

# ACCOUNTING FOR RANDOMNESS IN HEURISTIC SIMULATION OPTIMIZATION

Justin Boesel and Barry L. Nelson  
Department of Industrial Engineering & Management Sciences  
Northwestern University  
Evanston, Illinois 60208 USA  
E-mail jboesel@nwu.edu and nelsonb@nwu.edu

## KEYWORDS

Operations research, Optimization, Stochastic, Discrete simulation, Statistical packages

## ABSTRACT

Research on the optimization of stochastic systems via simulation often centers on the development of algorithms for which global convergence can be guaranteed. Applications of optimization via simulation, on the other hand, typically involve search heuristics that have been successful in deterministic settings. Search heuristics give up on global convergence in order to be more generally applicable and to yield rapid progress toward good solutions. Unfortunately, most commercial implementations do not formally account for the randomness in simulation responses, meaning that their progress may be no better than a random search if the level of randomness is high. In addition, they do not provide statistical guarantees about the goodness of the final results. In this paper, we report on the work we have done to uncouple the error control for the search from the error control for the final solution. We also report on our implementation of this work in software developed for JGC Corporation of Japan.

## THE PROBLEM

Like many organizations, JGC, a Japanese construction management company, uses simulation to evaluate and compare proposed designs for facilities such as pharmaceutical plants, oil refineries and automobile manufacturing plants. A JGC research supervisor noticed that his engineers spent a great deal of time adjusting model decision variables (such as buffer size) and comparing the output results (such as work-in-process inventory). Furthermore, he recognized that the conclusions drawn from these com-

parisons were not guaranteed to be statistically valid. To remedy these shortcomings, JGC approached Northwestern University, asking for a simulation-optimization package that could provide good results on a broad range of problems in a reasonable amount of time, and provide statistical guarantees on those results.

From an optimization viewpoint, several difficulties emerge. First, the optimization approach needs to handle simulation models that may combine integer decision variables (such as the number of drills in a machine shop), continuous decision variables (such as conveyor speed in an assembly plant or flow rate in a pharmaceutical plant) and categorical decision variables (such as queue discipline or scheduling rules). This means that some traditional simulation-optimization techniques, such as gradient-search, cannot always be applied.

Second, the response properties of the problems are unknown. That is, no exploitable properties, such as convexity or continuity, can be assumed. Not surprisingly, this makes the task of finding the best design much more difficult because it prevents us from inferring anything about solutions that are not explicitly evaluated.

Third, the responses are stochastic, so one needs multiple (and perhaps very many) replications to get reliable information on a single solution.

## Existing Approaches

Broadly speaking, two general approaches have been developed for stochastic simulation-optimization problems. Academic research has focussed on approaches for which asymptotic convergence to the global optimum solution can be proven as the number of replications or the run length approach infinity. These procedures may not seek rapid improvement in the early stages of the algorithm, and provide no statistical guarantees for a finite number of replications. For these reasons they are not widely applied

in industrial settings. See (Fu 1994, Jacobson and Schruben 1989) for review articles.

Most applied simulation-optimization approaches use heuristic search procedures—including genetic algorithms, tabu search and pattern search—that were designed for use in a deterministic setting. Typically, the number of replications taken at each solution is preset by the user. While these approaches often find good solutions quickly, they may also devolve into a random search if the level of sampling variability is high or the user has set the number of replications too low. On the other hand, these procedures may be overly conservative and slow if the user sets the number of replications too high or the level of sampling variability is low.

### Our Approach

Our approach has two separate components:

1. **Search:** Like some commercially available packages, our algorithm uses a heuristic search procedure (genetic algorithm) to seek out better solutions. Unlike any commercially available package, our algorithm uses variance information to adjust the number of replications taken at each solution during the search. This provides adequate (but not excessive) error control during the search, keeping it from blindly devolving into a random search.
2. **Selection:** In order to provide the user with a statistical guarantee as to which of the visited solutions is the best, our algorithm uses a screen-and-select procedure developed by (Nelson et al. 1998). This procedure screens out clearly inferior solutions (those which are very unlikely to be the best), and then performs additional replications on the remaining solutions to determine which is the best. The procedure guarantees that the returned solution is within  $\delta$  of the best solution visited by the search with probability  $1 - \alpha$ . The user-defined parameter  $\delta$  is the smallest difference in expected performance that is practically significant to the user, while  $1 - \alpha$  is the overall confidence level that the user desires. Small  $\delta$  and large  $1 - \alpha$  imply that more simulation effort will be expended to achieve the user's goals.

