization algorithms can be improved by exploiting the ability to observe and change literally *anything* at any time while a simulation is running. It is also not necessary to stop simulating candidates for the optimal system before starting to simulate others. The ability to observe and change many concurrently running simulated systems considerably expands the possibilities for designing simulation experiments. Examples are presented for a range of simulation optimization algorithms including randomized search, directional search, pattern search, and agent-based particle swarm optimization.

1 INTRODUCTION

Simulation projects involve modeling, experimentation, analysis, and implementation. Some simulation textbooks describe these as distinct activities (see the blocks in the project workflow graphs in Banks et al. (2010) and Law (2007)). However, they can be integrated to provide new opportunities for designing simulation experiments. Simulation response optimization experiments are used here for examples, but many of the basic ideas apply more broadly.

This paper focuses on two characteristics that are unique to experimenting with simulated systems: (1) outputs can be estimated and inputs can be changed while running a simulation program, and (2) it is not necessary to stop running one simulation model to start simulating others. These are far more powerful than simply building or changing a simulation interactively while it is running (Schruben 1992).

It is common practice to run simulation experiments sequentially or independently on multiple processors (Kleijnen 2008). For each run, the input factor settings are held fixed at points in an experimental design. Independently simulating each design point fails to take full advantage of the above two features of simulation models. Instead of simulating only one design point in each run, multiple design points can be simulated together. An entire experimental design can conceptually be run simultaneously by simply indexing every variable and event (or activity or process) in the simulation model with the design point being simulated. This "array of simulation models" can then be executed in a single run. (This not suggested in practice; see Schruben (2010) for other ways to embed experiments inside a single simulation model.) This paper emphasizes ways the outputs from each concurrently executing simulation can be used to guide changes to the inputs for the others, to their mutual benefit, all while they continue running. The design of simulation experiments can be generalized considerably by recognizing that a computer run at a single design point is not the only possible experimental unit.

In simulation optimization research, the simulation is usually regarded as a black-box or oracle. The output response, *Y(x)*, to input settings, *x*, are typically regarded as a deterministic function, *f(x)*, with added random noise, *e(x)*: *Y(x) = f(x)+ e(x)*. The work required to conduct the experiment is measured coarsely by a "sample size": the number of simulated jobs, replications, batches, observations, regenerative cycles, etc. Uncertainty is driven to irrelevance by running each simulation "long enough". The practical problems of initializing and terminating runs, determining what to observe, selecting appropriate response estimators, and implementing the results, are often not considered.

In the next Section, a re-formulation of generic stochastic optimization problems is presented that requires that optimal solutions be both feasible and *achievable*. This formulation is fundamentally different from the usual additive-noise formulation, and is arguably more appropriate and intuitive for simulation optimization. This is followed by an example of a simulation line search algorithm that may be new (or very old) that illustrates some advantages of combining simulation models, experiments, analysis, and implementation. Section 3 presents several examples where these are used to enhance several simulation response optimization algorithms including randomized search, directional search, pattern search, and particle swarm optimization.

2 BACKGROUND

Modeling is using one system to study another. *Model* is a transitive verb so it must have a direct object. The system being studied will be called the *object* system. Models might be physical systems, systems of equations, legal systems, or even religions or philosophies. Object systems could be real systems, hypothetical systems, value systems, or even beliefs or ideals. Humans think by modeling.

In this paper, the models are discrete-event simulation computer programs. The objects are engineered physical systems with continuous design factors. Such systems include production, transportation, communications, and service systems. Experiments are run to optimize the performance of a simulation model for the sole purpose of implementing the solution in the object system. The two are more tightly connected than widely recognized in simulation research or practice.

In textbook simulation projects, the object system provides structure for coding model dynamics, and perhaps some data to model uncertainty. The object system also has technological limits to implementation that are important in designing simulation experiments. These limits come in the form of constraints and tolerances. Constraints limit the possible choices for the design factor settings to their *feasible* values. Tolerances limit the precision with which these intended settings can be realized in the object system to their *achievable* ranges. Discrete-valued design factors typically are constrained, but have point tolerances. Continuous-valued design factors typically have engineering tolerances, even if they have only one optimal setting.

Constraints apply when choosing the design factor settings that are to be simulated. Tolerances apply when determining design factor settings that are *not* to be simulated.

Simulation optimization algorithms balance a trade-off between spending effort searching near the current best solution (called exploitation) and looking for better solutions elsewhere (called exploration). However, effort must also be spent in response estimation (Andradóttir and Prudius 2009). Ideally, one wants better estimates of the performance of the better performing candidates. A technique for ensuring this is given in Section 2.1.4. The theme of this paper is that there are benefits to not treating exploitation, exploration, estimation, and implementation as distinct activities. *Achievability* will be defined more precisely next and then used in designing a simulation optimization algorithm where the knowledge of the object system design and process engineers is critically important.

2.1 Achievable Optimal Solutions

Proofs of global convergence for simulation optimization algorithms require technical assumptions about the smoothness of the response. Smoothness assumptions make sense: an optimal simulation response that occurs in a small region surrounded by poor performance may not be desirable since it might be hard to realize in the object system (Gabriella, Kleijnen, and Meloni, 2012). This practical consideration has been likened to a parachutist trying to land on the highest peak in a mountain range by deciding where to jump (Samuelson 2010). It may be better to aim for a broad high plateau instead of aiming for the highest sharp peak. This intuitive analogy implies a different response function from that commonly assumed for statistical meta-models. Instead of a deterministic response with *added* noise, $Y(s) = f(x)+e(x)$, the response for this analogy is $Y(x) = f(e(x))$. This because the heights of the mountains are fixed. A parachutist who jumps at x will land at an uncertain point, $e(x)$. This intuitive response model is arguably more descriptive of simulations where fixed input parameters, x, may be used to generate random variables, $e(x)$, that are used in a simulation program. (Perhaps more so if, impractically, the input includes all pseudo-random number generator seeds, run initialization and termination parameters, and so forth.)

