Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

AN INTRODUCTION TO SIMULATION OPTIMIZATION

Nanjing Jian
Shane G. Henderson

Operations Research & Information Engineering
Cornell University
Ithaca, NY, 14853, U.S.A

ABSTRACT

In this tutorial we give an introduction to simulation optimization, covering its general form, central issues and common problems, basic methods, and a case study. Our target audience is users with experience in using simulation, but not necessarily experience with optimization. We offer guiding principles, and point to surveys and other tutorials that provide further information.

1 INTRODUCTION

The usual reason for implementing simulation models is that we want insight or guidance on a decision. The word "decision" is somewhat synonymous with the word "optimization." If the decisions can be represented as decision variables within a simulation model, then we have the option of performing some kind of simulation optimization, i.e., choosing the decision variables to try to maximize or minimize some performance measure that is estimated using simulation. For example, in the bike-sharing example that we use as a case study, the decisions are how to position bikes across the stations in a city in advance of the morning rush hour, and we have a simulation model that can simulate the morning rush hour.

In this paper we provide a tutorial on simulation optimization, assuming a reader that is familiar with simulation experimentation, but not necessarily with simulation optimization. Our goal is to provide a sense of the big issues involved in such problems, rather than to talk about specific algorithms. There have been many tutorials on simulation optimization in the Winter Simulation Conference in recent years (Fu 2001, Fu, Glover, and April 2005, Fu, Chen, and Shi 2008, Chau, Fu, Qu, and Ryzhov 2014), many books have chapters on simulation optimization (Banks, Carson, Nelson, and Nicol 2010, Law 2007, Henderson and Nelson 2006), and excellent surveys exist (Fu 2002, Fu 2015, Jalali and Van Nieuwenhuyse 2015). We will point to several of these works as we proceed, rather than attempting to cover everything afresh here. Our attention is focused on issues that are completely separate from the more "hands-on" issues dealt with in, e.g., Hugan (2014) that relate to the project-management issues you are likely to confront along the way. Indeed, the issues we discuss here result entirely from the mathematical and computational difficulties associated with simulation-optimization problems.

Simulation is computationally expensive, and optimization is bound to make things even slower, so why bother with simulation at all? A shortcut, often followed unconsciously, is to replace all random variables with estimates of their means, and then proceed as if the random variables are not random. This can be a useful heuristic, but in general it is a very bad idea, as we can see through the following (extreme) example of the flaw of averages (Savage 2015). The example shows that replacing random quantities with their expected values can yield very different results from leaving them as random variables. Mathematically it provides an extreme example in which $Ef(X) \neq f(EX)$, so replacing the random variable X with its mean and then plugging the mean into simulation logic f is a bad idea.

Is drinking and walking on a divided highway a good idea? In your limited state you wander somewhat randomly. Measuring across the road, if the borderlines of the road are 0 and 1, then we might model your random location across the road, ξ say, as being uniform on $[0,1]$. Then your mean position is right in the middle of the road: $E\xi = 1/2$. Let your vital status be a function f of your instantaneous position y, so that on the centerline you are fine, but off the centerline you are roadkill, i.e.,

$$f(y) = \begin{cases} \text{dead} & \text{if } y \neq \frac{1}{2}, \\ \text{alive} & \text{if } y = \frac{1}{2}. \end{cases} \tag{1}$$

Therefore, with the *average position* $E\xi = 1/2$, you would be home safe and sound with a status of $f(E\xi) =$ alive. However, the *average state* $E(f(\xi))$ of your life is dead, because the probability that you are on the centerline ($\xi = 1/2$) is 0. Thus $f(E\xi) \neq Ef(\xi)$, and the difference can be catastrophic. In general then, if we model a random system using the average of each input parameter, then the modeled system may not give us results that are even vaguely close to what we would see in reality. So why not keep stochasticity in your model? In this case, *simulation optimization* is potentially a good choice of tool.

As indicated by its name, simulation optimization is the intersection of two powerful traditional decision-making techniques, simulation and optimization. From the simulation perspective, it is motivated by the desire to compare the effects of different decision variables on the output of a complex simulation model; from the optimization perspective, one might be interested in accounting for randomness in a deterministic model to better capture the real life system being modeled. Simulation optimization is also known as "optimization via simulation" or "simulation-based optimization." The general formulation of a simulation optimization problem can be written as

$$\min_{x \in \Theta} f(x), \tag{2}$$

consisting of an objective f, decision variables x, and constraints $x \in \Theta$. This depiction of the objective function and constraints is deceptively simple, since both can be quite complex. As with a deterministic optimization problem, the objective and the constraints can be linear or nonlinear, and the decision variables can be continuous or discrete. However, unlike deterministic optimization, at least one of the objective function and/or constraint functions involves randomness and cannot be evaluated exactly. A standard formulation involves expected values, where f and Θ may be written as

$$\begin{aligned} f(x) &= Ef(x,\xi), \text{ and} \\ \Theta &= \{x : Eg(x,\xi) \geq 0\}, \end{aligned}$$

where ξ symbolizes the randomness in the system, x is the set of decision variables, $f(x,\xi)$ represents the output of the simulation logic for the objective for one replication, and similarly $g(x,\xi)$ represents the simulation output for the constraints for one replication, so $g(x,\xi)$ can be a vector. In this case, the expected value in the objective can be estimated by Monte Carlo simulation using $f_n(x) = \frac{1}{n}\sum_{i=1}^{n} f(x,\xi_i)$, the average over n realizations $\xi_i, i = 1,\cdots,n$ of ξ, with each $f(x,\xi_i)$ being the output of one replication of the simulation model. (And similarly for the constraints, if present.) Another formulation arises when

$$\begin{aligned} f(x) &= \inf\{y : P(f(x,\xi) \leq y) \geq \alpha\} \\ \Theta &= \{x : P(g(x,\xi) \geq 0) \geq \beta\}, \end{aligned}$$

where the objective is the $\alpha-$quantile of $f(x,\xi)$ for some $\alpha \in (0,1)$, and we use so-called chance constraints. For example, in portfolio optimization the objective in the first formulation is to maximize the expected return (equivalently, minimize the negative of the expected return) while the objective in the second formulation is to minimize the value-at-risk.

Some examples of classical simulation-optimization applications include the following. For more detail on these and many other problems, there is a library of simulation-optimization problems (Pasupathy and Henderson 2012).

- Newsvendor: How many newspapers, x, should be purchased at the start of each day, so that the long-run daily profit, $f(x)$, is maximized with random demands ξ?
- Supply chain inventory: What base stock level x minimizes a combination $f(x)$ of inventory holding costs and the loss from unmet demand in the presence of random demand ξ?
- Queuing systems like call centers: With customers arriving randomly (ξ), how many servers x should we staff in order to minimize staffing costs $f(x)$ while ensuring that expected waiting times $g(x) = Eg(x, \xi)$ are not excessive?
- Financial Optimization: What portfolio of stocks x should we invest in, so that the expected return $f(x)$ is maximized subject to a bound on the potential losses $g(x) = Eg(x, \xi)$?
- Healthcare: Where should we put the ambulance bases x in a city so that the expected response time $f(x)$ to calls (ξ) is minimized?

