IIE Transactions (2003) **35**, 221–229
Copyright © 2003 "IIE"
0740-817X/03 $12.00 + .00
DOI: 10.1080/07408170390175459

A framework for simulation–optimization software

JUSTIN BOESEL[1], BARRY L. NELSON[2,*] and NOBUAKI ISHII[3]

[1]*Information and Technology Center, The MITRE Corporation, 1820 Dolley Madison Blvd. McClean, VA 22102, USA*
[2]*Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208-3119, USA*
E-mail: nelsonb@nwu.edu
[3]*JGC Corporation, Yokohama World Operation Center, 3-1, Minato Mirai 2-chime, Nishi-ku, Yokohama 220-60, Japan*

Received May 1999 and accepted March 2000

Research on the optimization of stochastic systems via simulation often centers on the development of algorithms for which global convergence can be guaranteed. On the other hand, commercial software applications that perform optimization via simulation typically employ search heuristics that have been successful in deterministic settings. Such search heuristics give up on global convergence in order to be more generally applicable and to yield rapid progress towards good solutions. Unfortunately, commercial applications do not always formally account for the randomness in simulation responses, meaning that their progress may be no better than a random search if the variability of the outputs is high. In addition, they do not provide statistical guarantees about the "goodness" of the final results. In practice, simulation studies often rely heavily on engineers who, in addition to developing the simulation model and generating the alternatives to be compared, must also perform the statistical analyses off-line. This is a time- and labor-consuming process. In this paper, we report on the work we have done to implement statistical error control within a heuristic search procedure, and on our automated procedure to deliver a statistical guarantee after the search procedure is finished. We describe how we implemented these techniques in software developed for JGC Corporation of Japan.

1. 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, chemical refineries and automobile manufacturing plants. Simulation is especially important to firms like JGC who must propose designs that satisfy client requirements within a limited time and budget. In an effort to optimize, or at least improve, the design of facilities, these firms use simulation to evaluate and compare alternative designs.

At JGC, many different simulation studies are conducted simultaneously. Some examples of such studies are described below:

Design of Material Handling System (MHS) in a pharmaceuticals plant: The MHS consists of a large Automated Storage and Retrieval System, Automated Guided Vehicles (AGVs), AGV stations, lifters and conveyors. The design variables include the number of AGVs, load-per-AGV and the AGV routings. The performance criterion is a single-dimension function based upon AGV utilization, waiting time of each transportation order, and the overall investment cost.

Design of Liquefied Natural Gas (LNG) tanker transportation system: An LNG transportation system consists of LNG tankers, LNG loading and unloading facilities and LNG storage facilities. Design variables include LNG tanker size, number of LNG tankers required, capacity of loading and unloading facilities (including the number of jetties) and LNG tank capacity in the shipping and receiving sites. Simulation is used to evaluate a cost-based performance criterion for each alternative design.

Buffer allocation in an automobile engine assembly line: In an auto assembly line, a larger buffer (queue) between work stations can increase workstation utilization, but may also drive up space requirements. Simulation evaluates each alternative design using a cost function that weighs these competing factors.

Despite the availability of simulation modeling software—which allows engineers to build models quickly—and powerful computers—which allow even large complicated models to run relatively quickly—optimization via simulation is still a time- and labor-consuming process. For complicated facilities, like the ones described above, an analyst may encounter a large number of design alternatives. Furthermore, an analyst may spend a great deal of time on the analysis of simulation outputs, such as ensuring that simulations have reached a steady state and determining whether observed differences between systems are statistically significant. As a result, it

*Corresponding author

usually takes firms like JGC several months to complete a simulation study. This time is critical for engineering firms because they must propose a good design to their clients at an early phase of each project.

JGC wanted to reduce the amount of time required to complete a simulation study and wanted its simulation analysts to spend more of their time on model development, rather than on trial-and-error search. JGC approached Northwestern University asking for research and development of simulation-optimization software that could provide good results on a broad range of problems in a reasonable amount of computer time, and could also provide statistical guarantees on those results.

From an optimization viewpoint, projects like the ones described above present several difficulties. First, the optimization approach needs to handle simulation models that combine integer decision variables (such as the number of AGVs), continuous decision variables (such as LNG tank capacity) and categorical decision variables (such as AGV routes or scheduling rules). This means that simulation–optimization techniques designed for problems with continuous decision variables, such as gradient-search methods, cannot always be applied.

Second, the response properties of the problems are unknown. That is, no exploitable structure, such as convexity or even continuity, is known to exist. Not surprisingly, this makes the task of finding the best system much more difficult, because it prevents one from saying or inferring anything about systems that are not explicitly evaluated.

Third, the responses are stochastic, so one needs multiple (and perhaps very many) replications (or batch means, in a non-terminating simulation) to get reliable information on a single system.

Broadly speaking, two general approaches have been developed to search the solution space in simulation–optimization problems that include discrete variables. The first approach guarantees convergence to a global optimum as the number of iterations approaches infinity. These rigorous procedures typically do not seek rapid improvement in the early stages of the algorithm, and provide no statistical guarantees for a finite number of replications (or batch means). See, for instance, Andradóttir (1998) for an overview of these techniques.