The response model, $Y(x) = f(e(x))$, tightly binds the simulation model and the object system. Engineers designing the processes for the object system can provide important advice for designing simulation experiments. In particular, design parameters in the object system usually have engineering tolerances. The simulation study goal is to find feasible input values, $x*$, that globally minimize the response. But, this optimum must be achievable in the object system within the tolerance region, $T(x*)$.

More formally: define all continuous controllable decision, environmental, operational, and process parameters collectively as input variables and denote their feasible values by $x \in X$. Values for these input variables may be set in the object system, but their effects can be random. For example, setting the controls in a bioreactor at specific values does not precisely determine the actual temperatures or pressures every cell experiences; or, where you jump out of an airplane does not determine where you land.

Say the input settings that give us our current best simulation performance are x^c where $f_{min} = f(e(x^c))$. Another setting, x, is not a viable candidate for implementation in the object system if it can only outperform the current best ($f(e(x)) < f_{min}$) by the response improving an unrealistic amount within its tolerances. A *global achievable optimum* is any solution such that, for any better solution to exist at x, the response must improve faster than a limiting rate, $L(x)$: $x \in T(x)$. $L(x)$ is a majorizing function on the amount a response can improve in the object system while inputs are within their tolerances. For appropriate values of the function, $L(x)$, it is necessary again to turn to object system engineers.

Definition: an optimum, x^* for $f(x)$ is <u>achievable</u> if $max f(x) - \min f(x) < L(x): x \in T(x^*)$

There is no reason to look for a better simulation model response in regions where inputs must be controlled tighter than their tolerances in the object system. The technical knowledge of object system engineers about process constraints, tolerances, and performance capabilities is useful in designing effective simulation optimization algorithms. A simple example is presented next.

2.2 A Deterministic Line Search

As an example of a searching for an *achievable* optimal solution, a simple deterministic ($e(x)=x$) line search algorithm is introduced in this section. Achievability will be used to avoid searching futile regions. Consider a deterministic search over the closed interval $[a<x<b]$, where the values of x can only be controlled within symmetric tolerance intervals with half-width, $T(x)$. The rate of change of the response function is bounded by L. Here the change-bounding function $L(x)$ is just a Lipschitz constant. (A symmetric input tolerance and fixed change rate bound are used for clarity. These can non-trivially be extended to asymmetric, stochastic input tolerances and rate bounding functions. With stochastic tolerances, it may become impossible to find any solution that is optimal, feasible, and achievable as the problem size increases.)

Assume the response function $f(x)$ has been evaluated at $k+1$ points, $x_0 = a, x_1, x_2, \ldots, x_k = b$ and we want to decide where to look next. A search can be based on the observation that for a response function to reach a new minimum within any interval $i = (x_i, x_{i+1})$ it must go from its value at one boundary, $f(x_i)$, below the current minimum, f_{min}, and back up to its value at the other interval boundary, $f(x_{i+1})$. Assuming the mean value theorem applies, then a new minimum for f can only be reached in interval i if the absolute rate of change in the function exceeds

$$g_i = \frac{f(x_i)+f(x_{i+1})-2f_{min}}{x_{i+1}-x_i}.$$

The point, x_i^*, in Figure 1 is where this required maximum rate of response change is smallest.

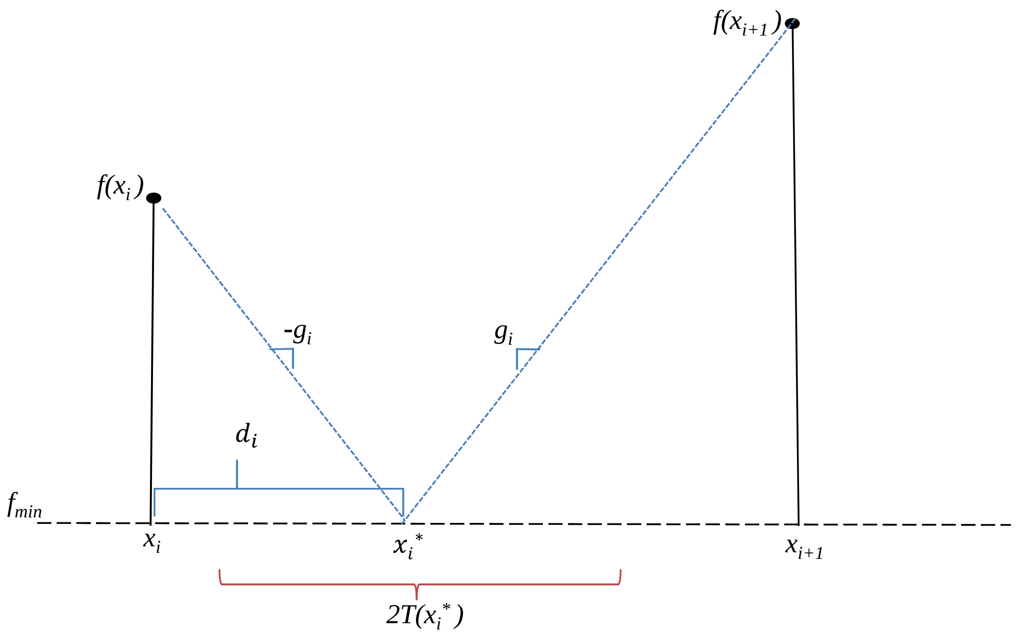
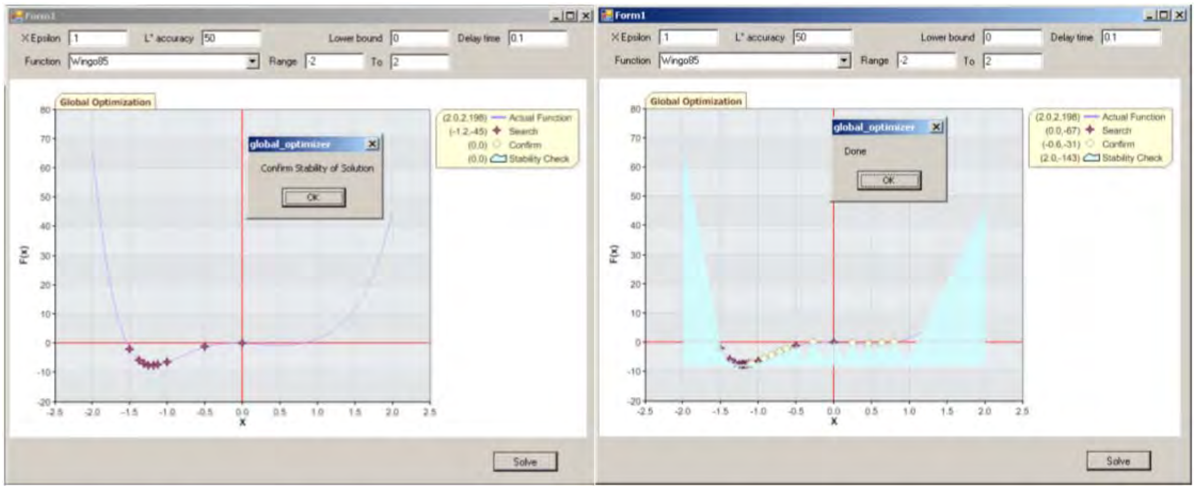