Simulation optimization problems are not easy to solve. Beyond the complexity in solving deterministic optimization problems, which is considerable, we have the added problem that we can never compute $f(x)$ exactly; we can only obtain *estimates* of $f(x)$. This creates a host of problems in trying to come up with algorithms that reliably identify high-quality solutions. The many complex issues that arise are discussed in §2. §3 describes some tools and principles one can use. §4 is a case study on the bike sharing program in New York City, Citibike, where we consider some aspects of a real-world problem in light of the ideas outlined earlier in the paper.

2 COMMON ISSUES AND REMEDIES

Understanding a few issues from the outset can save you a great deal of pain in your work.

2.1 Local Versus Global Solutions

We all wish we could obtain the exact value of $f(x)$ at any $x \in \Theta$, without any simulation error at all. Even if we had such a perfect simulation, so that the "simulation" part in the "simulation optimization" phrase is redundant, the optimization problem is almost always *still very hard*. This happens because the function f may be very complicated. Suppose the decision variables x take on continuous values. The true function can look like a rolling landscape, with many valleys and basins. Unfortunately, no simulation optimization algorithm can see that whole landscape. Imagine that you are standing on this landscape at the point x in dense fog. You have no idea what the rest of the function looks like. That is the problem faced by an optimization algorithm. If you are fortunate, then you may have access to derivative information, which means that you can feel the slope of the hill underneath your feet, but you are still in dense fog. This analogy helps to understand the magnitude of the task an optimization algorithm faces.

Most simulation optimization algorithms keep one or more solutions in mind, and try out nearby solutions, looking for reductions in f relative to the current solutions. In doing so, they may follow a promising valley to a solution from which no improvement is possible. Such a solution is called a *local* minimum. The local minimum with the smallest objective value is called the global minimum. There is no guarantee that a local optimum identified by our search algorithm is a global optimum. The problem is that in the fog, there is no way to know whether the basin in which you are standing is the lowest of all basins. Indeed, if you are standing in dense fog at the bottom of the Grand Canyon thinking you have found the lowest point in the USA, how would you know about Death Valley?

Accordingly, unless the function f has special structure, such as some form of convexity, *the best an algorithm can promise is to locate a local minimum.* This is often good enough, but if you would like to have a chance of finding better local minima, a standard approach is to restart the search algorithm from some new solution, often chosen randomly. Algorithms with random restarts can be quite effective if your landscape is not too rolling. Also, they can be proved to eventually visit all minima (under conditions) so in theory guarantee you'll find a global minimum, but *in practice* the time required for this guarantee to take effect is usually enormous.

2.2 Many Decision Variables Means Huge Search Space

This issue is sometimes referred to as the Curse of Dimensionality. The idea is easiest to understand with discrete decision variables, but applies to problems with continuous variables as well. Suppose your decision variables are the number of agents that will work particular shifts in a call center. Let d be the number of shifts, and let x_i be the number of agents that will work Shift i. If your shift schedules repeat on a daily basis and start on the hour, then $d = 24$, but typically d is far larger because shifts may have different scheduled times for breaks etc. Suppose also that the maximum number of agents that can take a specific shift is n, so the possible values for x_i are $0, 1, 2, \dots, n$. Then there are $(n+1)^d$ possible values for the vector of decision variables $x = (x_1, x_2, \dots, x_d)$. For example, if $d = 24$ and $n = 10$, then there are 11^{24} possible x vectors. With so many options, most algorithms will struggle to find good solutions.

What can one do in the face of such complexity? One option is to exploit your domain knowledge. For example, in the call center problem, you might start the search with the x_is that correspond to your current shift schedule. Another option is to use an algorithm that has some awareness of the structure of the function f. For example, if f gives the average speed of answer as a function of the number of agents working each shift, then we'd expect that f decreases as each component of x increases, and that there are diminishing returns in f in each component of x, at least when each component of x gets large.

In any case, when running algorithms on desktop computers, or even servers, you cannot always expect rapid identification of high quality solutions in problems involving even moderately complex simulation models. *Get used to leaving your machine running overnight (and even* that *might not help)!*

It is not all bad news. One avenue offers much promise, as commented on frequently in the past, e.g., Fu (2002) and as discussed later. The use of parallel computing, perhaps through the use of high-performance computing, or cloud computing, can help greatly. We make this observation with some humility, however, since even very powerful parallel computers are eventually undone by the curse of dimensionality.

Not all algorithms are created equally in terms of running time. One can plot the progress of simulation optimization algorithms as a function of the computational time spent on a selection of example optimization problems as a means to compare algorithms. This is the goal of the test-problem library described in Pasupathy and Henderson (2011). This site is constantly on the lookout for interesting problems to add to the library, so if you have a nice example then do consider contacting the administrators of that site.

See, e.g., Rinnooy Kan and Timmer (1987) and references therein for much more on these issues.

2.3 Continuous versus Discrete Decision Variables

Concepts like continuity and differentiability can sometimes be exploited when all the decision variables in a problem are continuous valued. Problems with continuous objective functions allow one to use efficient local search algorithms that exploit the fact that “close” points will likely have “close” objective function values. Local search is harder when the variables are discrete, like in the call-center example, because concepts like continuity are not natural. Some recent work tries to adapt algorithms for continuous variables to problems with integer-ordered variables (Wang, Pasupathy, and Schmeiser 2013, Luo and Lim 2011).

2.4 Optimizing In The Presence of Noise

So far, all of the issues we have discussed in this section apply to deterministic optimization problems just as much as simulation optimization problems. Unfortunately, in simulation optimization we have the added complexity of dealing with imperfect estimates of $f(x)$ for any given x. The difference between the exact value of $f(x)$ and the estimated solution value $f_n(x)$ that our simulation produces from a run of length n is called the simulation error or noise, and it is unavoidable in stochastic simulation. In the presence of this noise, we can never be absolutely certain that one point, x_1, is better than another, x_2, i.e., we can never be absolutely certain that $f(x_1) < f(x_2)$. We *can* become more and more certain through using longer simulation runlengths at x_1 and x_2, but that takes computational time that might be better spent exploring other x values. Simulation error thus creates issues that are not a problem in the zero-error case.