The second approach, employed in most commercial simulation–optimization packages, uses heuristic optimization procedures—such as genetic algorithms, tabu search, or neural nets—that were designed for use in a deterministic setting. Typically, the number of replications taken at each system is pre-set by the user. While these approaches often find good systems quickly, they may also devolve into a random search if the level of output variability is high or if 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 if the level of output variability is low. Furthermore, these approaches do not provide statistical guarantees about the "goodness" of the observed best system. In other words, one does not know the probability that the system with the best observed value is truly the best system visited.

Our approach works in three phases:

1. *Problem Definition:* In this phase, the user defines the problem by providing the simulation model to evaluate, as well as the output measure to be optimized and the objective (maximization or minimization). Furthermore, the user must define the design variables, specify the allowable range for each variable, and choose the increments into which to divide the range of each variable. Because the software discretizes continuous variables, the user must choose increments for both discrete and continuous variables. The user must also choose the amount of clock time available for the alternative-generation-and-search phase (described below) and the required confidence level, $1 - \alpha$, for the selection-of-the-best phase (also described below). Finally, the user must select the smallest practically significant difference worth detecting, δ. For instance, if a facility produces \$1 000 000 worth of output per day, a difference of \$500 is relatively small. Thus, one could set the indifference amount, δ, to \$500.

2. *Alternative-generation-and-search:* In this phase, new systems are generated by a search procedure or other means. Our software has several "non-search" generators, including one that allows the user to input alternatives manually, so that if the user has an idea as to which systems might be good, then those systems can be evaluated early in the process. Like some commercially available packages, our software employs a heuristic search procedure (genetic algorithm) to seek out better systems. Unlike any commercially available package, our algorithm uses variance information to adjust the number of replications taken at each system during the search. This provides adequate (but not excessive) error control during the search, keeping it from blindly devolving into a random search. These techniques are developed in Boesel and Nelson (1999).

3. *Selection-of-the-best:* After the alternative-generation-and-search phase has finished, the systems are passed to a procedure that provides a statistical guarantee as to which of the generated systems is the best. Some additional simulation may be required to achieve the guarantee. Our software uses one of the procedures developed by Boesel *et al.* (1999) for this purpose.

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 system over the entire solution space, just over those systems visited by the search procedure. In other words, we are not able to make statements about unvisited systems. Of course, if the system generators exhaust the system space and visit all feasible systems, then the statistical guarantee applies to the entire system space. Because exhaustion is possible in smaller problems, we designed the software to explicitly exhaust the system space if the user has provided adequate time.

In the remainder of the paper we describe the design of the software that we developed (Section 2), how we control error during the search phase (Section 3) and how we control error during the selection phase (Section 4). In Section 5 we present an illustrative example, and in Section 6 we offer some brief conclusions.

2. Software

The software that Northwestern delivered to JGC has five inter-related components that are described below. The flow of information among these five components in each of the three phases (problem definition, alternative-generation-and-search, and selection-of-the-best) is diagrammed in Fig. 1.

1. *Interface:* The interface allows the user to define the simulation-optimization problem (amount of time available, ranges of the decision variables, etc.) and lets the user add promising systems directly into the search. As Fig. 1 shows, some of this information goes exclusively to the system generators, some goes exclusively to the selection-of-the-best procedure, and some goes to both.

2. *Alternative-generation-and-search:* The software currently has four complementary methods for generating new systems, some combination of which will be employed during a single simulation–optimization run. All of the systems are evaluated by the simulator and passed on to the database, as shown in Fig. 1.

User-defined systems: Because the engineer developing the simulation model usually has good ideas about what systems might be promising, the software interface allows the user to input these systems at the beginning of the algorithm.

Extreme point finder: Because good systems often lie at the extreme points of the system space, our software generates all of the extreme (vertex) point systems 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 system space.

Exhaustor: On smaller problems, it often makes sense to simply evaluate all possible systems. Our software explicitly exhausts the system space if there is adequate time. The software decides whether there is adequate time by observing the average time to evaluate a system. Exhaustion is desirable because the statistical guarantee returned under exhaustion covers the entire system space.

Genetic algorithm: Our software uses a genetic algorithm to search the solution space. Our genetic algorithm is initialized by filling the first population with the extreme points and the best of the user-input systems.

3. *Simulator:* Central to the software is a simulation package which evaluates each system produced by the system generators. Our software employs the AweSim! simulation package (Symix Corporation/Pritsker Division), which is used by JGC. The user develops a simulation model as usual, independent of our software, and performs a few modifications to make the decision parameters into variables. The user then defines the objective function, which can be any function of any combination of simulation outputs, in a C++ user insert, which also provides the "hooks" that allow our software to control AweSim!