Figure 1: The point, x_i^*, is where a new minimum is most easily achieved in interval i.

An interval where the rate of change in the response that is required to reach a new minimum is smallest is $i^o = argmin\{g_i\}$, so the next point to be evaluated is at $x_{i^o}{}^* = x_{i^o} + f(x_{i^o}))/g(x_{i^o})$.

The distance from this point to its nearest tolerance boundary is $d_{i^o} = min\{x_{i^o}^* - x_{i^o}, x_{i^o+1} - x_{i^o}^*\}$. The search produces non-decreasing values of g_{i^o} and non-increasing values of d_{i^o} and stops when (if) $d_{i^o} < T(x_{i^o}^*)$ or $g_{i^o} > L$ since there are no remaining achievable better solutions. (In stochastic settings, if can be reasonably assumed that the response behaves like a Brownian Bridge between observed function values (Sun, et al. 2011), then a most likely place to find a new minimum next is at $x_{i^o}^*$.)

This algorithm was applied to some classic test functions. Some of the search results are shown in Figures 2 and 3, where the response cannot enter the shaded regions before the search stops. In all the deterministic test functions, the algorithm quickly found and halted at the global achievable minimum. Some of these test functions are famously difficult. Not surprisingly, the algorithm converged most slowly for the very simple problem of minimizing a quadratic response over the positive unit hypercube. Here the unique global solution is at zero. Like secant root-finding algorithms that are based on similar intuition, the line search slowly crawls toward zero until it is closer than its achievable tolerances.



(a)

X Epsilon 1 L* accuracy 50 Lower bound 0 Delay time 0.1
Function Wingo85 Range -2 To 2

(2.0,2,198) Actual Function
(-1.2,-45) Search
(0,0) Confirm
(0,0) Stability Check

global_optimizer
Confirm Stability of Solution

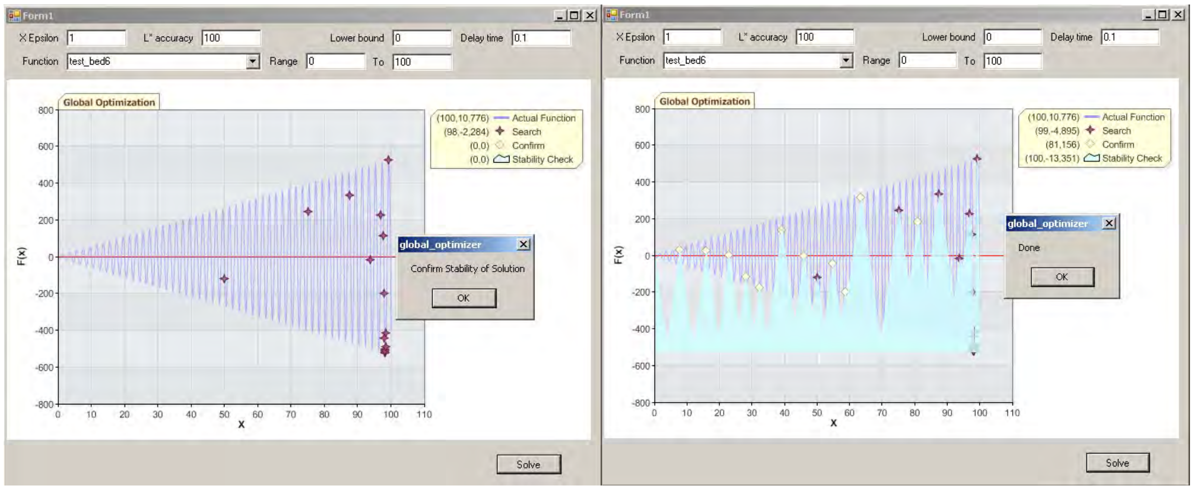
(b)

X Epsilon 1 L* accuracy 50 Lower bound 0 Delay time 0.1
Function Wingo85 Range -2 To 2

(2.0,2,198) Actual Function
(0.0,-67) Search
(-0.6,-31) Confirm
(2.0,-143) Stability Check

global_optimizer
Done

Figure 2: Local Search (a) followed by confirmation/improvement (b) for the Wingo85 function



(a)

X Epsilon 1 L* accuracy 100 Lower bound 0 Delay time 0.1
Function test_bed6 Range 0 To 100

(100,10,776) Actual Function
(98,-2,284) Search
(0,0) Confirm
(0,0) Stability Check

global_optimizer
Confirm Stability of Solution

(b)

X Epsilon 1 L* accuracy 100 Lower bound 0 Delay time 0.1
Function test_bed6 Range 0 To 100

(100,10,776) Actual Function
(99,-4,895) Search
(81,156) Confirm
(100,-13,351) Stability Check

global_optimizer
Done

Figure 3: Local search (a) followed by confirmation/improvement for the test_06 function

Many Search algorithms start with a coarse global search followed by a refined local search. This algorithm tends to behave in the opposite manner. It starts with local searches of promising points until the tolerance constraints are reached. This is followed by globally ruling out regions that cannot possibly contain a better achievable minimum.