2.4.1 Simulation Noise Can Swamp the Signal

Suppose that $f_n(x)$ is the estimated value of $f(x)$ based on a simulation run of length n. The "noise" at x is the value $\varepsilon_n(x) = f_n(x) - f(x)$. As the simulation runlength gets longer, i.e., $n \to \infty$, $\varepsilon_n(x) \to 0$. However, for any finite n, $\varepsilon_n(x) \neq 0$. We prefer x_1 to x_2 if $f(x_1) - f(x_2) < 0$. Our best estimate of this difference is $f_n(x_1) - f_n(x_2)$. The error in our estimate of the difference is $f_n(x_1) - f_n(x_2) - (f(x_1) - f(x_2)) = \varepsilon_n(x_1) - \varepsilon_n(x_2)$. If this error is larger in absolute value than $f(x_1) - f(x_2)$ then we can be easily misled about which of x_1 and x_2 is the better point. Hence, if algorithms employ runlengths that are too small, then the algorithms may "chase noise," in that their behaviour is dominated by the noise terms $\varepsilon_n(x)$. Accordingly, good-quality algorithms should take care to choose runlengths to avoid this problem. Often this is left to the user. In such cases, Hong and Nelson (2009), §5.1, suggests that you pick some representative x values from the solution space Θ and choose a runlength that ensures statistically significant differences between their estimated function values. One needs fewer replications to estimate the sign of $f(x_1) - f(x_2)$ than to estimate the value itself, so the needed runlength can be smaller than you might expect.

A way to try to improve the signal to noise ratio is to employ some kind of *common random numbers*. In this method, we drive the simulation at both x_1 and x_2 with the same random sequences. This typically ensures that $\varepsilon_n(x_1)$ and $\varepsilon_n(x_2)$ are positively correlated, so that $\varepsilon_n(x_1) - \varepsilon_n(x_2)$ is reduced relative to the case where we use independent simulations at x_1 and x_2. Hence, the noise is reduced while the signal is retained. To do this, use the same sets of random number seeds to drive the simulations at different solutions. The method of *sample average approximation* that is discussed later in this paper is closely related to this idea, and an important theme in algorithm design (and our bike-sharing case).

2.4.2 Failing to Recognize an Optimal Solution

As noted in Hong, Nelson, and Xu (2015), this problem can easily arise during the search, and it can happen with near-optimal solutions as well. We may fail to select a high-quality solution even if that solution is visited. The remedy for such problems is to try to employ simulation runlengths that yield statistically significant differences in estimated function values. While this is difficult when the user is required to set the runlengths, it is relatively straightforward if one employs so-called ranking and selection methods (see §3) to perform a second pass over visited solutions to determine which one is the best.

2.4.3 Getting Poor Estimates of the Objective of the Estimated Optimal Solution

Hong, Nelson, and Xu (2015) point out that the estimated objective function value $f_n(X)$ of the estimated optimal solution X should be viewed with great suspicion at the conclusion of the optimization run. The issue is that this quantity is biased low, so the estimated objective function value looks better than it actually is. This happens even if $f_n(x)$ is an unbiased estimator of $f(x)$ for each x. To see why this happens, imagine a situation where the true function $f(x)$ is constant, so that *any* x is optimal. In this setting, the noise drives the selection of the optimal point, so we are naturally drawn through the optimization process toward selecting a point X with negative error. This concept appears to have been known informally for a very long time. It was formalized for sample average approximation in Mak, Morton, and Wood (1999), who suggest exploiting this bias problem to give lower bounds on the optimal value of the *true* objective function. Such bounds are hard to get but very useful; they can be compared with the (estimated) objective value of a feasible solution to help determine when to stop a search algorithm, for example.

The remedy for this problem is to take the estimated optimal solution X from one's simulation optimization run and estimate $f(X)$ using an independent sample (new random seed). The new sample will provide an unbiased estimate of $f(X)$ (taking X as fixed, i.e., conditional on the choice of X).

2.4.4 When to Stop?

Deterministic optimization algorithms typically halt when they fail to make progress in the objective for some time. Simulation optimization algorithms can do the same, but if the objective is noisy, then it can be hard to identify progress as opposed to noise that makes us *think* we have made progress. The research literature is very thin on this issue, with most algorithms described as if they will run forever, with the user expected to terminate the search when they can't stand to wait any longer! A practical approach is to increase the sample size used to estimate the function values as the search proceeds, so you can at least feel confident you've estimated the objective reasonably accurately. If you don't have control on the sample size within the algorithm, then you can start with a small sample size, identify a good solution with the small sample size, then sequentially double the sample size and reoptimize starting from the previous estimated best solution. If you have control over when the algorithm stops each time, then a practical rule is to stop when the algorithm fails to make progress for m "steps" in a row, where m is a parameter that you will have to choose, and "steps" can mean different things in different algorithms.

2.5 Model Madness

In the setting of simulation optimization there is a tendency to first build a simulation model, and only towards the end of that process to decide how to perform optimization. Put more bluntly, we tend to try to "bolt on" optimization only after the simulation model is built. This process stands in stark contrast to the recommendations of modeling textbooks, where one is strongly advised to begin with the question at hand and to build a succession of models, each of which answers the question to some degree; see, for example, Starfield, Smith, and Bleloch (1994). The models are usually increasing in complexity as we progressively learn more about which factors are most important to capture to obtain a quality decision. Interestingly, these models need not be simulation models. In fact, it may be better if the early models are not simulation models, owing to the complexity of building and maintaining complex simulation models.

This process of developing a sequence of models, rather than a single model, is in line with top-down design principles in software engineering, and similar ideas have previously been expressed in the simulation literature, e.g., Nelson (1995), Schruben (2010), Henderson (2005). It is also in line with comments by Barry Nelson in Fu et al. (2014), in which he suggested we view simulation as a decision support tool through the lens of mathematical programming, where questions of optimality, feasibility, and sensitivity are central. Asking such questions for each model in a sequence of models allows one to compare the insights from each model before deciding whether to increase model complexity and, if so, how.

3 TOOLS

Numerous simulation optimization methods exist; see, e.g., the taxonomy in Hachicha et al. (2010). We choose to discuss only a few representative methods and concepts.

3.1 Sample-Average Approximation

Recall that we can estimate $f(x) = Ef(x,\xi)$ in (2) by a sample average $f_n(x) = n^{-1}\sum_{i=1}^{n} f(x,\xi_i)$. Here each ξ_i represents the random number inputs for the ith replication. If we *fix* the sample size n *and* the ξ_is from the outset, using the *same* random number inputs for different x values, then we are using sample-average approximation (SAA). For the fixed inputs, we can minimize the sample-average function f_n with a deterministic optimization algorithm that we are familiar with. We can also use a sample average to approximate constraint functions if they are stochastic. SAA can be implemented by carefully setting the random number seeds before each simulation run at any x.

For example, in call-center staffing SAA generates the customer arrival times, service times etc. associated with n different days, and then finds the staffing plan x that minimizes the cost of staffing subject to meeting the service-level requirements over those n days.

Assuming n is large enough, it seems reasonable to expect SAA to find high-quality solutions to the original (not sampled) problem, and that is true under very general conditions. Furthermore, SAA takes advantage of common random numbers (see §2) to sharpen the comparison of the objective at different x values, because we use the same set of random numbers $\{\xi_i, i = 1, 2, \cdots, n\}$ at all points.