### Compromises

Because of the difficulties mentioned above (lack of known response properties, stochastic response and limited time), our algorithm does not guarantee that it returns the best solution over the entire solution

space, just over the solutions visited by the search procedure. In other words, we do not make statements about unvisited solutions. Of course, if the search procedure exhausts the solution space and visits all possible solutions, then the statistical guarantee applies globally. Because exhaustion is possible in smaller problems, we designed the software to do so if the user has provided enough time.

### SOFTWARE

The software Northwestern delivered to JGC earlier this year has five interrelated components:

1. **Interface:** The interface allows the user to define the simulation-optimization problem by defining the amount of time available, defining and setting the ranges of the decision variables, selecting the simulation model or models to be evaluated, setting the desired level of statistical confidence, and setting the practically significant difference  $\delta$ .
2. **Solution Generators:** All generated solutions are evaluated by the simulator and passed on to the screen-and-select procedure. The software currently has four methods for generating new solutions.
  - (a) **User-Defined Solutions:** Because the engineer developing the simulation model usually has good ideas about what solutions are promising, the software allows the user to input these solutions at the beginning of the algorithm.
  - (b) **Extreme Point Finder:** Because good solutions often lie at the extreme points of the solution space, our software generates all of the extreme (vertex) point solutions at the beginning of the algorithm. These extreme points may later be fed into the genetic algorithm, ensuring that it provides an adequately broad search of the solution space.
  - (c) **Exhaustor:** On smaller problems it often makes sense to simply evaluate all possible solutions. Our software explicitly exhausts the solution space if there is adequate time. The software decides whether there is adequate time by observing the average time to evaluate a solution. Exhaustion is desirable because the statistical guarantee returned under exhaustion covers the entire solution space.

(d) **Genetic Algorithm (GA):** A genetic algorithm is a “population-based” search algorithm as opposed to a “point-based” algorithm in that in each iteration it simultaneously keeps a number of solutions active. A GA uses the Darwinian concepts of “natural selection” and “survival of the fittest” to produce improved solutions. Essentially, a GA assigns better solutions a higher probability of survival, where survival means the ability to pass on characteristics to future populations of solutions. These characteristics are passed on by combining or mating parent solutions to form a new population of child solutions. The genetic algorithm is initialized by filling the first population with the extreme points and the best of the user-defined solutions.

3. **Screen-and-Select Procedure:** Newly generated solutions are sent to the screen-and-select procedure, which screens out clearly inferior solutions and requests additional replications for unscreened solutions. This procedure provides the statistical guarantee that makes our software unique among simulation-optimization packages.
4. **Simulator:** Central to the software is a simulation package, which evaluates each alternative produced by the solution generators. Our software links to the AweSim! simulation package (Symix Corp./Pritsker Division), which is used by JGC. The user develops a simulation model, independent of our software. The user then defines the objective function, which can be any function of any combination of simulation outputs, in a C++ “user insert.” The user insert also provides the “hooks” that allow our software to control AweSim!.
5. **Database:** Because the solution generators may produce a large number of alternatives, each of which has unique parameter settings and output data, we maintain a database to record this information. Each unique solution generated has a record in the database. Furthermore, because a GA tends to generate the same solution more than once, and because we want to avoid wasting simulation effort on repeat evaluations, the genetic algorithm first checks the database to see if a solution has been evaluated previously. If not, the GA requests the information from the simulator. The simulator writes all output information to the database, while the screen-and-select procedure writes status information (such

as “screened” or “unscreened”) to the database.

The database, which is in Microsoft Access format, also enables the user to analyze solution output after the simulation-optimization run has concluded.

Our software, dubbed **Scenario Seeker**, runs under Windows 95 and Windows NT. We developed the interface in Visual Basic, while the solution generators and the screen-and-select procedure were written in C++. We licensed GALib—a C++ library of genetic algorithms written by David Wall at M.I.T.—to develop our GA.

## ERROR CONTROL DURING THE SEARCH

An iteration of a GA is defined by creating a new population of  $m$  solutions from a current population of  $m$  solutions. Composition of the next population of solutions depends on which solutions in the current population are selected to form the parents for the next generation.

In a typical deterministic GA that employs rank-based selection, the solutions in a population are ranked from best to worst according to their performance (objective function value), and a probability of selection is assigned based on these ranks. For our discussion, it is more convenient to use antiranks, so that the rank- $m$  solution is the best in the current population, and the rank-1 solution is the worst. For some constant  $q \geq 2$ , a  $q$ -tournament selection procedure (Bäck, 1996) is equivalent to assigning probability  $p_i = (i^q - (i - 1)^q)/m^q$  to the  $i$ th ranked solution, so that the expected number of copies of the  $i$ th ranked solution that will be in the parent pool for the next population is  $mp_i$ .