While the author has not found this line search algorithm elsewhere, the motivation is the same as for secant root-finding algorithms (take the shortest path to possibly better solutions). Viewed this way, it is a very old idea; the secant root-finding method is reportedly thousands of years old (Papakonstantinou 2007). Like hybrid secant-bisection root-finding algorithms, this algorithm has been implemented with two search modes: a local bisection search of the most promising interval that is still larger than its tolerances, and a global confirmation screening to rule out intervals where a new minimum is unachievable.

This algorithm was refined and first implemented by Lenrick Johnston who added the bisection search. Johnston produced Figures 2 and 3. Links to download his implementation and more results are at the bottom of (http://www.ieor.berkeley.edu/~schruben/). This algorithm likely can be improved by enhancements like Brent's method for secant root-finding.

Lenrick Johnston and the author have extended this algorithm to higher dimensions. The intuition is that to achieve a new optimum a response function must travel from every corner point in every tessellation below the current best minimum and back to every other corner point in its tessellations. The new point that can be reached with the least absolute rate of change in response value is the next point evaluated. Tessellation facets that are smaller than tolerance regions are not searched. This performed well on some classic higher dimension test functions, but has not yet been studied further. Naïve tessellation achievability is not expected to be sufficient to assert global convergence. The focus of this paper is not on this search algorithm; it provides a context for the formulation for stochastic optimization problems and to motivate the notion of an *achievable* global optimal solution.

3 OPTIMIZING SIMULATIONS

Various optimization algorithms were enhanced to exploit special properties of simulations in the author's fall, 2013, simulation class at Berkeley. Some of these were adapted from the literature and some were the students' own inventions. The results included were chosen to illustrate how various optimization experiments can be embedded inside simulation models. None of the algorithms require stopping simulating one candidate system before starting to simulate others. The algorithms include pattern searches, adaptive random searches, line searches, and the agent-based method of particle swarm optimization. Embedding optimization algorithms in simulations was technically straightforward using a variety of tools including Matlab, Sigma, and a new open-source Java code with the working name Tao. (The author would be delighted to discuss the Tao project with anyone interested.)

Two object systems from the simopt.org simulation optimization test-bed were studied (Pasupathy and Henderson 2011). Section 3.1 solves some 1-dimensional GG1 queue optimization problems, and Section 3.2 solves some n-dimensional Queueing System Design problems. These are good test problems for two reasons: the exact steady-state optimal solutions are easy to find analytically; but ironically, these systems are difficult (some versions impossible) to simulate exactly ((Asmussen, Glynn, and Thorisson 1992) and (Sigman 2012)). Simulation initialization, termination, and estimation decisions greatly influence the outcomes like they can in real simulation projects (Schruben 1976).

3.1 Problem 1: The GI/G/1 Queue Problem on Simopt.Org – Line Searching

The object system here is a hypothetical single-server queue with random interarrival times, *A*, and service times, $B(\theta)$, that depend only on the mean-service time parameter $\theta \in [\theta_{min}, \theta_{max}] \subset R$ (L'Ecuyer and Glynn 1994). The objective is to find the optimal service level, θ^*, that minimizes the sum of the average sojourn time plus a cost for providing faster service. The recommended test configuration on simopt.org is an M/M/1 queue with a quadratic cost of service θ^2 and an arrival rate of $\lambda = 1$. The exact steady-state solution to this problem is easy to obtain analytically as

$$C(\mu) = \frac{\lambda}{\theta-\lambda} + \theta^2 \tag{1}$$

3.1.1 Classic Line Search

The worst deterministic performance for the line search algorithm in Section 2.2 was for a quadratic response with the solution at a boundary. A stochastic counterpart to this situation is the transient form of Problem 1 suggested on simopt.org (warm-up for 20 jobs and collect waiting times for the next 50) where the solution is generally unknown. However, over an *unstable* feasible region of $\theta \in [1, 2]$ the unique global solution is at the boundary $\theta = 1$. The algorithm in Section 2.2 and two classic line searches, Golden Section and Binary Search, were applied to this transient version of Problem 1 in this unstable region by Li (2013). His results are presented in Figure 4 where the algorithm in section 2.2 performed better, in this one case, than the other two. Observe that in this unstable region, the true unknown transient response function will be steeper (easier to optimize) than a quadratic response. However, because of the very short warm-up and run durations, the simulated transient response function will be much flatter (harder to optimize) than the true transient response, and more nearly quadratic.