SAA is not an algorithm itself, but rather a "recipe" for tackling simulation-optimization problems. One must still choose a deterministic optimization algorithm. Typically, this is done under the assumption that the sample function f_n has some structure, e.g., smoothness or convexity, that can be exploited by a deterministic algorithm, so while SAA is highly appealing for a number of reasons, it also leaves a lot of work to the user. To the best of our knowledge, if you want to use SAA you need to code it directly, because it is not built into any standard packages. For more information, including a more in-depth discussion on when it is useful, and an entry point to its large literature, see Kim, Pasupathy, and Henderson (2015).

3.2 Metamodeling

In *metamodeling*, also known as response surface methodology, one approximates the true function f by some simple-to-compute function $\tilde{f}$, and uses $\tilde{f}$ to direct the search for an optimal point. Such approximations can be *global* in that they attempt to approximate the function f over the entire search space, or *local* in that they approximate f only in the neighborhood of a current iterate. Global metamodels are usually used only for problems with a small number of decision variables because it is hard to match a response surface over large regions. Local metamodels iteratively approximate f by $\tilde{f}$ near some point, and use $\tilde{f}$ to identify and move to an improved point. To learn more about metamodeling, see Barton (2009), Kleijnen (2015).

3.3 Stochastic Approximation and Gradient Estimation

Stochastic approximation (SA) is a category of local-search methods that can be applied to solve simulation optimization problems with a smooth objective function f on a continuous space Θ. Versions exist for problems with constraints, provided those constraints are fairly simple, like bounds on the decisions variables. SA is similar to the steepest-descent method in deterministic nonlinear optimization. It generates a sequence of proposed solutions $(x_n : n \geq 0)$, and relies on a gradient estimate $g(x_n, \xi_n)$ at each iteration n. At iteration n it moves from x_n to x_{n+1}, where $x_{n+1} = \Pi_\Theta(x_n - \alpha_n g(x_n, \xi_n))$. Here $\alpha_n \geq 0$ is a stepsize that is usually only a function of n, and Π_Θ is a projection back onto the solution space Θ in case the step is infeasible. The stepsize sequence $(\alpha_n : n \geq 0)$ is typically chosen so that $\sum_n \alpha_n = \infty$, $\sum_n \alpha_n^2 < \infty$ which ensures convergence of x_n to a locally optimal solution under certain conditions. A standard choice is $\alpha_n = c/n$ for some positive constant c. Each step of the SA method is very fast, but the long run performance depends heavily on the choice of the stepsize α_n. If the stepsize is chosen "just right" then SA is very hard to beat. However, if the stepsize is too small, the convergence of x_n may be too slow, and if the stepsizes are too big, the steps will keep oscillating between the boundaries of Θ without making any progress. Some algorithms adjust the stepsize as they go, but no completely satisfactory method has been found over the 60-year history of the algorithm.

There are several ways to estimate the gradient of f. The simplest (but biased) one is to use finite differences by perturbing the solution x_n in each dimension i. Other gradient estimation techniques yield unbiased estimates of the gradient in special cases. To the best of our knowledge, if you want to use SA you need to code it directly, because it is not built into any standard packages.

No matter which gradient estimation method one uses, SA is only a *local* search technique, so if you want to find globally optimal solutions, you need some additional mechanism such as random restarts. See Chau and Fu (2015) for much more on SA, along with an entry point to its vast literature.

3.4 Ranking and Selection

Ranking and Selection (R&S) methods are usually applied when the search space Θ is finite and "small" (say, less than 500 solutions), so that we can afford to estimate the quality of every solution by at least a few simulation runs. These methods do not perform search, instead exhaustively testing all solutions. Most R&S methods offer a strong guarantee associated with the estimated optimal solution, as we discuss further below. The major approaches of R&S primarily differ in how they allocate a computational budget among systems, and when to stop and declare an optimal solution. There are Bayesian procedures that successively update a prior distribution on the system means (Chick 2006, Branke, Chick, and Schmidt 2007), but here we focus on the more common frequentist viewpoint.

Suppose we have a total of k solutions (usually called alternatives or systems), $x_1, x_2, \ldots, x_k$, and let $\mu_i = f(x_i) = Ef(x_i;\xi)$ be the objective of the ith solution, $i = 1,2,\cdots,k$. R&S algorithms usually assume that one can obtain a sequence of independent and identically distributed replications $(X_{i,j} : j = 1,2,\ldots)$ from System i, although some variants can work with steady-state simulations too. Separate algorithms are usually needed when the replications on different systems are independent, as opposed to when using common random numbers. It is usually assumed that $X_{i,j}$ is normally distributed, which can be approximately ensured through the central limit theorem by obtaining replications in batches.

R&S procedures typically start by obtaining a small sample of size 10 or so on each system, and then use that initial sample to decide how much, if at all, to further simulate each system. Many algorithms perform some kind of screening in which systems are successively eliminated as the sampling proceeds. Hence, the total sample sizes, n_i say for System i, can be different for each i. The (automated) choice of n_i is almost always quite complicated, but some key principles are that n_i tends to be larger for systems with large variance, and that n_i tends to be larger for systems that are close to optimal.

R&S algorithms typically offer one of two guarantees. In explaining these guarantees, we assume that the systems are listed from worst to best ($\mu_1 \geq \mu_2 \geq \cdots \geq \mu_k$), so that μ_k is optimal, but of course the R&S algorithms are unaware of this ordering.

The first type of guarantee is called a "probability of correct selection" (PCS) guarantee. A correct selection happens when the algorithm returns System k. However, if the difference in performance between System k and System $k-1$, $\mu_{k-1} - \mu_k$, is very small then it is very hard to ensure a correct selection. Hence, this kind of guarantee requires an "indifference zone" $\delta > 0$, and the guarantee only holds if $\mu_{k-1} - \mu_k \geq \delta$. The parameter δ is specified by the user, and represents the smallest difference worth detecting. So a probability of correct selection guarantee means that when the best system is at least δ better than the second best, the algorithm will return the best system with at least some desired probability. Unfortunately, when the best system is not at least δ better than the second best, then these algorithms offer no guarantee.

A probability of good selection (PGS) guarantee fixes this problem. A good selection arises when the system that is reported as best is at most δ worse than the very best, i.e., if I is the (random) index of the system reported as best, then the probability of good selection is $P(\mu_I \leq \mu_k + \delta)$, and good-selection procedures are designed to guarantee that this probability is at least some desired probability. This guarantee implies a probability of correct selection as well, so it is the stronger guarantee.

Algorithms that provide either of these guarantees are typically very conservative, in that the actual PCS or PGS is much higher than 0.95, which means that they tend to sample much more than they really need to. This is not ideal, but comes about because the theory is not perfect. There are other methods for ranking and selection with finite solution space that are more efficient, but they cannot *guarantee* PCS or PGS. Many such methods use the "Optimal Computing Budget Allocation" to decide on runlengths, while others that are tailored towards certain loss functions (functions that measure the loss associated with not selecting System k) use the "Expected Value of Perfect Information"; see Chen, Chick, and Lee (2015).

A relatively recent development is the use of large-scale parallel computing to solve R&S problems. Such methods can handle on the order of $k = 10^6$ systems, but are still in development. See Luo and Hong (2011), Luo et al. (2014), Ni et al. (2015).

Although R&S algorithms have been implemented in some commercial simulation software packages, they are not widely available. To learn more about R&S methods, see Kim and Nelson (2006).

3.5 Random Search Methods

A weakness of the methods discussed so far is that they are not necessarily easily tailored to different problems, so there is a lot expected of the users of such methods. Both SAA and SA also suffer from the fact that they work best when the problem enjoys some structural properties that are not easily checked. Random search methods are easier to implement and do not require problem structure, and these have proved to be decisive advantages in their adoption in commercial simulation software. The downside is that they usually come with few or no guarantees, and the guarantees that are provided are usually relevant only for very large simulation runlengths that are often infeasible from a practical perspective. Nevertheless, they have helped many people obtain improved solutions to their problems, so are important.

A random search method typically has the following form. For each iteration $n = 1,2,\ldots,$

1. Sample: Choose a sampling strategy L_n based on the information we have collected so far. Sample a set of solutions $x_n^1, x_n^2, \cdots, x_n^{r(n)}$ in Θ using L_n.
2. Evaluate: Use simulation to estimate $f(x_n^s), s = 1,2,\cdots,r(n)$ by $\hat{f}(x_n^s), s = 1,2,\cdots,r(n)$. From all the information collected to date, calculate an approximation x_n^* to the optimal solution.

The sampling strategy L_n can be deterministic or randomized, and it can depend on all points visited, only the last point visited, or nothing, the latter meaning that L_n is independent of the past. For example, simulated annealing selects a random point x' in a neighborhood of the current point x_{n-1}, estimates the function value at x', and moves there with a certain probability. The probability of moving is 1 when there is an estimated improvement in the objective function ($\hat{f}(x') < \hat{f}(x_{n-1})$), and is less than 1 otherwise. Other examples of random search methods include tabu search, genetic algorithms, stochastic ruler (Yan and Mukai 1992), nested partitions (Shi and Ólafsson 2000), and COMPASS (Hong and Nelson 2006).

The quality of a random search method is determined by the sampling strategy. On one hand, we want to sample in unexplored areas, but on the other hand, sampling more points around a good solution might yield better solutions. This is the famous (but hackneyed) exploration/exploitation tradeoff.

An important improvement to random search is to use R&S after the search concludes to identify a best system from the better solutions that are visited in the search; see Boesel, Nelson, and Kim (2003).

Random search methods are used in most commercial simulation codes. To learn more, see Andradóttir (2015), Hong, Nelson, and Xu (2015), Fu, Glover, and April (2005), Ólafsson (2006).

4 BIKE SHARING

Citibike operates a bike-sharing program in New York city (NYC). The fluid model and simulation optimization work in this section is new, but the other results are based on work by a Cornell research group; see O'Mahony and Shmoys (2015) for an overview.

Citibike uses approximately 380 bike stations and 4000 bikes. Daily usage can exceed 40,000 rides. An important problem is the design of overnight rebalancing schemes, in which Citibike repositions bikes overnight in preparation for the morning rush from 7am to 9am. The problem is important because net daily flows do not "cancel out." The question we consider here is where bikes should be positioned across stations in advance of rush hour to avoid stock outs and full racks.

The decision of where to position bikes is made more complex by user behaviour. In the event of a stockout or a full rack, users go to a nearby bike station, which they can locate using a smart-phone app that supplies real-time information about the number of bikes and open docks available in stations. A bike may be abandoned by a user after a few failed attempts to return it, although this is very rare.

Suppose we decide to place x_i bikes at station i right before the start of the morning rush hour. The constraints account for the dock capacity r_i at each station and the total budget of bikes b. (We take the rack capacities to be fixed, even though they too can be modified.) The optimization problem is then to

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & \sum_i x_i = b \\ & 0 \le x_i \le r_i, x_i \text{ integer } \forall i, \end{aligned} \tag{3}$$

where the *objective function* $f(x)$ models the quality of the allocation x. (Here x is 380 dimensional.) We take $f(x)$ to be the expected number of "annoyed customers" that are unable to check out or return a bike at some station because of empty or full racks during the morning rush.

A lot of effort is needed to simulate this system, let alone to optimize it. Before starting to write a complex discrete-event model it is helpful to proceed through a sequence of ever more complex models to obtain intuition on how the system behaves and the form of a good allocation; see §2 on Model Madness.

Our first "model" is simple: ensure that each station is half full, based on the heuristic that we want both bikes and space at each station. In fact, this was the solution Citibike originally planned to use before Cornell became involved. However, it has some serious shortcomings due to flow imbalances.

4.1 A Fluid Model

Suppose that we ignore the randomness in the arrival and departure processes of bikes from stations, instead assuming that customers drop off bikes at a constant rate λ_i, and depart with bikes at a constant rate μ_i, from each bike station i throughout the period. (These rates can be estimated from data, taking care with the censoring that arises when we run out of bikes or racks.) Then, the number of bikes at Station i is a linear function of time, at least until bikes run out or the capacity is hit. More precisely, the number of bikes at Station i at time s is $Y_i(s) = x_i + s(\lambda_i - \mu_i)$, provided that this number lies between 0 and the capacity r_i. We incur cost if this value hits 0 (when $\lambda_i < \mu_i$) or hits the capacity r_i (when $\lambda_i > \mu_i$).

First suppose $\lambda_i < \mu_i$, so there is a net flow away from the station. We then hit 0 at time $x_i/(\mu_i - \lambda_i)$. From this moment on, bikes still arrive at rate λ_i, so annoyed bikers accumulate at rate $\mu_i - \lambda_i$. Accordingly, the contribution to the objective function from Station i over the morning rush period, denoted $[0,t]$, is

$$\left(t - \frac{x_i}{\mu_i - \lambda_i}\right)^+ (\mu_i - \lambda_i) = ((\mu_i - \lambda_i)t - x_i)^+,$$

where $z^+ = \max\{z, 0\}$. Similarly, if $\lambda_i > \mu_i$, then the contribution is $((\lambda_i - \mu_i)t - (r_i - x_i))^+$.