4. *Selection-of-the-best procedure:* After the system-generation phase has concluded, all systems are sent from the database (described below) to the selection procedure, which provides the statistical guarantee. Although several different selection procedures are available, our software employs a sort-screen-and-select procedure that sorts systems by sample mean and requests additional replications on the observed best system. Each subsequent system is then either screened out by an earlier system, or it also receives additional replications. Under certain conditions, the procedure guarantees that the system with the best sample mean is the best or within δ of the true best system visited by the search with probability $\geq 1 - \alpha$.

5. *Database:* Because the system 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 system generated has a record in the database. Furthermore, because a GA tends to generate the same system 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 system 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 selection procedure writes status information (such as "screened" or "unscreened") to the database. These information flows are diagrammed in Fig. 1. The database also enables the user to analyze system output after the simulation–optimization run has concluded.

Our software, dubbed Scenario Seeker, runs under Windows 95, 98 and NT. We developed the interface in Visual Basic, while the system generators and the statistical procedures were written in C++. We used GALib—a C++ library of genetic algorithms written by Matthew Wall at MIT—to develop our GA.

Scenario Seeker employs an Access database to maintain all simulation results.

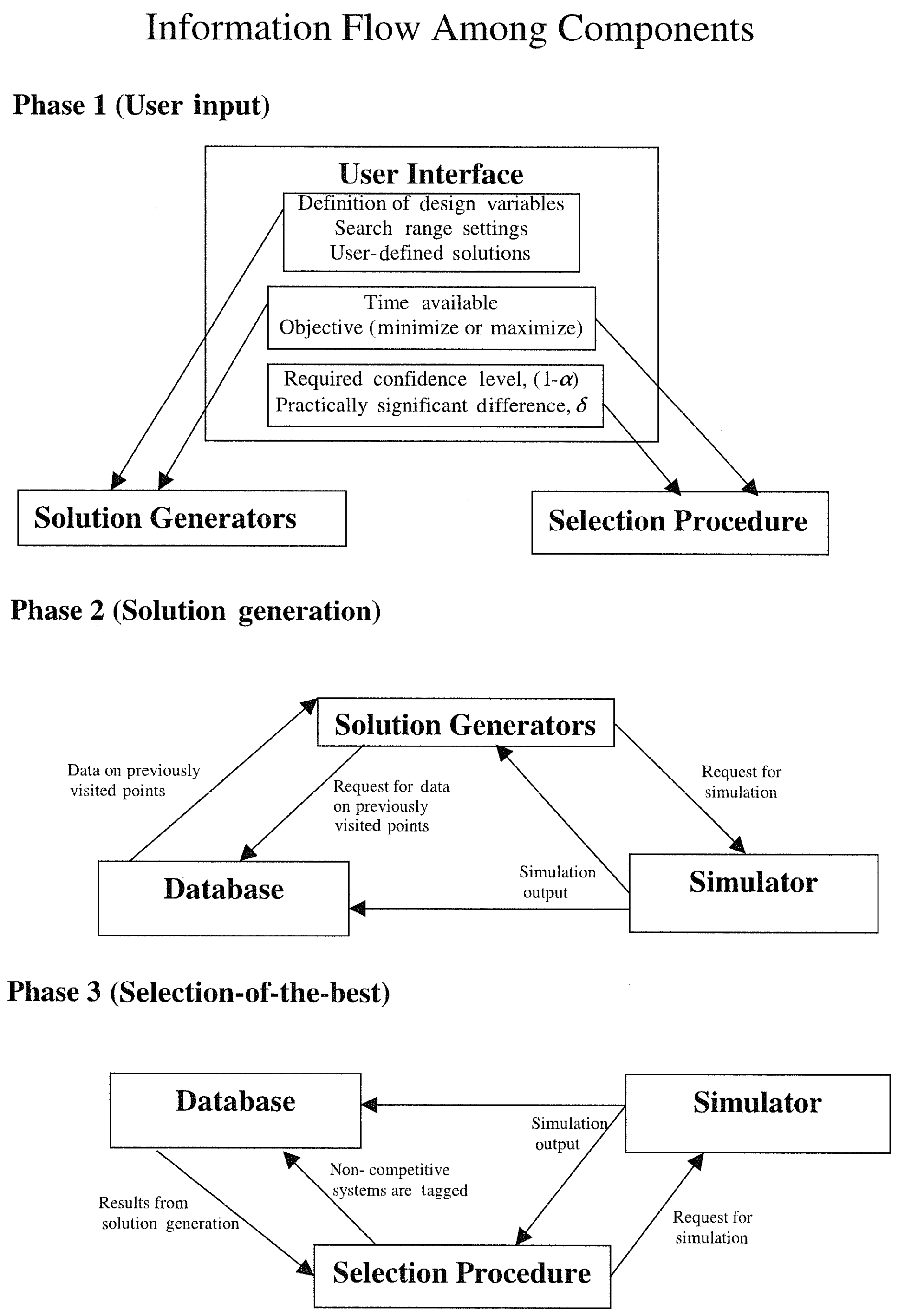

Information Flow Among Components
Phase 1 (User input)
User Interface
Definition of design variables
Search range settings
User-defined solutions
Time available
Objective (minimize or maximize)
Required confidence level, (1-α)
Practically significant difference, δ
Solution Generators
Selection Procedure
Phase 2 (Solution generation)
Solution Generators
Data on previously visited points
Request for data on previously visited points
Request for simulation
Database
Simulation output
Simulator
Phase 3 (Selection-of-the-best)
Database
Simulator
Simulation output
Non-competitive systems are tagged
Results from solution generation
Request for simulation
Selection Procedure

Fig. 1. The three phases of simulation–optimization.