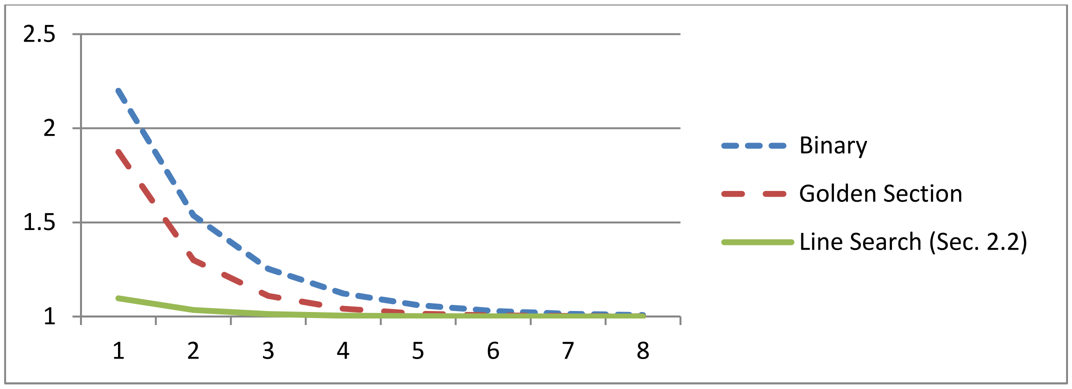


Figure 4: Objective value for Problem 1 using 8 interior simulations (Li 2013)

3.1.2 Particle Swarm Optimization

The input parameter values, θ's, can be simultaneously changed for many concurrently executing simulations. These simulations' different values of θ are analogous to the locations of simulation "particle" agents. Then the collection of simulations is analogous to a swarm of particles. This suggests using agent-based Particle Swarm Optimization (PSO) methods. There are many versions of PSO (the general idea is that different agents will try to move toward the agent that is performing best). This typically involves random perturbations in the speed and direction of the particle moves to avoid being stuck. Various PSO algorithms can be implemented to optimize discrete-event simulations as the following simple example illustrates.

Consider estimating the steady-state response for Problem 1 with an arrival rate of λ =2 and the stable service rate interval of (2, 4]. Here the unique global optimum for the steady-state response function (1) occurs at $\theta^* = 2.45$. A swarm of 40 concurrently executing simulations (particles) starts with their parameter values (locations) uniformly dispersed in the feasible region. These agents will "move" (change their values of θ) toward better performing agents, usually the one with parameters closest to $\theta^*=2.45$. The move rule used here has a similar motivation as the line search algorithm in Section 2.2. After the i^{th} batch of jobs is processed by agent, j, it's parameter setting is changed by

$$\theta_{i+1,j} = \theta_{i,j} + (f(\theta^*) - f(\theta_{i,j})\frac{\theta^* - \theta_{i,j}}{f(\theta*)}$$

(White noise with zero mean and standard deviation 1/i is used to perturb the particle moves. The emphasis here is not on this crude particle move rule, but on the general idea of applying PSO concepts to solve discrete-event simulation optimization problems. Some examples of particle swarm paths are shown in Figure 5, where it can be seen how the values of θ for the 40 simulations "swarm" toward the optimum.

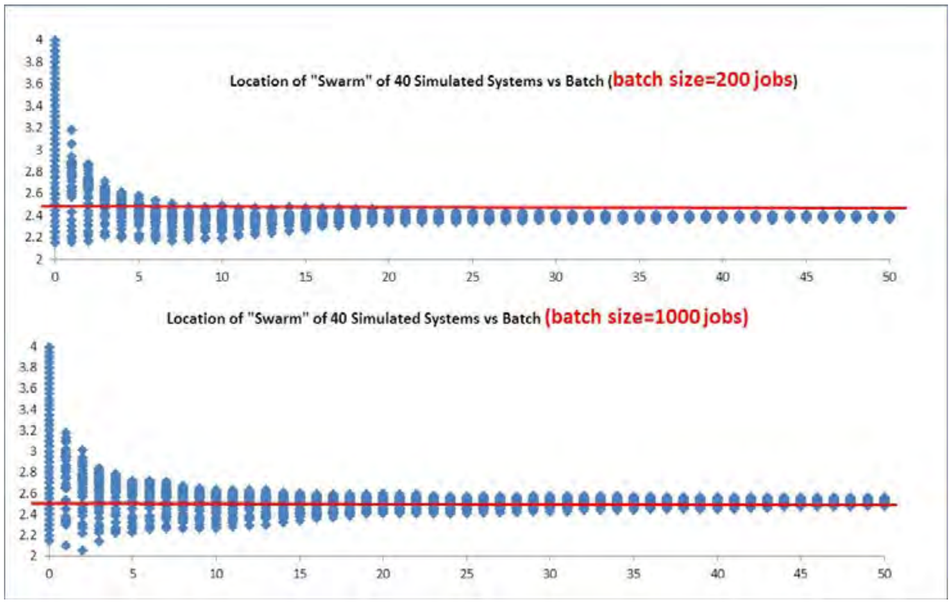
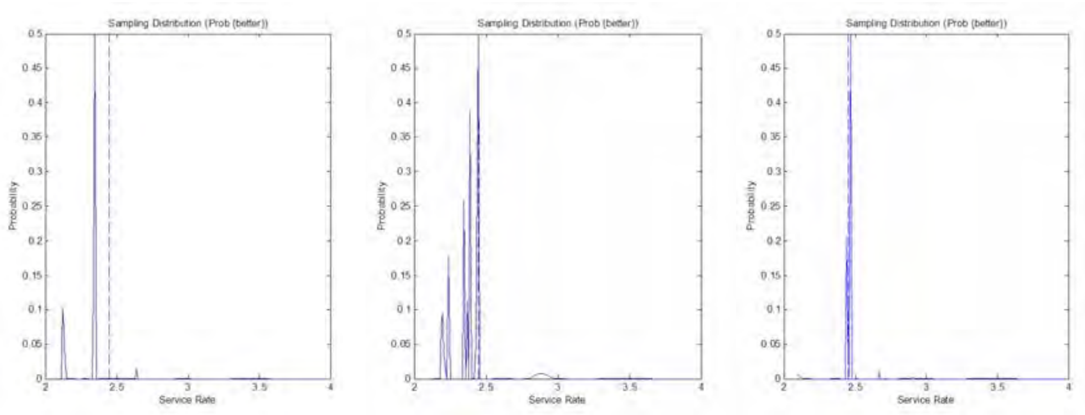


Figure 5: Paths toward optimality by a simulated “swarm” of 40 systems.