So if $\mathcal{O}$ is the set of outflow stations where $\lambda_i < \mu_i$, and $\mathcal{I}$ is the set of inflow stations where $\lambda_i > \mu_i$, then our optimization problem can be written as in (3), where

$$f(x) = \sum_{i \in \mathcal{O}} ((\mu_i - \lambda_i)t - x_i)^+ + \sum_{i \in \mathcal{I}} ((\lambda_i - \mu_i)t - (r_i - x_i))^+ \tag{4}$$

This problem yields a linear integer program. Linear programs are convex, so if we ignore the integrality constraints the problem is convex and we avoid the "local versus global" issue. We could go ahead and solve this formulation numerically, but we instead use the problem formulation to gain some insights.

First, for outflow stations, the minimum number of bikes x_i needed to ensure happy customers is $(\mu_i - \lambda_i)t$, which is essentially the flow imbalance over the rush period. If this value is greater than the rack capacity r_i, then we will have some unhappy customers at Station i. Similarly, for inflow stations, if $(\lambda_i - \mu_i)t > r_i - x_i$ then we will have unhappy customers. Second, the flow imbalance at a station may be so large that there is no way to avoid the problem, apart from increasing the capacity r_i at that station. Third, suppose we take an incremental approach to finding an optimal solution, starting with no bikes at any station. We then add bikes to outflow stations, but to which one should we first add a bike? This formulation makes no distinction; adding a bike to any outflow station for which the imbalance $(\mu_i - \lambda_i)t \ge 1$ gives the same improvement in objective function. This observation results from the fact that our objective function views an unhappy customer at a busy station as equally important as an unhappy customer at a quiet station.

With this model, it is optimal to allocate *no* bikes to inflow stations, and to allocate to outflow stations the minimal number of bikes that ensures one does not run out before time t, assuming the capacity constraints and the overall bike limit b are not binding. These “knife-edge” solutions do not allow for any randomness in the flows, but that is how we formulated the model and, as such, the model is doing as it should. This feature of the model is particularly problematic when we think about balanced, or near-balanced stations. Suppose $\lambda_i = \mu_i$ for some Station i. Then according to our fluid model, it makes no difference how many bikes we allocate to the station! We could allocate no bikes and the model would assume all customers at that station will be happy. Instead, we expect that if the rates are equal, then the station should be half full at the start of rush hour. This is the case, as a more refined model discovers.

4.2 A Continuous-Time Markov Chain Model

Now suppose that we model the flows of customers wanting to pick up bikes at the different stations as independent time-homogeneous Poisson processes with rates μ_i, $i = 1,2,\ldots,S$, where we again assume we know μ_i for each i. Customers picking up bikes at Station i choose Station j as their destination with probability P_{ij} independently of all else. Under the admittedly very strong assumption that bike stations never run out of bikes, arguments related to the splitting and merging of Poisson flows allow us to conclude that we can model the number of bikes present at Station j as an $M/M/1/r_j$ queue, with service rate μ_j and arrival rate $\lambda_j = \sum_i \mu_i P_{ij}$, independently of all other stations!

The assumption that bike stations never run out of bikes is very strong, but may be approximately satisfied if the system is well dimensioned (the rack capacities r are well chosen) and well stocked at the start of rush hour (the x vector is nearly optimal). Moreover, the simplifications we obtain are dramatic, so that we are compelled to try this model.

Our goal is still to minimize the expected loss from annoyed customers during the morning rush. The decision variable vector is still x, where x_i is the number of bikes we allocate to Station i at time 0, the beginning of the morning rush hours. Let $E_x(\cdot)$ be the expected value assuming that we start with x bikes at a station, so $E_x(\cdot) = E(\cdot|X(0) = x)$. Here $X(s)$ is a random vector, the ith component of which gives the number of bikes at Station i at time s. The contribution of Station i to the objective function can then be shown to be

$$\mu_i E_x \int_{s=0}^{t} \mathbb{1}\{X_i(s) = 0\} ds + \lambda_i E_x \int_{s=0}^{t} \mathbb{1}\{X_i(s) = r_i\} ds, \tag{5}$$

which depends on the initial bike vector x only through x_i. The objective function is the sum of these terms over each station i, and the feasible region remains the same as in (4).

It turns out that one can compute (5) very efficiently without simulation. Also, one can show that (5) is discretely convex in x, which makes the optimization easy if we relax the integrality constraint.

The CTMC model improves over the fluid model in that it captures the stochastic nature of bike flows. Unlike the fluid model, an empty (full) station will not stay empty (full) for the rest of the rush period. The predictions of this model are very much in line with our intution from the fluid model, and the knife-edge cases are resolved with more intuitive allocations. Moreover, stations with perfectly balanced rates ($\lambda_i = \mu_i$) have $x_i = r_i/2$, i.e., are half full at the start of rush hour as expected. We can use the CTMC model to give starting solutions for the simulation optimization.

4.3 A Discrete Event Simulation Model

A discrete-event simulation model can capture all of the complexities we have thus far ignored. In O’Mahony’s model, each station generates trips according to a Poisson process at rate μ_i. A trip finding an available bike is in state “trip-start"; otherwise, it is in state “failed-start" and the trip is canceled. A “trip-start" entity arising at station i at time s is assigned a destination station j with probability $P_{ij}^{(s)}$ that can depend on s. The duration of a trip from i to j is Poisson distributed, which is simple if inflexible. When a bike is successfully returned to an available dock, the trip is terminated by state “trip-end." If no dock is available, the trip state becomes “failed-end." Then we generate a new trip to a nearest station that has not yet been visited and has at least 1 available dock. After a threshold of such steps we assume the user abandons the bike incurring a “bad-end" state.

The optimization problem is as before, except we use a new objective consisting of the old objective plus a penalty multiplier c_b times the expected number of failed ends in one rush hour, i.e., we

$$\underset{x}{\text{minimize}} f(x) = E_x[\#\text{failed-starts}] + E_x[\#\text{failed-ends}] + c_b E_x[\#\text{bad-ends}]. \tag{6}$$

(Our previous models did not account for bad-ends, so there is no inconsistency between models.) We cannot compute the expected values in the objective function f exactly, but we can estimate them using the sample-average approximation f_n, as introduced in §3.1, over n replications of the morning rush (7-9am).

There are several ways to use the model to estimate the optimal allocations $X(0) = x$. First, as mentioned above, the objective in the CTMC model has a continuous extension that is convex, so we might conjecture the same for this model. Thus, we might try (discrete) stochastic approximation (§3.3), estimating the gradient using finite differences. However, we have 378 bike stations, so to evaluate a single gradient estimate, we need to calculate f_n at least 378 times! This is too computationally costly, even if we expect to find a good solution within a few steps due to our convexity conjecture.

Second, we could do a random search. From the ideas in §3.5, one such method repeats the following procedure until it finds a good solution x^k to move to in each step $k = 1,2,\ldots$.

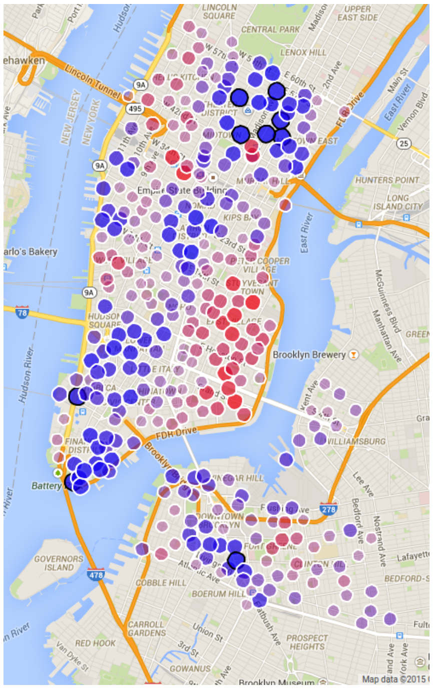
1. Swapping: Randomly choose a pair of stations i and j. Generate a trial allocation $\hat{x}^k$ by moving w bikes from station i to station j ($\hat{x}_i^k = x_i^{k-1} - w, \hat{x}_j^k = x_j^{k-1} + w$), if station capacities are sufficient.
2. Simulation: Use n simulation replications of the morning rush hour period starting with the initial allocation of the trial solution $\hat{x}^k$ to estimate the objective function, by $f_n(\hat{x}^k)$ say.
3. Evaluation: Check if the trial solution gives an improvement in the estimated objective, i.e., if $f_n(\hat{x}^k) < f_n(x^{k-1})$. If so, move to the trial solution by setting $x^k = \hat{x}^k$. Otherwise, go to Swapping.

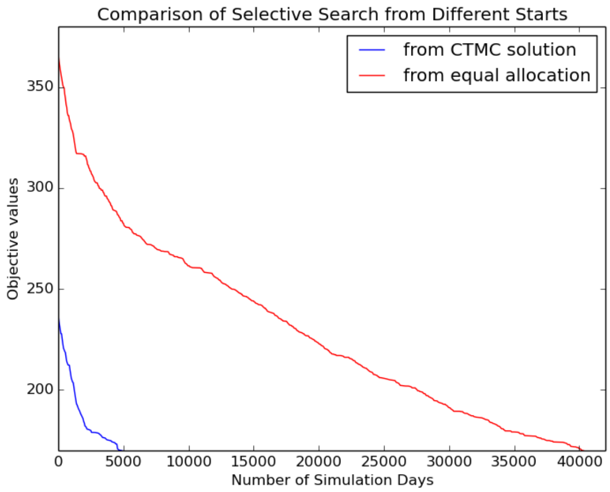
Our implementation of this search was not able to improve on initial solutions very much. The problem lies with the dimensionality of our problem. With 378 stations, there are 378 * 377 = 142,506 possible station pairs to consider to find improving solutions. But it may be that the objective is large primarily because of a modest number of very busy stations, and therefore we are very unlikely to find station pairs amongst those problem stations with this algorithm.

The random search uses the simulation model only through the output of the estimated function values. We can perform a better search by exploiting more information from the simulation. When estimating the objective (6), we can estimate the contribution from *each* station by counting how many failed starts, failed ends, and bad ends it incurs. Then, when choosing the pair of stations to swap, we can let i be the station that generates the largest number of failed ends, and j be the station that has the largest number of failed starts. In this way, we greedily rebalance stations with large net outflows and those with large net inflows.

With this method we obtained much better results starting from both an “equal” allocation that assigns bikes to stations in proportion to the station capacity, as well as from the CTMC solution. We swapped $w = 2$ bikes for the first 50 steps and $w = 1$ bike for later steps to ensure both speed and accuracy. We used sample average approximation with $n = 50$ replications. Each replication takes about 0.2 seconds, meaning function evaluations take 10 seconds. The search ends when the estimated improvement in the objective in one step is below a tolerance of 0.001. In order to estimate the true improvement in the objective of our initial and final solutions, we use another 100 independent replications. Figure 1 gives an example of the output from the two different starting solutions, with 4000 bikes and 378 stations.

The (starting) CTMC solution is already far better than proportional allocation. Both methods quickly find good solutions. Thus, a succession of models, along with the intuition we obtain from those models,





	Starting solution	
	CTMC	Equal Allocation
Starting objective	237 ± 5	366 ± 7
Ending objective	169 ± 6	166 ± 7
Final estimation	166 ± 6	172 ± 6

Figure 1 & Table 1: Left: The CTMC solution, with emptier stations in blue and fuller stations in red. Right: Comparisons of the search method starting from both the CTMC and equal-allocation solutions.

can greatly assist in making simulation optimization possible in this problem, and perhaps in many others as well.

ACKNOWLEDGMENTS

Thanks are due to Eoin O'Mahony for sharing much of his thesis work associated with the bike sharing case study, including the continuous-time Markov chain and discrete-event simulation models. This work was partially supported by NSF grant CMMI-1200315.

REFERENCES

Andradóttir, S. 2015. "A Review of Random Search Methods". In *Handbook of Simulation Optimization*, edited by M. C. Fu, Volume 216 of *International Series in Operations Research & Management Science*, 277–292. Springer New York.

Banks, J., J. Carson, B. Nelson, and D. Nicol. 2010. *Discrete-event system simulation*. 5th ed. Prentice Hall.

Barton, R. R. 2009. "Simulation Optimization Using Metamodels". In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, 230–238. Piscataway NJ: IEEE.

Boesel, J., B. L. Nelson, and S.-H. Kim. 2003. "Using Ranking and Selection to "Clean Up" after Simulation Optimization". *Operations Research* 51 (5): 814–825.

Branke, J., S. E. Chick, and C. Schmidt. 2007. "Selecting a Selection Procedure". *Management Science* 53 (12): 1916–1932.

Chau, M., and M. C. Fu. 2015. "An Overview of Stochastic Approximation". In *Handbook of Simulation Optimization*, edited by M. C. Fu, Volume 216 of *International Series in Operations Research & Management Science*, 149–178. New York: Springer.

Chau, M., M. C. Fu, H. Qu, and I. O. Ryzhov. 2014. "Simulation Optimization: A Tutorial Overview and Recent Developments in Gradient-based Methods". In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk, D. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, 21–35. Piscataway NJ: IEEE.

Chen, C., S. E. Chick, and L. H. Lee. 2015. "Ranking and Selection: Efficient Simulation Budget Allocation". In *Handbook of Simulation Optimization*, edited by M. C. Fu, Volume 216 of *International Series in Operations Research & Management Science*, Chapter 3, 45–80. New York: Springer.

Chick, S. E. 2006. "Subjective Probability and Bayesian Methodology". In *Simulation*, edited by S. G. Henderson and B. L. Nelson, Handbooks in Operations Research and Management Science. Amsterdam: Elsevier.

Fu, M. C. 2001. "Simulation Optimization". In *Proceedings of the 2001 Winter Simulation Conference*, 53–61. Piscataway NJ: IEEE.

Fu, M. C. 2002. "Feature Article: Optimization for Simulation: Theory vs. Practice". *INFORMS J. on Computing* 14 (3): 192–215.

Fu, M. C. 2015. *Handbook of Simulation Optimization*. International Series in Operations Research & Management Science. Springer New York.

Fu, M. C., G. Bayraksan, S. G. Henderson, B. L. Nelson, W. B. Powell, I. O. Ryzhov, and B. Thengvall. 2014. "Simulation Optimization: A Panel on the State of the Art in Research and Practice". In *Proceedings of the 2014 Winter Simulation Conference*, edited by A. Tolk, D. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, 3696–3706. Piscataway NJ: IEEE.

Fu, M. C., C.-H. Chen, and L. Shi. 2008. "Some Topics for Simulation Optimization". In *Proceedings of the 40th Winter Simulation Conference*, 27–38. IEEE.

Fu, M. C., F. W. Glover, and J. April. 2005. "Simulation Optimization: A Review, New Developments, and Applications". In *Proceedings of the 37th Winter Simulation Conference*, 83–95. Piscataway NJ: IEEE.

Hachicha, W., A. Ammeri, F. Masmoudi, and H. Chachoub. 2010. "A Comprehensive Literature Classification of Simulation Optimisation Methods". Mpra paper, University Library of Munich, Germany.

Henderson, S. G. 2005. "Should We Model Dependence and Nonstationarity, and If So How?". In *Proceedings of the 2005 Winter Simulation Conference*, edited by M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 120–129. Piscataway NJ: IEEE.

Henderson, S. G., and B. L. Nelson. (Eds.) 2006. *Simulation*. Handbooks in Operations Research and Management Science. Amsterdam: Elsevier.

Hong, L., and B. Nelson. 2009. "A Brief Introduction to Optimization via Simulation". In *Proceedings of the 2009 Winter Simulation Conference*, 75–85. Piscataway NJ: IEEE.

Hong, L., B. Nelson, and J. Xu. 2015. "Discrete Optimization via Simulation". In *Handbook of Simulation Optimization*, edited by M. C. Fu, Volume 216 of *International Series in Operations Research & Management Science*, 9–44. Springer New York.

Hong, L. J., and B. L. Nelson. 2006. "Discrete Optimization via Simulation Using COMPASS". *Oper. Res.* 54 (1): 115–129.

Hugan, J. C. 2014. "A Practical Look at Simulation Project Management". In *Proceedings of the 2014 Winter Simulation Conference*, 98–102. Piscataway, NJ: IEEE.

Jalali, H., and I. Van Nieuwenhuyse. 2015. "Simulation Optimization in Inventory Management: A Classification". *IIE Transactions* To appear.

Kim, S., R. Pasupathy, and S. G. Henderson. 2015. "A Guide to Sample Average Approximation". In *Handbook of Simulation Optimization*, edited by M. C. Fu, Volume 216 of *International Series in Operations Research & Management Science*, 207–243. New York: Springer.

Kim, S.-H., and B. L. Nelson. 2006. "Selecting the Best System". In *Simulation*, edited by S. G. Henderson and B. L. Nelson, Volume 13 of *Handbooks in Operations Research and Management Science*. Amsterdam: Elsevier.

Kleijnen, J. P. C. 2015. “Response Surface Methodology”. In *Handbook of Simulation Optimization*, edited by M. C. Fu, Volume 216 of *International Series in Operations Research & Management Science*, 277–292. Springer New York.

Law, A. M. 2007. *Simulation Modeling and Analysis*. 4th ed. New York: McGraw-Hill.

Luo, J., and L. J. Hong. 2011. “Large-scale Ranking and Selection using Cloud Computing”. In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, 4051–4061. Piscataway, NJ: IEEE.

Luo, J., L. J. Hong, B. L. Nelson, and Y. Wu. 2014. “Fully Sequential Procedures for Large-scale Ranking-and-selection Problems in Parallel Computing Environments”. *Manuscript*.

Luo, Y., and E. Lim. 2011. “Simulation-based Optimization over Discrete Sets with Noisy Constraints”. In *Proceedings of the Winter Simulation Conference*, 4013–4025. Piscataway NJ: IEEE.

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.

Nelson, B. L. 1995. *Stochastic Modeling: Analysis and Simulation*. Mineola, NY: Dover Publications, Inc.

Ni, E. C., D. F. Ciocan, S. G. Henderson, and S. R. Hunter. 2015. “Efficient Ranking and Selection in High Performance Computing Environments”. Working paper.

Ólafsson, S. 2006. “Metaheuristics”. In *Simulation*, edited by S. G. Henderson and B. L. Nelson, Handbooks in Operations Research and Management Science, 633–654. Amsterdam: Elsevier.

O’Mahony, E., and D. B. Shmoys. 2015. “Data Analysis and Optimization for (Citi)Bike Sharing”. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, 687–694. AAAI.

Pasupathy, R., and S. G. Henderson. 2011. “SimOpt : A Library of Simulation Optimization Problems”. In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, 4080–4090. Piscataway NJ: IEEE.

Pasupathy, R. and S. G. Henderson 2012. “SimOpt”. Accessed Apr. 28, 2015. http://www.simopt.org.

Rinnooy Kan, A. H. G., and G. T. Timmer. 1987. “Stochastic Global Optimization Methods Part I: Clustering Methods”. *Mathematical Programming* 39:27–56.

Savage, S. 2015. “The Flaw of Averages”. Accessed Apr. 28, 2015. http://flawofaverages.com.

Schruben, L. 2010. “Simulation Modeling for Analysis”. *ACM Trans. Model. Comput. Simul.* 20 (1): 2:1–2:22.

Shi, L., and S. Ólafsson. 2000. “Nested Partitions Method for Global Optimization”. *Oper. Res.* 48 (3): 390–407.

Starfield, A. M., K. A. Smith, and A. L. Bleloch. 1994. *How to Model It: Problem Solving for the Computer Age*. Burgess.

Wang, H., R. Pasupathy, and B. W. Schmeiser. 2013. “Integer-Ordered Simulation Optimization using R-SPLINE: Retrospective Search using Piecewise-Linear Interpolation and Neighborhood Enumeration”. *ACM TOMACS* 23 (3).

Yan, D., and H. Mukai. 1992. “Stochastic Discrete Optimization”. *SIAM J. Control Optim.* 30 (3): 594–612.

AUTHOR BIOGRAPHIES

NANJING JIAN is a PhD student in the School of Operations Research and Information Engineering at Cornell University. Her research is in simulation optimization, with concentration in detecting the structure of noisy black-box functions. Her email address is nj227@cornell.edu.

SHANE G. HENDERSON is a professor in the School of Operations Research and Information Engineering at Cornell University. His research interests include discrete-event simulation and simulation optimization, and he has worked for some time with emergency services and bike sharing applications. He co-edited the Proceedings of the 2007 Winter Simulation Conference. His web page is http://people.orie.cornell.edu/shane.