In a stochastic simulation it is not possible to conclusively rank any population of solutions without expending excessive simulation effort (number of replications). Therefore, our approach is to expend enough simulation effort to achieve *stochastic equivalence* for some important characteristic of our stochastic GA and a corresponding deterministic GA. Here we will describe preserving  $mp_m = m(1 - ((m-1)/m)^q)$ , the expected number of copies of the *best* solution in the current population that enter the parent pool for the next population. Stated in GA terminology, we try to match the “selective pressure” in a stochastic GA to a desirable selective pressure in a deterministic GA. Notice that in a  $q$ -tournament a larger value of  $q$  implies a larger selective pressure.

Since we cannot rank the solutions with certainty, we expend just enough simulation effort to divide

them into a minimum number of distinct *groups* of solutions. We then assign the same selection probabilities to all solutions within a group, specifically the average selection probability across all of the ranks in the group.

For instance, if the best group contains  $g$  solutions, and we are highly confident that the best solution is in that group, then the expected number of copies of the best solution that will be in the parent pool for the next population is

$$\frac{m}{g} \left( 1 - \left( \frac{m-g}{m} \right)^q \right)$$

while the desired expected number is  $m(1 - ((m-1)/m)^q)$ . We can therefore obtain stochastic equivalence—in terms of expected number of copies of the best solution—by setting  $q'$  so that

$$\frac{m}{g} \left( 1 - \left( \frac{m-g}{m} \right)^{q'} \right) = m \left( 1 - \left( \frac{m-1}{m} \right)^q \right).$$

That is, we increase the pressure parameter to  $q'$  so as to achieve the same expected number of copies of the best solution as we would achieve in a deterministic GA with parameter  $q$ . Of course, in order for this matching to be feasible at least a minimum number of groups must be formed; this minimum is a function of the population size  $m$  and desired value of  $q$ . The **Scenario Seeker** software dynamically controls the number of replications obtained at each solution to insure that at least the minimum number of groups can be formed.

## ERROR CONTROL AFTER THE SEARCH

At the conclusion of the search phase, the GA has explored some portion of the decision space, uncovering good solutions and (quite likely) many poor solutions as well. We therefore turn our attention to separating those solutions into the best, near best and inferior solutions. Since we apply only enough error control in the search phase to insure that the search makes progress, it is quite likely that there is too much sampling error in the performance estimates to make these finer distinctions. In this section we describe a screen-and-select procedure that takes the output of our search, eliminates clearly inferior solutions, and obtains enough additional replications to separate the best and near-best from the rest. This procedure is based on (Nelson et al. 1998).

Suppose that at the end of the search phase  $k$  distinct solutions have been simulated. Let  $n_i$  be the number of replications obtained from solution  $i$ , for  $i = 1, 2, \dots, k$ , let  $X_{ij}$  be the output from the  $j$ th

replication of solution  $i$ , and let  $\delta$  represent the minimum practically significant difference in performance that the user considers worth detecting. Under the assumption that the  $X_{ij}$  are approximately normally distributed, and that larger expected performance is better, the following procedure retains the solution with the largest true mean with probability  $\geq 1 - \alpha_0$ , while screening out inferior solutions:

1. Set  $t_i = t_{(1-\alpha_0)^{\frac{1}{k-1}}, n_i-1}$ , where  $t_{\beta, \nu}$  is the  $\beta$  quantile of the  $t$  distribution with  $\nu$  degrees of freedom.
2. Compute the sample means and variances  $\bar{X}_i$  and  $S_i^2$  for  $i = 1, 2, \dots, k$ . Let

$$W_{ij} = \left( \frac{t_i^2 S_i^2}{n_i} + \frac{t_j^2 S_j^2}{n_j} \right)^{1/2}$$

for all  $i \neq j$ .

3. Set

$$I = \{i : \bar{X}_i \geq \bar{X}_j - (W_{ij} - \delta)^+, \forall j \neq i\}.$$

4. Return  $I$ .

The number of solutions in  $I$  will typically be much smaller than the  $k$  solutions explored by the search. The subset  $I$  contains, with high probability, the best and near best solutions, as well as some solutions that could not be eliminated due to sampling error. The selection procedure described below obtains enough additional replications to make these distinctions clear. In the procedure,  $h$  is a critical value that depends on  $k$ ,  $n_i$  and a desired confidence level  $1 - \alpha_1$ :

5. If  $I$  contains a single solution, then select that solution. Otherwise, for all  $i \in I$  compute the total sample size

$$N_i = \max \left\{ n_0, \left\lceil \left( \frac{h S_i}{\delta} \right)^2 \right\rceil \right\}.$$

6. Obtain  $N_i - n_0$  additional replications from all solutions  $i \in I$ . Compute the overall sample means  $\bar{X}_i$  for  $i \in I$ .
7. Select as best the solution with the largest  $\bar{X}_i$ .