Figure 5 shows 50 changes in the values of θ for a swarm of 40 concurrent simulation particles. Two batch sizes of 200 jobs/move and 1000 jobs/move are shown. The batch size (jobs/move) here was of little consequence. No initial warmup period was used. However, the initialization bias naturally dissipated since all 40 simulation particles kept running even while they were changing their parameter values. This experiment evolved into evaluating the best response 40 times, without needing to know which was best!

3.1.3 Gaussian Processes-Based Search (GPS)

The Gaussian Process-Based Search method of (Sun et al. 2011) was implemented by (Lu 2013) for the same steady-state problem as in the previous section (an MM1 queue with an arrival rate of λ =2 and feasible, stable service rates in (2, 4] where the optimal solution is $\theta^* = 2.45$). The GPS stopping uncertainty grid for θ was set to .01 with a termination gap of .02. There was a warmup of 50 jobs at each sampled point. The sampling distribution for some typical simulations of different durations are shown in Figure 6 (a), (b), and (c).



p

Figure 6: Number of jobs in each simulation (a) 500 jobs (b) 1000 jobs (c) 10000 jobs (Lu 2013)

Figure 6 shows how the performance of this algorithm for optimizing simulations might depend on the simulations' durations. This one-run-at-a-time GPS algorithm might benefit by being embedded into a single simulation like the PSO example in Section 3.1.2, with iterations executed concurrently. For a rough comparison, the best performing GPS experiment (6.c) required simulating about 8 times more jobs than the top swarm in the previous section. (The top plot in Figure 5 required 10k total jobs, and the right-most plot in Figure 6 over 80k.)

3.1.4 Time Dilation - the estimation problem

It is not necessary for the same amount of effort to be spent on every concurrently running setting in an experimental simulation "swarm". The better performing simulation particles can effectively be run more than the poorer performing ones using the technique of time dilation (Schruben 1997). The general idea of time dilation is that simulations using shorter time scales are effectively run longer. Time scales can be dilated while simulations are running. The simulated time scales for a swarm of simulations from Sections 3.1.2 is shown in Figure 7. Here the time scales were simply increased proportionally to each simulation particle's current relative cost. A more efficient time dilation algorithm was developed by Hyden (2003). The time scales can be increased (equivalently, runs shortened) for poorly performing simulations until eventually almost all remaining computation is devoted to simulating optimal settings; again without needing prior knowledge of which are the best candidates. Time dilation can address an important simulation experimental design problem: wanting better performance estimates for better performing systems. In this sense, the experiment is again being "designed" while it running. Time dilation can be used in various optimization algorithms. It should be particularly useful in PSO and GPS algorithms.

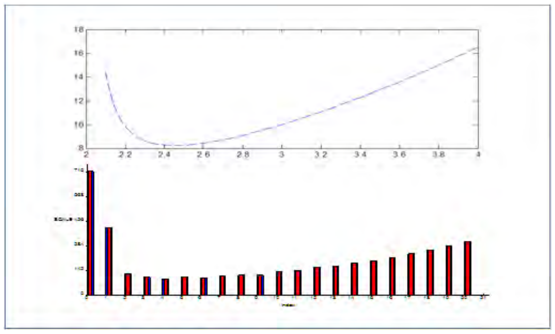


Figure 7: Dilated time scales for a single model (bottom) and true response function (top)

3.2 Problem 2: The Queueing System Design Problem on Simopt.org – Higher Dimensions

The Queueing System Design problem in the simopt.org test bed is based on a model form Barton and Meckesheimer (2006). The object system consists of n communications networks. Figure 8 from simopt.org shows this system for $n = 3$ networks.

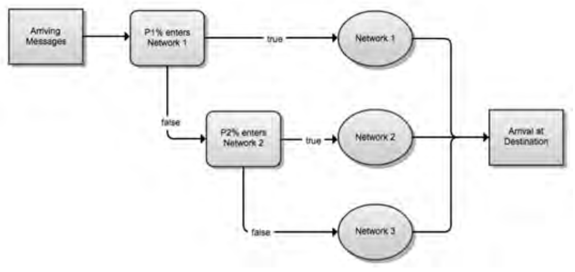


Figure 8: message routing through n = 3 communications channels

The problem is to determine optimal state-independent percentages for routing randomly arriving messages to different communications networks. Messages arrive independently according to a Poisson(1) process and are sequentially routed to networks. Network 1 is considered first and selected with a probability of $p_1 = P_1/100$. If network i is not selected, then network $i+1$ is considered next. The last network accepts all messages that reach it. Each network processes messages one at a time taking random times S_i that have symmetric triangular distributions with a mode i and a range of 1 time unit. All message buffer capacities are unlimited.

The cost of using network i, is $c_i = 1/i$, and there a unit cost of c=.005 for the sojourn time that each message spends in the system. Decomposing the Poisson arrival process according to the conditional probabilities of each network being selected makes this system equivalent to a set of n independent M/G/1 queues, which is easily solvable in steady state. However, this system is not trivial to simulate in steady state. Simopt.org recommends testing a transient version of this problem by processing N=1000 messages starting with an empty system. This makes the simulation response estimation problem trivial, but then the optimal solution is unknown.

3.2.1 Response Surface Methodology for Simulation

Barton and Meckesheimer used several methods including conventional (one run at a time) Response Surface Methodology (RSM) to solve this problem. Response Surface Methodology (RSM) is one of the earliest statistical methods for optimizing stochastic systems. Local regression models are fit to the response surface to estimate where to look next. RSM is usually done in two phases, a sequence of exploratory line searches followed by a more intensive search around a suspected local optimum. In phase I (RSM-I) low order local regression models, usually planes, are used to estimate promising directions for search. This is followed by a line search in that direction until no estimated improvement is seen. RSM-I is repeated until the estimated gradient magnitude is small, indicating a region around a local optimum. Then phase II (RSM-II) is executed where a higher order local regression model is fit. The optimum is estimated for this fitted model, followed by confirmation runs. RSM has been adopted for simulation optimization largely unchanged.

RSM can be improved for simulation since it is possible to simulate multiple design points at the same time. This enables many new opportunities. For example, the full response surface regression model itself ***can be the simulation output*** (Schruben 2010). For example, in Problem 2 the responses are the costs, $Y_{i,j}$, for jobs i=1,2,…N for system factor setting j=1,2,…k. The response model estimate for regression parameter p, $\beta^{(p)}$, is the linear weighting, say with weights $\alpha^{(p)}$, of the average responses at the k design points $(\bar{Y}_1, \bar{Y}_2, ..., \bar{Y}_k)$. The outputs of regression model parameter estimators are simply running averages of the time series $\{\hat{\beta}_i^{(p)}\}$, defined as $\hat{\beta}_i^{(p)} = \sum_{j=1}^{k} \alpha_j^{(p)} Y_{i,j}$.

Since the gradient direction and magnitude are continually being updated, a line search can begin before the estimated gradient search direction is finalized. The differences in conventional RSM applied to simulation and Simulation RSM (SRSM) are illustrated in Figure 9. (Design points labeled “occasionally running” may still be running using time dilation (TD), or may have completed their N jobs.)

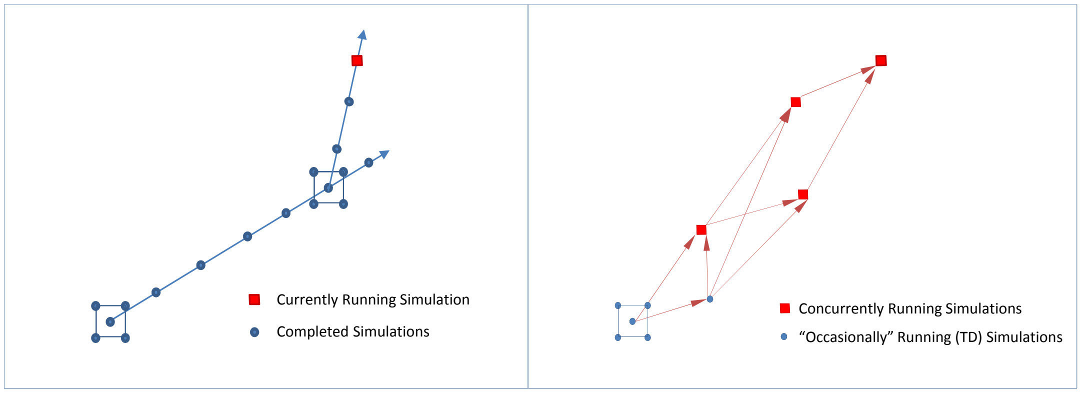

Currently Running Simulation
Completed Simulations
Concurrently Running Simulations
“Occasionally” Running (TD) Simulations

Figure 9: Conventional (one-run-at-a-time) RSM-I search (right) and Simulation RSM (SRSM)

As illustrated in Figure 9, the concurrently running experimental design used in Simulation RSM (SRSM) changes over time by starting or completing runs at factor settings while other experiment points are still running. Xu and Lu (2013) implemented SRMS and reported very good results for problem 2. For n=3, an optimum of (52 63) was found by Barton and Meckesheimer with the log of the optimal cost equal to 3.484. The Xu and Lu SRSM stopped at a solution of (51 61) with a log response of 3.482.

Most notably, with SRSM, Xu and Lu had no need to use a higher order RSM-II model or design. Comparing the search paths for RSM with SRSM in Figure 8 shows why. RSM can be like walking blindfolded along a straight and level path on the edge of a steep mountain slope. By zigzagging along the path like SRSM, one quickly learns they are on the edge of a slope (and can zig more than zag). This may be a good idea for extending RSM beyond simulations, but is not recommended for climbing real mountains.

3.2.2 Pattern Search

A pattern search with concurrently simulated design points was implemented by Adduri and Mehraein (2013) using Tao. The method they used is similar to a Nelder-Mead simplex search. Like RSM, the Nelder-Mead method has been implemented in simulations with little change (Barton and Ivey 1996). However, Adduri and Mehraein included many different design points in their simulation, running concurrently on multiple CPUs. Using Doehlert designs to measure the responses at each of the vertices with concurrent simulations, they periodically pivoted simplex with the worst performing vertex through the weighted centroid formed by the remaining points. A new experiment was included in the simulation at this new point. They reported solving steady-state versions of Problem 2 with 10 networks to very close the theoretically optimal solution found with CPLEX in about 11 seconds on a laptop with 4 CPUs.

3.2.3 Adaptive Random Search

An adaptive random search algorithm was created by (Valadez and Núñez 2013) where experimental points were included in the simulation that were normally distributed centered around the current best setting, P^i_{min}, with decreasing variances. Their algorithm with m experimental design points in the simulation is as follows:

```
Parameters: Set $\sigma = 50,\ p = 0.05$ and $x = 0.7$.
Initialize: Execute a simulation that includes $m$ design points in the feasible region at $X^i_1 \sim U(0,100)$
While $\alpha_n \le .05$:
        Iterate on n: $X^i_n \sim N\left(P^i_{min}, \sigma\alpha_n\right)$, (truncate coordinates at the boundaries) with
        $\alpha_n = x - (n-1)p$
```

The simulation model included all *m* points selected from Normal distributions with shrinking variances.

Valadez and Núñez solved the transient problem (1000 messages) for 3 routers to within 98% of the solution found by Barton and Meckesheimer. They also found near optimal steady state solutions for 10 router problems running a single Sigma simulation model embedded with *m*=1000 moving design points whose settings were selected by their algorithm.

4 SUMMARY

A fundamental rethinking of simulation experiments comes by recognizing that it is not necessary to stop simulating with one input setting to start simulating others. Anything can be observed, and anything changed, at any time, in a simulation. Experimental designs can be thought of, and coded, as part of a simulation, instead of thinking of running a simulation as being part of an experiment. This paper demonstrated how to dynamically embed various response optimization algorithms in a single simulation. Combinations of these techniques are encouraged such as time dilation of particle swarms where new "particles" are added to the simulation using GPS.

The broader intent of this paper is to encourage practitioners to motivate software vendors to develop tools that will enable them to fully utilize the enormous analytical powers of simulation. The ability to observe concurrently simulating systems, and use this information to change them, enables alternatives to run-based experimental designs. Sir R. A. Fisher, credited with inventing statistical experimental design, has been quoted: "The best time to design an experiment is *after* you've done it." With today's simulation technology, Fisher might now agree that the best time to design a simulation experiment is *while* you're doing it.

5 ACKNOWLEDGEMENTS

The author thanks the students in his spring 2013 graduate course for exploring simulation optimization with an open mind and creating some of the examples. He thanks Dr. Lenrick Johnston, CEO of The Bioproduction Group, Inc. (bio-g.com) for enriching the line-search algorithm and screen shots in Sec. 2.2, and Dr. Giulia Pedrielli for her numerous insights (and risotto). The author was partially supported by the National Science Foundation under grant 1153694, CMMI (OR) to the University of California, Berkeley.

REFERENCES

Adduri, P., and Mehraein S. (2013), Homework for Introduction to Data Modeling**,** Statistics and Systems Simulation **(**IEOR 231), University of California, Berkeley, CA.

Andradóttir, S., and A. A. Prudius (2009), Balanced Explorative and Exploitative Search with Estimation for Simulation Optimization, *Informs Journal on Computing*, 21.2 pp. 193-208.

Asmussen, S., Glynn, P., and Thorisson, H. (1992). Stationary detection in the initial transient problem. *ACM Trans. Model. Comput. Simul.*, 2, 130-157.

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol (2010), Discrete-Event System Simulation. 5th ed. Prentice-Hall.

Barton, R. R., and Ivey, J. Jr. (1996), Nelder-Mead Simplex Modifications for Simulation Optimization, *Management Science*, 42.7, pp. 954-973.

Barton, R. R., and M. Meckesheimer (2006), Meta-model based simulation optimization. In Handbook of Simulation, ed. S. G. Henderson and B. L. Nelson, 535--574. Elsevier.

Gabriella Dellino, Jack P. C. Kleijnen, Carlo Meloni (2012), Robust Optimization in Simulation: Taguchi and Krige Combined. *INFORMS Journal on Computing* 24(3): 471-484 (2012)

Hyden, P., (2003), "Time dilation : decreasing time to decision with discrete-event simulation", PhD Thesis, https://catalog.library.cornell.edu/cgi-bin/Pwebrecon.cgi?BBID=4834253&DB=local

Law, A. M. (2007), Simulation modeling & analysis. 4th ed. New York: McGraw-Hill, Inc.

L'Ecuyer, P., and P. W. Glynn (1994), Stochastic Optimization by Simulation: Convergence Proofs for the GI/G/1 Queue in Steady-State", *Management Science*, 40, 12, pp. 1562-1578.

Li, Ruoyang (2013), Homework for Introduction to Data Modeling**,** Statistics and Systems Simulation **(**IEOR 231), University of California, Berkeley, CA.

Lu, Mengshi (2013), Homework for Introduction to Data Modeling**,** Statistics and Systems Simulation **(**IEOR 231), University of California, Berkeley, CA.

Papakonstantinou, J. (2007), "The Historical Development of the Secant Method in 1-D" Presentation at the annual meeting of the Mathematical Association of America, Aug. 3, San Jose, CA. (abstract).

Pasupathy, R. and S. G. Henderson (2011), SimOpt: A Testbed of Simulation-Optimization Problems. Invited paper in *Proc. of the Winter Sim. Conf.*, S. Jain, R. R. Creasey, J. Himmelspach, K. P. White and M. Fu, eds. pp-4080-4090.

Samuelson, D. A. (2010), When close is better than optimal, *ORMS Today*, December, 36.6.

Schruben, L. (1978), Reply to Fox, *Management Sci*. 24.8, April p. 862.

Schruben, L. (1997), Simulation Optimization Using Simultaneous Replications and Event Time Dilation. *Proc. of the Winter Sim. Conf.*, pp. 177-180.

Schruben, L. (1992). Sigma: Graphical Simulation Modeling, Scientific Press.

Schruben, L. (2010), Simulation Modeling For Analysis. *ACM Trans. Model. Comput. Simul.* 20, 1 (Jan. 2010), 1-22. (with online technical companion).

Sigman, K. (2012), Using the M/G/1 Queue Under Processor Sharing for Exact Simulation of Queues. *Annals of Operations Research* (To appear)

Sun, L., Hong, L. Jeff, and Hu, Z. (2011), Optimization via Simulation using Gaussian Process-based Search, *Proc. of the Winter Sim. Conf.*, pp. 4139-4150.

Valadez, R. and Núñez, S. J. (2013), Homework for Introduction to Data Modeling**,** Statistics and Systems Simulation **(**IEOR 231), University of California, Berkeley, CA.

Yang Xu, Yang and Cheng Lu, (2013), Homework for Introduction to Data Modeling**,** Statistics and Systems Simulation **(**IEOR 231), University of California, Berkeley, CA.

AUTHOR BIOGRAPHY

LEE SCHRUBEN (LeeS@berkeley.edu) is a Chancellor's Professor in the Industrial Engineering and Operations Research faculty at the University of California, Berkeley. Before moving to Berkeley he was on the Operations Research and Information (*née* Industrial) Engineering faculty at Cornell for 23 years, where he held the Andrew Schultz Professorship of Industrial Engineering. Prior to that, he was on the faculty in the Colleges of Medicine and Engineering at the University of Florida, Gainsville. He has a BS degree from Cornell Engineering, a MS in Statistics from the University of North Carolina, and a PhD from Yale.