3. Error control in the search

Our software's primary search method is a Genetic Algorithm (GA). A GA is a probabilistic search and optimization scheme that applies the ideas of Darwinian evolution (survival of the fittest, reproduction, and mutation) to difficult optimization problems. Essentially, a genetic algorithm starts with an initial population of m distinct systems, selects the better ones for "breeding," creates children by "mating" parent systems, "mutates" a few of the children, and starts over again by evaluating the new generation of m systems. In a deterministic setting, a genetic algorithm determines the "fitness" of each individual system by evaluating it via an objective function. Loosely speaking, the better/fitter systems are assigned a higher probability of being selected for mating.

We employed a real-valued GA in which child systems were created via two-parent recombination by taking a uniformly distributed, convex combination of each parent's decision-variable value, for continuous and integer variables, or making a random choice between the parent systems for categorical variables. Independent Gaussian mutation was used on all decision variables. The selection probability mechanism is described more fully below. The population size was held constant at $m = 30$. See Bäck (1996) or Michalewicz (1996) for a general description and overview of genetic algorithms.

To adapt a deterministic GA for use in a stochastic setting, we focused our attention upon the selection probability assignment mechanism, which is the algorithm's "guidance system." Because this mechanism depends upon objective function evaluations, this is the only GA component that could be badly misled in a stochastic setting where we have an uncertain and expensive evaluation method (simulation). The other operators, such as mating (also called crossover) and mutation do not depend directly upon fitness evaluations.

In a deterministic setting, there are several ways to translate a system's objective function value into a selection probability. In a *ranking* scheme, a system's selection probability is assigned strictly by rank in the current generation. In other words, it does not matter if the best system is 100 or 0.01 units better than the next best, the assigned probability will be the same. By contrast, in schemes such as proportional selection, if the best dominated the rest it would receive a higher selection probability than if it barely beat the rest (Bäck, 1996).

In a genetic algorithm, the selection probabilities *favor* the better systems while still giving the poorer systems some chance of survival; this keeps the search broad and robust. The mapping of systems to the selection probabilities, however, can take several different forms and still provide good search performance. These mappings, then, are not set in stone.

In a deterministic setting, we can easily and accurately evaluate the objective function, so we use these exact evaluations to determine selection probabilities. In a stochastic setting, however, it is not possible to conclusively rank any population of systems without expending excessive simulation effort (number of replications). Because the success of a GA does not depend upon a strict mapping of systems to probabilities, it is not worthwhile to expend a great deal of effort trying to get highly accurate estimates if less accurate estimates will suffice. In other words, we have an uncertain evaluation method (simulation), so we want to take advantage of the fact that the GA seems relatively robust to a range of selection probability mappings.

Therefore, our approach is to expend enough simulation effort to achieve *stochastic equivalence* for some important characteristic of our GA applied to a stochastic problem and a GA applied to a corresponding deterministic problem. Specifically, we can guarantee that, for any GA generation, the expected number of copies of the best system chosen for the mating pool will be the same in both deterministic and stochastic settings. In many instances, such a guarantee can be delivered at a reasonable sampling cost, as we show below.

In a stochastic setting, it typically requires less effort (fewer simulation replications) to form and rank *groups* of systems than to perform a comprehensive ranking of all systems individually. For instance, we may be reasonably confident that a group of $g < m$ systems contains the best system, and that each member of that group is superior to each non-member, even though we may be uncertain about the within-group ordering of the g systems. Below we show how to ensure that the expected number of copies of the best system placed in the mating pool is the same under a group-ranking procedure (which typically requires very little sampling effort) as under a comprehensive ranking procedure (which may require much more effort).

To simplify the analysis, we use anti-ranks, instead of ranks, so the best system has an index m rather than one. Using anti-ranks and q-tournament selection (Bäck, 1996), the ith anti-ranked system in the current generation is assigned a selection probability of

$$p_i = \frac{i^q - (i-1)^q}{m^q},$$

where $2 \le q \le m$ is a parameter that controls the "pressure" on good systems (larger q implies more pressure, which means highly ranked systems are assigned relatively larger values of p_i). Under this scheme, the sum of the selection probabilities for a group of size g starting with anti-rank j can be expressed as

$$\sum_{i=j}^{j+g-1} p_i = \frac{(j+g-1)^q - (j-1)^q}{m^q}.$$

In particular, the sum of the selection probabilities of the best group is

$$\sum_{i=m-g+1}^{m} p_i = \frac{m^q - (m-g)^q}{m^q}. \tag{1}$$

In a deterministic q-tournament, the selection probability of the single best system in the current generation is p_m. Because the mating pool is formed by taking m independent random draws with replacement from the current population, the expected number of copies of the best system in the mating pool is mp_m.

Suppose that we want to maintain the expected number of copies of the true best system in the mating pool at mp_m in a stochastic setting. Because we are uncertain as to which of the g members of the best group is the true best, we assign each member of the best group the same selection probability, $p_{\text{Best }g}$. Let q' be the pressure parameter used to determine the selection probabilities under grouping. As a result, each member of the best group is assigned its group's average selection probability

$$p_{\text{Best } g} = \frac{m^{q'} - (m-g)^{q'}}{g m^{q'}},$$

which is simply the right-hand side of (1) divided by g, the number of group members.

To get the same expected number of copies of the true best in the mating pool in a stochastic (and grouped) environment as in a deterministic (and ungrouped) environment, we give each member of the best group the same selection probability as the best system would receive in a deterministic environment, that is $p_{\text{Best } g} = p_m$. To do this, we set q' so that

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

implying a revised pressure parameter

$$q' = \frac{\ln(1 - g(1 - (m-1)/m)^q)}{\ln((m-g)/m)}. \tag{2}$$

To implement (2), we use a statistical grouping procedure based on Calinski and Corsten (1985) to divide the systems into non-overlapping groups. The grouping procedure begins with each system in its own group. Starting from the system with the largest (smallest) sample mean, the procedure iteratively forms larger groups by absorbing the next adjacent system. In each iteration, the absolute difference between the sample means of the largest (smallest) group member and the new candidate member is compared to an expression based on sample variances, the number of replications taken, and a studentized range statistic. If the absolute difference of the means is smaller than the expression, then the candidate is absorbed into the group. If the absolute difference of the means is larger than the expression, then the candidate system becomes the first point in the next group, which then begins absorbing adjacent systems. This process continues until the procedure reaches the smallest (largest) system.

Suppose the procedure forms c groups, where group h has g_h different systems and $g_1 + \cdots + g_c = m$, the population size. If we let G_h be the set of the indices of the systems in group h, then

$$\begin{aligned} G_1 &= \{1, 2, \ldots, g_1\} \quad (\textit{worst group}), \\ G_2 &= \{(g_1 + 1), \ldots, (g_1 + g_2)\}, \\ &\vdots \\ G_c &= \{(g_1 + g_2 + \cdots + g_{c-1} + 1), \ldots, \\ &\quad (g_1 + g_2 + \cdots + g_{c-1} + g_c)\} \quad (\textit{best group}). \end{aligned}$$

If g_c, the size of the group containing the best system, is greater than $1/p_m$, then it is not possible to match the selection pressures, because $p_m g_c > 1$, and the sum of all selection probabilities must be equal to one. In this case, one would have to go back for more data (that is, perform more simulation replications) to reduce g_c.

Once g_c is small enough, q' can be determined according to Equation (2), letting $g = g_c$. To find the individual selection probabilities, first find each group's total selection probability, then divide by the group's size to assign each system its group's average selection probability. If we let $j_h = g_1 + \cdots + g_{h-1} + 1$ be the index of the worst system in group h, the total selection probability for group h is given by the expression

$$\sum_{i=j_h}^{j_h+g_h-1} p_i = \frac{\left(\sum_{\ell=1}^{h} g_\ell\right)^{q'} - \left(\sum_{\ell=1}^{h-1} g_\ell\right)^{q'}}{m^{q'}}.$$

As a result, each individual system i in group h is assigned the average of the group's probability,

$$p_{ih} = \frac{\left(\sum_{\ell=1}^{h} g_\ell\right)^{q'} - \left(\sum_{\ell=1}^{h-1} g_\ell\right)^{q'}}{g_h m^{q'}}, \tag{3}$$

where $h = 1, 2, \ldots, c$ and $\sum_{\ell=1}^{0} \equiv 0$. This concept of stochastic equivalence based on a statistical grouping procedure was implemented in the search portion of Scenario Seeker.

Although stochastic equivalence, in terms of the expected number of occurrences of the best system in the mating pool, is an effective way to insure progress of the GA search, it has a few drawbacks. For instance, systems not in the group containing the best may receive selection probabilities much lower than they would have received in a deterministic environment, and the non-best systems grouped with the true best system may receive much higher selection probabilities. A more comprehensive measure of "stochastic equivalence," which ameliorates some of these drawbacks, is described in Boesel and Nelson (1999).

4. Error control in selection after search

At the conclusion of the search phase, the GA has explored some portion of the system space, uncovering good systems and (quite likely) many poor systems as well. We therefore turn our attention to separating those systems into the best, near best and inferior systems. Since we apply only enough error control in the search phase to ensure 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.

Several different procedures for finding the best among a large number of simulated systems are developed in Nelson *et al.* (1998) and Boesel *et al.* (1999). These procedures guarantee, with probability $\geq 1 - \alpha$, that the selected system is the true best *of all the systems visited by the search* or is within some user-defined distance, δ, of the true best. First, these procedures perform a *subset selection*, using the sample data gathered during the search phase to eliminate or *screen out* clearly inferior systems, then they perform a *selection-of-the-best* procedure, which usually requires additional replications, to determine which of the remaining systems is the true best. The primary assumption behind these procedures is that the simulation output data are normally distributed; therefore, they are most appropriate when the performance measure is estimated by the sample average of a large number of more basic outputs.

A classical selection-of-the-best procedure, such as Rinott's procedure (Rinott, 1978), assumes that system means are arrayed in the *Least Favorable Configuration* (LFC); that is, the configuration of means that is most likely to cause the procedure to fail (for further discussion, see Bechhofer *et al.* (1995)). The least favorable configuration, in this case, is the *slippage* configuration, in which the best system has a true mean that is exactly δ better than the means of the inferior systems, all of which are equal. Because Nature is rarely so malevolent, this LFC assumption can be grossly conservative. Therefore, combining a subset-selection (screening) procedure with a selection-of-the-best procedure is often more efficient than a selection-of-the-best procedure alone because clearly superior systems will screen out inferior systems. As a result, no additional sampling is required from the (screened-out) inferior systems, greatly reducing the total simulation effort needed. This is especially important in our setting, where we encounter a large number of systems, many of which are clearly inferior.

Further refinements to the combined screen-and-select procedure, described in Nelson *et al.* (1998) and Boesel *et al.* (1999), can boost efficiency dramatically. For instance, one such refinement sorts systems by their first-stage sample means and takes additional replications of the best observed system. The next-best observed system then faces screening by the best system; if it survives, additional replications are taken from it as well. Similarly, each subsequent system faces screening by all those which have preceded it.

To set-up the procedure, let n_{0i} be the number of replications that system i received during the search, let N_i be the total number of replications that system i has received if it has received second-stage (selection) sampling, and let

$$W_{ij} = \left(\frac{t_i^2 S_{0i}^2}{\tilde{N}_i} + \frac{t_j^2 S_{0j}^2}{\tilde{N}_j} \right)^{1/2},$$

where

$$\tilde{N}_i = \begin{cases} n_{0i}, & \text{if system } i \text{ has only received first-stage (search) sampling,} \\ N_i, & \text{if system } i \text{ has received second-stage (selection) sampling,} \end{cases}$$

the constant t is a quantile from the t-distribution and S_{0i}^2 is a sample variance (both defined more carefully below).

We next present a step-by-step description of the sort-screen-and-select procedure. Assume that maximization is the goal, and that the search has uncovered k potential systems. In the procedure a superscript (1) indicates a quantity computed from first-stage (search) data, while a superscript (2) indicates a quantity computed from all available data after first- and second-stage (selection) sampling.

1. Select the overall confidence level $1-\alpha$ and indifference amount δ. Set $t_i = t_{(1-\alpha/2)^{1/(k-1)}, n_{0i}-1}$ and $h = h((1-\alpha/2)^{1/(k-1)}, n_{\min}, 2)$, where $n_{\min} = \min_i\{n_{0i}\}$, h is Rinott's constant and $t_{\beta,v}$ is the β quantile of the t distribution with v degrees of freedom.
2. Let $I_0 = \emptyset$ and $J_0 = \emptyset$ (I is the set of systems still in contention to be the best, while J is the set of systems that have been eliminated from consideration).
3. Based on the data generated during the search phase, compute the first-stage sample means and variances, $\bar{X}_i^{(1)}$ and S_{0i}^2 and set $\tilde{N}_i = n_{0i}$ for all i.
4. Sort by sample mean, $\bar{X}_i^{(1)}$. Reset the indices to reflect the sorted order; that is, let $\bar{X}_i^{(1)} \geq \bar{X}_{i+1}^{(1)}$ so that $\bar{X}_1^{(1)}$ is largest.
5. For each system, $i = 1, 2, \ldots, k$, do the following:

   If $\bar{X}_i^{(1)} \geq \bar{X}_j^{(1)} - W_{ij}, \forall j \in J_{i-1}$ and $\bar{X}_i^{(1)} \geq \bar{X}_j^{(2)} - W_{ij}, \forall j \in I_{i-1}$, then system i passes the screen, so more replications are required:

   (a) Compute the total sample size N_i based on Rinott's procedure; that is:
   $$N_i = \max\left\{ n_{0i}, \left\lceil \left(\frac{hS_{0i}}{\delta} \right)^2 \right\rceil \right\},$$
   where $\lceil a \rceil$ is the least integer greater than or equal to a.

   (b) Sample $N_i - n_{0i}$ additional replications from system i, and compute the second-stage sample mean $\bar{X}_i^{(2)}$.

   (c) Place this system into group I_i, so $I_i = I_{i-1} \cup i$. Let $J_i = J_{i-1}$.

   Otherwise, system i is screened out, and should be placed into J_i, so $J_i = J_{i-1} \cup i$. No more replications are needed on system i. Let $I_i = I_{i-1}$.
6. Select as best the system $i \in I_k$ with the largest sample mean $\bar{X}_i^{(2)}$.

This provably valid sort-screen-and-select procedure uses the second-stage replications taken on the better systems to help in the screening process (see Boesel, *et al.* (1999) for the proofs). These additional replications increase $\tilde{N}_i$, thus reducing W_{ij}, making it easier to eliminate inferior systems. Sorting by first-stage sample means ensures that the better systems—the ones most likely to screen out inferior systems—receive second-stage sampling early, increasing their ability to eliminate inferior systems. Eliminating inferior systems from contention before they can receive second-stage sampling saves simulation effort. A version of this procedure is implemented in Scenario Seeker.

5. Example

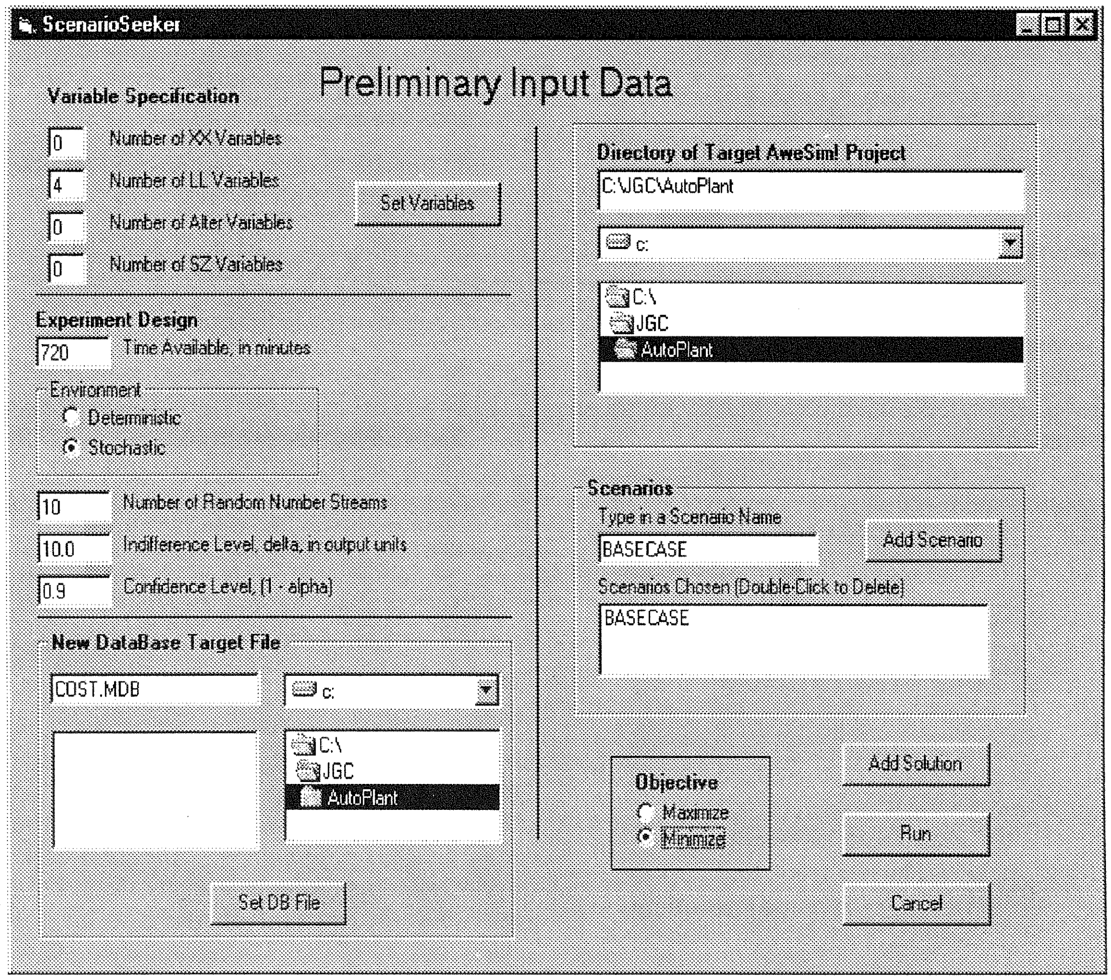
To illustrate the simulation–optimization software, JGC prepared a simulation model of the automobile assembly line problem mentioned earlier. In brief, the problem is to determine the size of four buffers in an automobile engine assembly line, while accounting for the additional production that larger buffer sizes could allow over a 2-week period. Our objective is to minimize the expected value of:

$$\$\frac{K + 1000 \times \text{Total buffer capacity}}{\text{Number of engines produced}},$$

where K is a constant representing costs not allowed to vary within our model. The capacities of the first and third buffers were allowed to range from two to 36, while the capacities of the second and fourth buffers were allowed to range from two to 60. As a result, $35^2 \times 59^2 = 4\,264\,225$ combinations were possible. The main user interface screen for Scenario Seeker (with settings for the example problem) is presented in Fig. 2.

We allowed the program to run for 12 hours. Of the 356 combinations evaluated by the software, the best combination encountered was (2,2,2,2). This combination yielded an expected per-unit cost of \$147.27. We are guaranteed that this is the best system visited by the search, or within $\delta = \$10$ of the best, with 90% confidence.

The best combination was the first system encountered, so the extreme point generator, rather than the search procedure, found the best system. To give a sense of the range of costs across systems, the worst combination encountered was (36,60,36,60), which had an estimated cost of \$256.00. Although it might well be that an analyst with sufficient training in queueing theory could have proven that the smallest possible buffer allocation would be best in this case, the reason for wanting good simulation–optimization software is to allow a practitioner without such training to obtain good results in real-world problems.



Variable Specification

0 Number of XX Variables
4 Number of LL Variables
0 Number of Alter Variables
0 Number of SZ Variables

Experiment Design

720 Time Available, in minutes

Environment
Deterministic
[✓ Stochastic]

10 Number of Random Number Streams
10.0 Indifference Level, delta, in output units
0.9 Confidence Level, (1 - alpha)

New DataBase Target File

COST.MDB
c:
C:\
JGC
AutoPlant

Directory of Target AweSim! Project

C:\JGC\AutoPlant
c:
C:\
JGC
AutoPlant

Scenarios

Type in a Scenario Name: BASECASE
Scenarios Chosen (Double-Click to Delete): BASECASE

Objective

Maximize
[✓ Minimize]

Fig. 2. Scenario Seeker's main user interface screen.

6. Conclusions

Our approach to simulation–optimization has three distinct phases: the first phase allows the user to define the problem and input promising systems; the second phase generates new systems; and the third phase uses a statistical procedure to determine which system is best. In the search segment of the system-generation phase, we have incorporated adaptive error control so that our approach expends adequate—but not excessive—simulation effort to deal with sampling variability.

One of the strengths of our approach is its separation of the search procedure from the selection procedure, which enables it to incorporate improved "component" procedures. For instance, if a problem-specific search procedure is known to work better than our general search procedure, it can be incorporated into our framework. Similarly, when better post-search "clean-up" procedures are developed, they too, can be incorporated. In fact, we believe that a better search procedure may make the post-search statistical selection procedure more efficient by generating better systems that screen out inferior systems with less simulation effort.

Acknowledgments

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

References

Andradóttir, S. (1998) Simulation optimization, in *Handbook of Simulation*, Banks. J, (ed.), Wiley, New York, ch. 9.

Bäck, T. (1996) *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press, New York.

Boesel, J. and Nelson, B.L. (1999) Designing evolutionary algorithms for stochastic optimization. Technical report, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208, USA.

Boesel, J., Nelson, B.L. and Kim, S.-H. (1999) Using ranking and selection to 'clean up' after simulation optimization. Technical report, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208, USA.

Bechhofer, R.E., Santner, T.J. and Goldsman, D. (1995) *Design and Analysis for Statistical Selection, Screening and Multiple Comparisons*, Wiley, New York.

Calinski, T. and Corsten, L.C.A. (1985) Clustering means in ANOVA by simultaneous testing. *Biometrics*, **41**, 39–48.

Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, New York.

Nelson, B.L., Swann, J., Goldsman, D. and Song. W. (1998) Simple procedures for selecting the best simulated system when the number of alternatives is large. Technical report, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL 60208, USA.

Rinott, Y. (1978) On two-stage selection procedures and related probability-inequalities. *Communications in Statistics—Theory and Methods* **A7**, 799–811.

Biographies

Justin Boesel is a Senior Simulation and Modeling Engineer at the MITRE Corporation in McLean, VA. Boesel received his Ph.D. in the Department of Industrial Engineering and Management Sciences at Northwestern University in 1999. While a graduate student at Northwestern, Boesel developed simulation–optimization theory and software for JGC, a Japanese construction management company.

Barry L. Nelson is a Professor in the Department of Industrial Engineering and Management Sciences at Northwestern University, and is Director of the Master of Engineering Management Program there. His research centers on the design and analysis of computer simulation experiments on models of stochastic systems. He has published numerous papers and two books, including *Discrete-Event System Simulation*, (Prentice Hall, 1995). In 1997 he received the Institute of Industrial Engineers Operations Research Division Award. Nelson has served the profession as the Simulation Area Editor of *Operations Research* and President of the INFORMS (then TIMS) College on Simulation. He has held many positions for the annual Winter Simulation Conference, including Program Chair in 1997.

Nobuaki Ishii is a Manager of the Management Science and Engineering Team at JGC Corporation (Nikki k.k.), Japan. He is currently developing an advanced planning and scheduling system for a food company. Ishii received his Doctor of Engineering Degree in Industrial Engineering and Management Science at Tokyo Institute of Technology in 1995. His research interests are in scheduling heuristics, simulation, life cycle engineering, and engineering economics.

Contributed by the Simulation Department