The combination of the screening and selection procedure guarantees, with probability  $\geq 1 - \alpha_0 - \alpha_1$ , that the selected solution is either the best, or within  $\delta$  of the best, of all the solutions visited by the search.

## EXAMPLE

In the software delivered to JGC, we included an example of a manufacturing facility design problem adapted from the AweSim! User's Guide. In brief, the problem is to find out how many of each of four different types of machines are required to achieve a throughput of 190 parts in 80 hours at minimum cost. The following costs are incurred: \$1000 for each Operation 10 machine, \$3000 for each Operation 20 machine, \$2000 for each Operation 30 machine, \$6000 for each Flexible machine (which can perform any operation), and  $\$100 \times (190 - \text{actual throughput})^2$  as a penalty for under- or overachieving the target throughput. Therefore, our objective is to minimize the expected value of

$$1000 \times (\# \text{ of Operation 10 machines}) + \\ 3000 \times (\# \text{ of Operation 20 machines}) + \\ 2000 \times (\# \text{ of Operation 30 machines}) + \\ 6000 \times (\# \text{ of Flexible machines}) + 100 \times (190 - T)^2$$

where  $T$  is the observed throughput, the only random variable in the objective. The required level of statistical confidence was set at 90%, and the indifference level was set at  $\delta = \$5000$ . The feasible numbers of machines of each type were set from 1 to 10, so there were  $10^4 = 10,000$  possible combinations.

We allowed the program to run for 5 hours. Of the 1,639 machine combinations evaluated by the software, the best combination encountered was (4,1,2,2); that is, 4 Operation 10 machines, 1 Operation 20 machine, 2 Operation 30 machines, and 2 Flexible machines. This combination yielded an estimated expected cost of \$28,500. We are guaranteed that this is the best solution visited by the search, or within \$5000 of the best, with 90% confidence. The next best solution encountered, (3,1,7,2), had an estimated expected cost of \$38,460.

The best combination was the 747th solution encountered, so it was generated about midway through the 5-hour run. The best combination found in the first 100 solutions was (1,4,1,5), with a cost of \$46,380. To give a sense of the range of costs across solutions, the worst combination encountered was an extreme point (1,10,1,1), which had an estimated cost of \$1,225,150.

## CONCLUSIONS

Our approach to simulation-optimization has two distinct phases: The first phase is a search for good solutions, and the second phase is a screen-and-select process to find the best among these solutions. In each of these phases we have incorporated adaptive error

control so that our approach expends adequate—but not excessive—simulation effort to deal with sampling variability.

In the search phase, we view error control as “stochastic equivalence” with respect to some important property of a GA. In this paper we presented a method that yields stochastic equivalence to one such property, the expected number of copies of the best solution that are passed through one GA iteration. We are currently developing methods that extend stochastic equivalence to the expected number of copies of *each* solution, not just the best, that are passed through each GA iteration.

In the screen-and-select phase, there are two goals: to eliminate or screen out as many inferior solutions as possible without expending additional simulation effort, and to use minimal additional effort to separate the surviving solutions into the best, near best and all others, while maintaining an overall statistical guarantee of being correct. Although the procedure used in **Scenario Seeker** provides the desired guarantee, it is statistically conservative, meaning that there are opportunities to improve its efficiency (reduce the simulation effort required).

## ACKNOWLEDGMENTS

This work was sponsored by JGC Corporation of Japan, National Science Foundation Grant Number DMI-9622065, Systems Modeling Corporation and Pritsker Corporation.

## REFERENCES

- Bäck, T. 1996. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, New York.
- Fu, M. 1994. “Stochastic Optimization via Simulation: A Review.” *Annals of Operations Research* 53:199–248.
- Jacobson, S. and L. Schruben. 1989. “A Review of Techniques for Simulation Optimization.” *Operations Research Letters* 8:1–9.
- Nelson, B. L., J. Swann, D. Goldsman and W. Song. 1998. “Simple Procedures for Selecting the Best Simulated System when the Number of Alternatives is Large,” Research Report, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL.