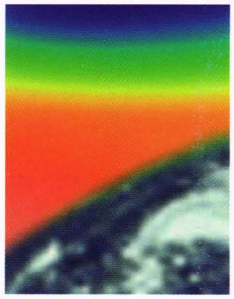


Using Genetics-Based Learning Methods to Improve Simulation Model Fidelity

Lloyd A. Best and Robert D. Sanders

Simulations offer a cost-effective alternative for modeling the complex behavior of objects. But achieving high fidelity can be difficult, especially without a skilled tactician to recommend parametric settings that cause the object to move within the desired kinematic motion. The added expense for an expert's time, however, may be too high for proof-of-principle projects on shoestring budgets. In addition, the underlying principles governing quasi-optimal object behavior are harder to extract for simulations requiring model parameters with search spaces having more than three dimensions. This problem represents a natural knowledge extraction bottleneck, as the human expert must think in four dimensions and beyond to locate areas within the model's parametric search space where quasi-optimal simulation performance will occur. To maintain cost-effectiveness and simulation fidelity, a simulation optimization tool employing the simple genetic algorithm can be used to adaptively peruse an arbitrary, complex domain space formed by a set of model parameters to provide quasi-optimal simulation performance.

INTRODUCTION

Genetic Learning for ORBIS Simulations (GLOS) is a simulation tool integrated into the background mode of the Object-oriented Rule-Based Interactive System (ORBIS), an expert system development tool used to create complex simulations. Funded by an FY 1994 Independent Research and Development project at APL, GLOS allows the user to define an arbitrary, complex, multiparametric search space representing simulation parameters that would normally be adjusted by hand. GLOS then employs the simple genetic algorithm (SGA) to explore the parametric space, finding quasi-optimal settings for the user. The goal of this project was to determine if GLOS could attain improvements in these parameters comparable to those achieved by a skilled operator, but in significantly less time. Results based on using a validated ORBIS simulation as a test bed indicate that GLOS is robust and fast, and can offer high-fidelity solutions to complex problems.

WHAT IS ORBIS?

ORBIS is an expert system development tool designed to create a variety of dynamic simulation

environments.[1] It allows for the placement of dynamic entities (represented as objects) within a synthetic environment to interactively analyze complex behavior (represented as rule sets) played out in terms of tactical doctrine and logic. Along with evaluation of operational guidance, ORBIS simulations have been used for advanced technology utility assessments, cost and effectiveness analyses, and distributed simulations on the Defense Simulation Internet.

An ORBIS simulation can run in two modes: interactive and background. During an interactive session, the user can apply a powerful capability called dynamic editing, a feature that allows "on-the-fly" editing of rule sets and object parameters in the middle of an interactive simulation run. When used in conjunction with a time-restore capability, this feature allows the operator to see the effect of "what if" considerations on the outcome of object interactions. In the background mode, Monte Carlo simulation runs are used to gather statistics, defined in terms of measures of effectiveness, to analyze the impact and effectiveness of tactical guidelines or the physical components of model object behavior.

However, to obtain high-fidelity model behavior, simulation developers must often run several background trials to locate optimal settings for parametric values associated with the rule bases and objects used in the simulation. This technique of exploring simulation parameters is frequently the only recourse for optimizing simulation-based control strategies and system designs, and is not cost-effective for simulation development for two reasons. First, an expert operator is required to supply the necessary skill and intuition to adjust the simulation model parameters, adding cost to software development. Second, a complex model can have an exhaustive, poorly understood, multiparametric search space where manual manipulation of parametric settings cannot produce quasi-optimal model behavior.

HOW DOES GLOS WORK?

The learning functionality of GLOS is based on the application of the principles of natural selection and genetics embodied in the SGA. Theoretically and empirically proven as an optimization strategy, the SGA provides a robust search over complex problem domain spaces. In this section we highlight the basics of genetic algorithm theory to give the reader a background for understanding how GLOS works. Srinivas and Patnaik,[2] Goldberg,[3] and Davis[4] are excellent sources for readers interested in a more comprehensive study of genetic algorithms.

Natural Systems

Genetic algorithms were introduced by Holland[5] in the early 1970s as computer programs that retain the mechanisms of natural systems. In nature, all species compete for limited resources (food, water, shelter) in order to survive. Successful competition is essential for survival. The traits that determine success or failure are manifested in the species through its genetic makeup. The base unit of this genetic makeup is composed of genes, which contribute certain features to the species (e.g., eye color, skin pigment, muscle tone, etc.). Collections of these genes form structures called chromosomes, which are the "blueprint" to how the species survives or adapts to its competitive environment. Success in reacting to the environment determines the fitness of the chromosome.

The most common manifestations of evolution are adaptive changes in the species itself, which increase its chances of survival. These changes necessitate changes in genetic makeup. In natural systems, "survival of the fittest" in a species is affected as genes of the more adaptive individuals survive and the weaker individuals die out. Evolution occurs naturally, as only the fitter survive to combine their more adaptive genes with others of the same species that also have survived because of more adaptive genes. The process of recombining genes during mating is called crossover of the genetic material. The whole mating process itself is called reproduction, as a new generation of species is created with a higher level of fitness than the parents. Therefore, the main observation from natural systems is that the crossover of genes of fitter individuals during reproduction leads to survival of the fittest among gene features.

Although natural systems use crossover as the main mechanism for evolving adaptive individuals, a secondary mechanism called mutation may manifest itself. Occasionally reproduction and crossover can result in the loss of certain useful genetic material. Mutation helps prevent the loss of potentially valuable genetic material that can get "weeded" out by the primary operators. It also introduces additional variation into the system.

The Simple Genetic Algorithm

The SGA operates on a set of potential solutions, called a population, to an optimization problem. Rather than a sequential search that would peruse the domain space one solution at a time, the SGA conducts a parallel search by maintaining many possible solutions. It uses an objective function to evaluate each solution within the current population. These values naturally will vary depending on the optimization

problem. Therefore, to maintain uniformity over various problem domains, a fitness function is used to normalize the objective function to a range from 0 to 1, giving the solution's worth as a possible solution. Genetic learning occurs as the best chromosomes of the current population are selected and the information encoded within their structures is interchanged to form a new population of solutions with potentially higher fitness.

The basic elements of the SGA's genetics-based search technique are presented in Fig. 1. The processing begins at generation 0 with an initial population randomly seeded with potential solutions. The fitness function is applied to each member of the population to determine each solution's fitness values. The population is next checked for convergence. Although the criteria indicating convergence will vary, the following typically apply:

- A particular best population fitness has been achieved, exceeding some predefined value.
- The difference between the population average fitness and the best population fitness approaches a predefined threshold value.
- The best population fitness value has been held for a predefined number of generations.
- The population contains multiple copies of only a few unique solutions.

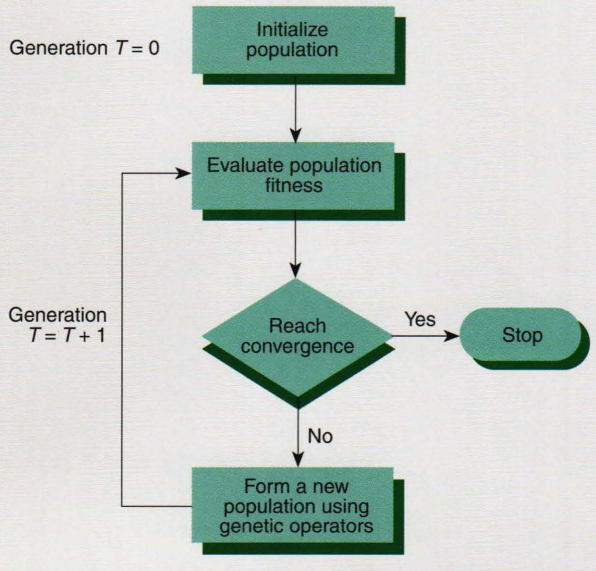


Figure 1. The simple genetic algorithm (SGA) processing cycle begins with a randomly seeded initial population and then computes each member's fitness to determine its worth as a solution to the optimization problem. When the fitness of the current population does not satisfy the predefined criteria, the SGA uses genetic operators to create a new population of potentially fitter solutions by interchanging the better notions on solving the optimization problem that are suggested by the fitter solutions. (T denotes one generation.)

When convergence has not occurred, the SGA uses genetic operators to recombine the information encoded in the best chromosomes to form the new population for the next generation. The cycle of evaluating population fitnesses and forming new populations continues until the convergence criteria are satisfied. This parallel search distinguished SGAs from point-to-point optimization methods, which can locate false peaks in multimodal search spaces.

Maintaining the proper population size is vital for the SGA to perform optimally. Populations that are too small cause the genetic algorithm to converge too quickly upon a solution, with a high probability of ending the search on a false peak. Populations that are too large cause the SGA to perform poorly, resulting in long waiting times for significant improvement in fitness of the solutions as processing proceeds from population to population. GLOS uses the genetic algorithm theory offered by Goldberg[6] to recommend the optimal population size for optimizing a set of parameters.

Each solution held within an SGA population is analogous to the chromosomes comprising the genetic makeup of a species within a natural system. Recall that the genes of the chromosome express a specific characteristic or trait for an individual of that species; therefore, GLOS adopts this scheme by representing each parameter of a simulation model as a gene. The range of parametric settings allowed for a particular parameter constitutes the traits of the gene. A collection of genes forms a chromosome and represents one possible solution to the multiparametric search space defined for the simulation model.

The Encoding Scheme

The traditional SGA works with an encoding of the parametric set for the problem being optimized, not the parameters themselves. In general, genetic algorithms require that the natural parametric set of the optimization problem be encoded as a finite-length string defined over some finite alphabet. Specifically, the SGA encodes each solution as strings of bits from a binary alphabet (0, 1). The concept of the encoding mechanism allows genetic algorithms to maintain their robustness among a wide range of optimization problems by tailoring the algorithm to meet the specific need of the problem, not vice versa.

There is no generic encoding mechanism for all problem types, since the mechanism is highly dependent on the nature of the problem. Therefore, GLOS adopts the traditional genetic algorithm encoding scheme recommended for encoding multiparametric optimization problems dealing with real parametric values. This method is called concatenated, multiparameter, mapped, fixed-point coding. Basically, the

method maps binary strings of length l over the interval specified by $[U_{min}, U_{max}]$, where U_{min} and U_{max} are the minimum and maximum values, respectively, for the parameter. Equation 1 gives the precision for the mapped coding:

$$\pi = \frac{U_{max} - U_{min}}{2^l - 1}. \quad (1)$$

GLOS allows the user to specify the (U_{min}, U_{max}) and the precision values; therefore, the encoding mechanism concerns itself only with the length l of the binary string needed to map the possible traits or values for each parameter. The string length needed for each parameter is easily determined by solving for l in Eq. 1 and obtaining Eq. 2:

$$l = \frac{\log_e \left(\frac{U_{max} - U_{min} + \pi}{\pi} \right)}{\log_e 2}. \quad (2)$$

The boxed insert demonstrates the encoding scheme adopted for GLOS.

The Fitness Function

Many optimization schemes require auxiliary information to perform properly. For example, gradient techniques require derivatives for peak climbing, and other local search procedures such as the greedy techniques of combinatorial optimization require access to most, if not all, tabular parameters. By contrast, the SGA is blind because it requires no auxiliary information, only the payoff values associated with the individual strings generated by the encoding scheme. These individual values are defined by the objective function. The fitness function is then used to normalize the objective values so that they are bound within the interval [0, 1].

Consider, for example, the optimization problem of maximizing the function $f(x) = x^2$ on the integer interval [0, 31]. Using the previously described encoding scheme with $U_{min} = 0$, $U_{max} = 31$, and precision = 1, the parametric space (0, 31) can be mapped linearly to

GLOS ENCODING MECHANISM: AN EXAMPLE

Step 1. Consider the following multiparametric set that is to be optimized.

Parameter	Minimum	Maximum	Precision
x	5	20	5
y	10	30	10

Step 2. Find the binary string length needed to map each parameter of the set using Eq. 2 (see text).

Parameter x

$$\text{Length} = \frac{\log_e \left(\frac{20 - 5 + 5}{5} \right)}{\log_e 2}$$

$$\text{Length} = 2$$

Parameter y

$$\text{Length} = \frac{\log_e \left(\frac{30 - 10 + 10}{10} \right)}{\log_e 2}$$

$$\text{Length} = 2$$

Step 3. Map binary strings to the list of possible values for each parameter, refining precision for those parameters that cannot be mapped linearly. Note that parameter x fits, but precision for parameter y is readjusted using Eq. 1 (see text).

Parameter x mapping			Parameter y mapping		
00	→ 5	} 5	00	→ 10.00	} 6.67
01	→ 10	} 5	01	→ 16.67	} 6.67
10	→ 15	} 5	10	→ 23.33	} 6.67
11	→ 20		11	→ 30.00	

Step 4. Multiparametric encoding is a concatenation of each of the individual parametric encodings.

Example string: |00|11|

x y

the set of 5-bit binary strings (00000, 11111), respectively. The next task will be to attach objective values for each string. A possible method would be to convert each binary string to a decimal equivalent using base 2 arithmetic and then square the results; e.g., the binary string 10101 decodes to 21 $(1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0)$. This result squared (441) will be the objective value for 10101. Once the objective value is computed, the fitness function would normalize that value. For the current optimization problem, a possible fitness function would be to divide the objective value by an arbitrary, large positive value (e.g., 961) that would be certain to bound the fitness within the interval [0, 1].

The Selection Mechanism

The SGA must model the survival of the fittest mechanism observed in natural systems. Simply stated, the SGA must select solutions within a population in such a way that reproduction results in fitter solutions surviving and weaker solutions dying out. The SGA uses a "roulette wheel" selection scheme. Each binary string in the population is allocated a sector or slot on the wheel, with the angle subtended by the sector at the center of the wheel equaling $2\pi(f_i / \bar{f})$, where $\bar{f}$ is the sum of all the string fitnesses and f_i is the fitness of the ith binary string. The SGA selects a binary string for reproduction if a randomly generated number in the range of 0 to 2π falls in the sector corresponding to the string. Figure 2 shows the construction of a roulette wheel for a population of binary strings and fitness values. This selection technique accomplishes "survival of the fittest," as the fitter (less fit) strings are assigned larger (smaller) sectors on the wheel, resulting in larger (smaller) probabilities of being selected.

Example population

String	Fitness	Percentage of total
01101	0.176	14.4
11000	0.599	49.2
01000	0.067	5.5
10011	0.376	30.9
Total	1.218	100.0

Roulette wheel

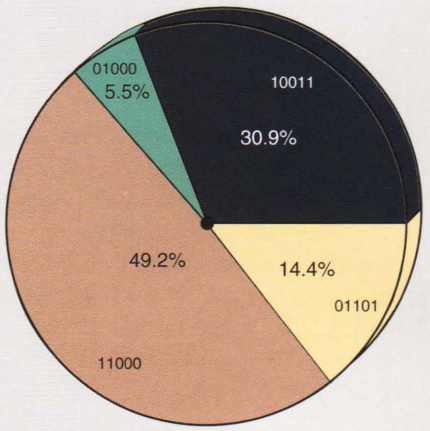


Figure 2. Example of constructing a roulette wheel. When the fitnesses of each population member are computed, the probability that each member will be selected to contribute offspring to the next generation is determined. This probability is expressed as the population member's fitness divided by the population's total fitness. The roulette wheel is weighted corresponding to each member's probability of being selected.

Genetic Operators

After roulette wheel selection picks a pair of parent strings that will contribute genes to the next population, the SGA applies the genetic operators, called crossover and mutation, to produce one pair of offspring. Crossover, the most critical operator, recombines the genetic material of the parent strings when a randomly generated number is greater than the probability of crossover (p_c), where $0 \leq p_c \leq 1$. This technique proceeds as follows. Given two binary strings of length l, a cross site is randomly selected from 1 to $l - 1$. The cross site may assume any of the $l - 1$ possible values with equal probability. The portions of the two strings beyond the crossover point are exchanged to form two new points. Figure 3 demonstrates the crossover operator. Although the SGA uses a single-point crossover, multiple-point (i.e., two cross sites or more) crossover operators have been suggested by Cavicchio[7] and Frantz.[8] They must be used with caution, however, as DeJong[9] has discovered that performance increasingly degrades with the increased number of cross points.

After crossover, the SGA applies the mutation operator to both binary strings on a bit-by-bit basis, where the mutation of one bit does not affect the probability of mutation of another bit. The SGA accomplishes the mutation process by flipping the value of the bit, i.e., changing 0 to 1 or 1 to 0. Figure 4

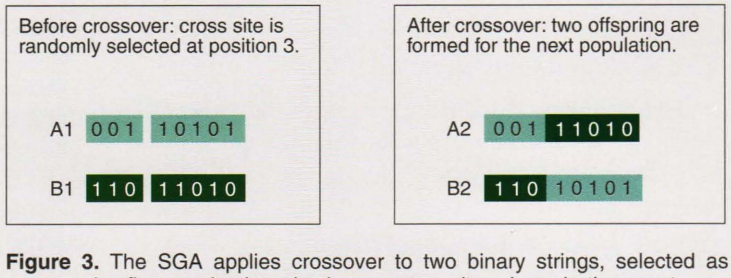


Figure 3. The SGA applies crossover to two binary strings, selected as parents, by first randomly selecting a cross site where both parents are separated, forming four substrings. The SGA then proceeds to form two new offspring by interchanging the genetic information represented within each parent's substring.

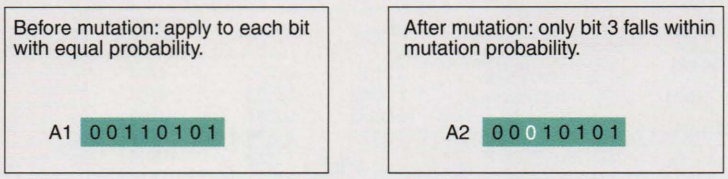


Figure 4. The SGA applies mutation to a binary string on a bit-by-bit basis. If the bit falls within the probability of mutation, its value is inverted.

demonstrates mutation. Mutation of a bit will occur only if a randomly generated number is greater than the probability of mutation p_m, where $0 \leq p_m \leq 1$.

Optimal selection of p_m and p_c remains an open issue, but two distinct parametric sets have emerged. One set has a small population size (30) with relatively large crossover ($p_c = 0.9$) and mutation ($p_m = 0.01$) probabilities. The other set has a large population size (100) with smaller crossover ($p_c = 0.6$) and mutation ($p_m = 0.001$) probabilities.[2]

Population Replacement Scheme

The process of selection and genetic operation continues until enough offspring are generated to replace the parent population. The SGA traditionally uses a nonoverlapping replacement scheme, where the entire population is replaced each generation. Many subsequent genetic algorithms apply overlapping techniques where only a portion of the population is selectively replaced. It is possible, for example, to keep one or more population members for several generations, as long as those individuals sustain a better fitness than the rest of the population. In addition, since maintaining the proper population size is critical to optimal convergence behavior, other nontraditional methods adaptively vary the population size to give the best convergence. GLOS applies some of these useful strategies by using an elitist principle, where one or more copies of the best solution of the parent population are always placed in the offspring population. GLOS also computes the optimal population size and maintains this constant size throughout the processing.

The SGA and Optimization

The following discussion focuses on why the SGA works and what to expect for convergence performance on optimization problems.

Optimal versus Near-Optimal Performance

Goldberg[3] notes that, when judging optimization procedures, the common focus is solely on convergence (i.e., does the method reach an optimum). Interim performance is entirely forgotten. This emphasis, stemming from calculus, is not the motivation of natural systems. For more humanlike optimization tools, the goal is improvement, that is, getting to some good, "satisfying" level of performance quickly.

Genetic algorithms do sort out interesting areas of a complex search space quickly, but they are weak and do not provide the guarantees of more convergent procedures. Thus, they can converge to near-optimal or quasi-optimal results. However, more convergent procedures are typically applicable to only a narrow class of problems. Genetic algorithms are extremely global in application and can be used where more convergent methods dare not tread. To help offset premature convergence, GLOS maintains the optimal population size and also employs fitness scaling to help prevent the dominance of false peaks within the population.

Why Does the SGA Work So Well?

Holland[5] captures the essence of SGA mechanics with the schema theory. The SGA takes advantage of parallel search by asking "In what way are the fittest binary strings similar?" To answer this question, the idea of the schema is proposed. A schema (plural schemata) is simply a similarity template describing a subset of strings with similarities at certain positions. The SGA defines a schema over the alphabet (0, 1, *), where * matches either a 0 or 1. The number of fixed positions of the schema is its order, and the distance between the outermost fixed positions of the schema is the defining length. For example, the schema *111* describes a subset with four members (01110, 01111, 11110, 11111) and has an order of 3 and a defining length of 2. The schema also has an associated fitness computed by taking the average fitness of the binary strings that it represents.

The concept of schemata is important because the SGA's search for optimal strings is a simultaneous competition among schemata to increase their instances within the population. The optimal string is viewed as a juxtaposition of schemata with short defining length and high fitness values. Such schemata are appropriately called building blocks. The notion that strings with high fitness values can be located by sampling building blocks with high fitness values and combining the building blocks effectively is called the building block hypothesis.

SYSTEMS ARCHITECTURE

As shown in Fig. 5, the systems architecture of GLOS has three main components: the User Interface Module, the Learning Module, and the Process Parsing Module. The primary interaction the user has with

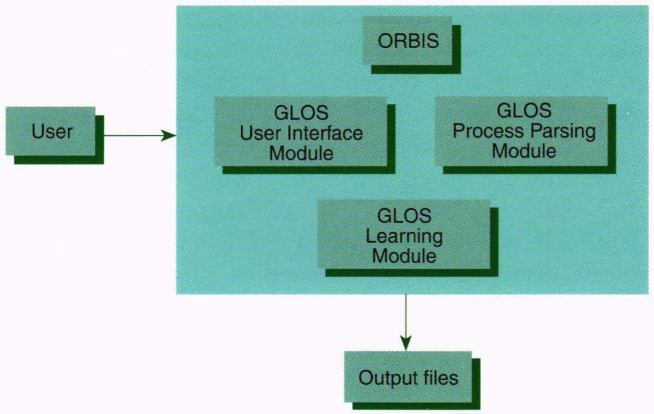


Figure 5. GLOS systems architecture. The User Interface Module provides a friendly environment for setting up an ORBIS simulation for optimization by the Learning Module. The Process Parsing Module provides a speedy execution of each processing cycle by evaluating each population in parallel, rather than sequentially.

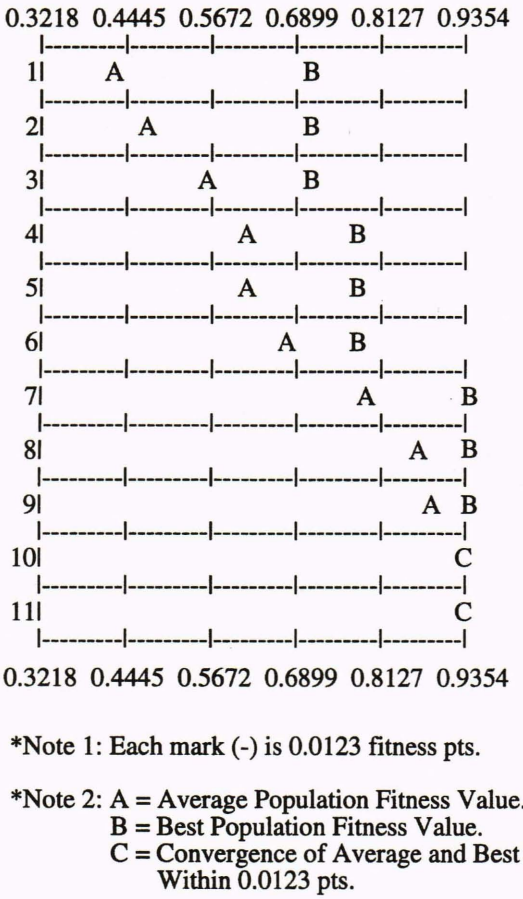


Figure 6. The GLOS learning graph file is used to monitor convergence on a best solution, which occurs when the population's average fitness differs from the best solution's fitness by a predefined threshold. GLOS terminates processing at convergence.

GLOS is the User Interface Module, which allows the user to create the input files that GLOS requires and to start or terminate a genetic learning session. It has been integrated into the existing ORBIS "background mode" menu system. The Process Parsing Module is the key to execution speedup for GLOS. The traditional SGA sequentially processes each potential solution within a population. GLOS, on the other hand, distributes solution evaluation over several central processing units so that subsets of the current population are processed simultaneously in parallel. When all solutions in a population have been evaluated, control is passed back to the Learning Module. The SGA is implemented entirely within this module. Its major tasks are to apply genetic operators to the current population, compute vital statistics related to the processing of the current population, and generate a population for the next processing cycle. The Learning Module also interacts with the Process Parsing Module to start evaluation of solutions of the next population and produces the output files for each generation.

OUTPUT FILES

Each generation, GLOS creates four output files that summarize the genetic learning results from the start of the session to the generation currently being processed. The learning graph file presents graphically the best-solution fitness and the average population fitness versus the generation number in which each occurred. As shown in Fig. 6, the file gives a high-level view of the learning progress of the session. On a generation-by-generation basis, the user can monitor the discovery of solutions of better fitnesses, leading up to a convergence of the best-solution and average fitnesses.

The best-solutions file (Fig. 7) lists two types of information: the best solution held for the current generation and the best solutions of previous generations when those solutions were held for more than one generation. Basically, the file is used to see how GLOS is converging on a single best solution or a set of best solutions. A set of best solutions can occur when the parametric search space is multimodal. A current best solution held for at least 10 generations is a good indication of convergence.

The population file contains a complete listing of the population held for the last generation processed. This file lists the number of occurrences or copies for each member maintained in the current population. For each member the minimum, maximum, and average fitnesses are also maintained. A population is considered converged when there are many copies of a few members in the current population.

The last file created by GLOS is the checkpoint file, which is used to save the last population processed. It can be used to restart a future learning session from the point that the current session ended.

```
==========================================
Current Best Solution
------------------------------------------
Generations Held: 5
Fitness: 0.9354
------------------------------------------
+++ COURSE  90.0 DEGREES
------------------------------------------
+++ SPEED 10.0 KNOTS
------------------------------------------
+++ DEPTH  135.0 FEET
==========================================

==========================================
Previous Best Solution: 1
------------------------------------------
Generations Held: 3
Fitness: 0.7759
------------------------------------------
+++ COURSE 45.0 DEGREES
------------------------------------------
+++ SPEED 8.0 KNOTS
------------------------------------------
+++ DEPTH 200.0 FEET
==========================================

==========================================
Previous Best Solution: 2
------------------------------------------
Generations Held: 3
Fitness: 0.7021
------------------------------------------
+++ COURSE 35.0 DEGREES
------------------------------------------
+++ SPEED 9.0 KNOTS
------------------------------------------
+++ DEPTH 300.0 FEET
==========================================


==========================================
========== END OF LISTING ==========
==========================================
```

Figure 7. The GLOS best-solutions file is used to monitor the parametric set that gives the best solution and the parametric sets that were held as previous best solutions. GLOS also keeps track of the number of cycles that each parametric set was held as the best solution.

EVALUATING GLOS PERFORMANCE

ORBIS was used by the cost and operational effectiveness analysis study team organized for the Naval Submarine Offboard Mine Search System (SOMSS) program.[10] At a mission assessment level, ORBIS provided support for the Milestone 1 decision for the Naval Acquisition Review Board. One cost factor involved in using ORBIS for the SOMSS assessment was the sensitivity analysis of parameters associated with the simulation model for planning safe paths through a mined region.

Several background mode trials were needed to locate optimal settings for parametric values associated with the path-planning model of the simulation. Consequently, additional cost was incurred by requiring a tactician to supply the intuition necessary to adjust those parameters. Therefore, this path-planning model was selected as a test-bed simulation to evaluate GLOS performance. Our primary goal was to test the skill of GLOS in selecting parametric settings that were "tweaked" by the tactician, that is, to determine if GLOS could offer the same intuition needed to replicate the tactician's results.

Selecting the Model Parameters to Learn

The SOMSS simulation involves an unmanned underwater vehicle (UUV), acting as an escort vehicle, that plots a safe path for a nuclear-powered attack submarine (SSN-688) to follow as they transit in tandem through a mined area. Figure 8 shows the general desired behavior. The path-planning model allows the UUV to plan a safe path around all detections encountered without violating a safe area defined around the mine. The model itself is composed of seven main parameters, listed in Table 1, which must be adjusted for a particular minefield pattern to produce high-fidelity behavior for the UUV.

Setting Up the Test Case Minefield

To evaluate GLOS skill to optimize over this parametric set, we selected a test case minefield identical to those used in the SOMSS simulation analysis. As shown in Fig. 9, the SSN-688/UUV transit was to begin randomly on a starting line and terminate randomly on an ending line. Two minefields were used—a line of mines with fixed positions interposed on a field of randomly placed false contacts called nonmine bottom objects or NOMBOS. To replicate the training patterns used by the tactician, each member of a GLOS population was trained with a simulation having a

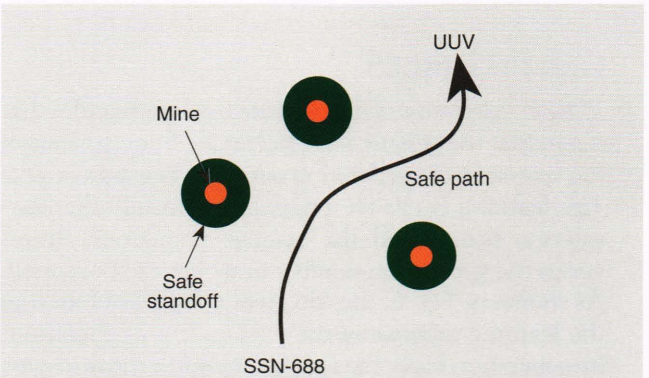


Figure 8. The GLOS simulation test bed is a path-planning model governing the motion of an unmanned underwater vehicle (UUV) to plan a path that keeps itself and an SSN-688 safe as they transit a minefield in tandem.

Table 1. Selected parametric set.

Parameter	Determines
Mine safe distance	How close to the mine any point on the path can be
Turn diameter	How tight the turns within the path should be
Maximum turn angle	What the maximum turn allowed within a path is
Test point distance	How many points along the path are needed
Exit count	How many attempts to find a path are made
Arc step	How much curvature within the path is needed
Safe backtrace distance	How far to back away from a mine to plan safely

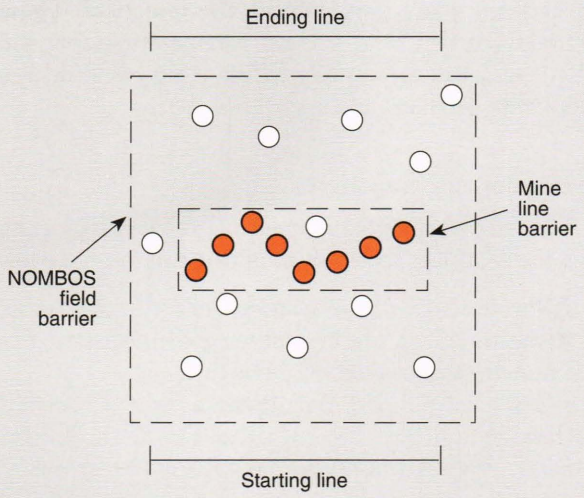


Figure 9. The test case minefield for the first evaluation is a fixed line of mines (closed circles) imposed on a randomly generated field of nonmine bottom objects (NOMBOS, open circles).

different starting location, ending location, and configuration of NOMBOS, with the same fixed line of mines interposed on it. This configuration forces GLOS to readjust parameters so that the UUV navigates successfully in the presence of a fixed line of mines, where the pattern or geometry of the NOMBOS field below the mine line always varies.

Performance Evaluation

GLOS must initially know the search space as well as the learning goal. Table 2 defines the search space by specifying the allowable values for each parameter. To compound the difficulty for GLOS we selected reasonable ranges for each parameter in such quantities that the search space became combinatorially explosive. Notice that this search space has 65,536 possible solutions calculated by taking the product of the number of possible values for each parameter of the parametric set.

GLOS next needs to know the user's learning goal, which allows it to distinguish between good and bad decisions. For our test, a learning goal similar to the one used for the SOMSS analysis was used. The learning goal is threefold:

1. Allow the SSN-688 to spend as little time as possible in the minefield. The best metric for this is to keep the simulation time as short as possible.
2. Allow the SSN-688 to maintain at least a 450-yd standoff from any detected mine or NOMBOS detected. The metric for this is the closest point of approach to any detection.
3. When the UUV detects enough mines within the mine line and determines that a line of mines indeed exists, it should avoid or skirt the mine line while maintaining at least 450 yd from both corner mines. The metric for this is the closest point of approach to any corner mine.

Results of the GLOS Skill Test

GLOS was run for four generations, maintaining a constant population size of 17 potential solutions for each generation, with $p_c = 0.9$ and $p_m = 0.01$. The

Table 2. Search space.

Parameter	Value
Mine safe distance (yd)	600 to 1300 × 100
Turn diameter (yd)	200 to 500 × 100
Maximum turn angle (deg)	30 to 45 × 5
Test point distance (yd)	100 to 250 × 50
Exit count	600 to 1200 × 200
Arc step (deg)	30 to 45 × 5
Safe backtrace distance (yd)	600 to 1300 × 100

parametric settings favored by GLOS are shown in Table 3 along with those of the SOMSS tactician. These settings are similar, but how did we compare in terms of simulation performance? Table 4 shows the average test case performance over 100 simulation runs using closest points of approach and simulation times as metrics of evaluation. The benefit seen here is that GLOS has the power to learn very quickly, needing only to consider about 0.1% of the solution space. GLOS is cost-effective: it can reproduce comparable performance in a learning session lasting only a few hours rather than the many weeks needed by the SOMSS tactician using a trial and error approach. Figures 10 and 11 show examples of paths planned by both approaches.

WHY USE GLOS?

We present here a simple scenario to demonstrate how GLOS can be used efficiently. Suppose that the sponsor of the SOMSS cost and effectiveness analysis requests that ORBIS be extended to prove by concept that the SSN-688 with UUV escort could be used to penetrate a densely packed mined area, not a fixed mine line against false bottom detections. The sponsor needs a rapidly prototyped simulation, but we cannot afford a tactician to train the path-planning model, there is not enough time to tweak the model parameters for high-fidelity behavior, and a tactician is not available. The sponsor has two alternatives: (1) extend the use of the parametric set obtained from the previous study or (2) use GLOS to recommend some good values that give us a safe transit within a quick simulation time.

Table 3. Recommended parametric settings for GLOS skill test.

Parameter	SOMSS tactician	GLOS
Mine safe distance (yd)	1300	1300
Turn diameter (yd)	400	400
Maximum turn angle (deg)	45	45
Test point distance (yd)	200	150
Exit count	850	800
Arc step (deg)	45	45
Safe backtrace distance (yd)	900	800

Setting Up the Test Case Minefield

Figure 12 shows the densely packed test case minefield that the SSN-688 with UUV escort is to penetrate safely. The SSN-688 transit will begin randomly on a starting line and will terminate randomly on an ending line. Mines are randomly placed within the minefield barrier. Each member of a GLOS population will be trained with a simulation that has a different starting and ending location as well as a different configuration of randomly place mines within the minefield. These factors force GLOS to readjust parameters so that the UUV navigates successfully within a minefield, whose pattern or geometry always varies.

Performance Evaluation

The same search space defined in Table 2 is used again, and the learning goals are modified slightly to read:

1. Allow the SSN-688 to spend as little time as possible in the minefield. The best metric for this is to keep the simulation time as short as possible.
2. Allow the SSN-688 to maintain at least a 500-yard standoff from any detected mine. The metric for this is the closest point of approach to any detection.

Results of the GLOS Simulation Speed-Up Test

GLOS was run for 14 generations, maintaining a constant population size of 17 potential solutions for each generation, with $p_c = 0.9$ and $p_m = 0.01$. The parametric settings favored by GLOS are shown in Table 5 along with those of the SOMSS tactician repeated from Table 3. Test case performance was again taken over 100 simulation runs using closest points of approach and simulation times as metrics.

Table 6 shows performance results obtained by GLOS for the last learning session. The benefit of using GLOS as an alternative to just reusing the old

Table 4. Simulation performance for GLOS skill test.

Learning goal	SOMSS tactician	GLOS
Percentage of runs for which SSN-688 maintained a 450-yd standoff from a corner mine and a detected mine	75	76
Average simulation time (h)	2:32:07	2:26:08

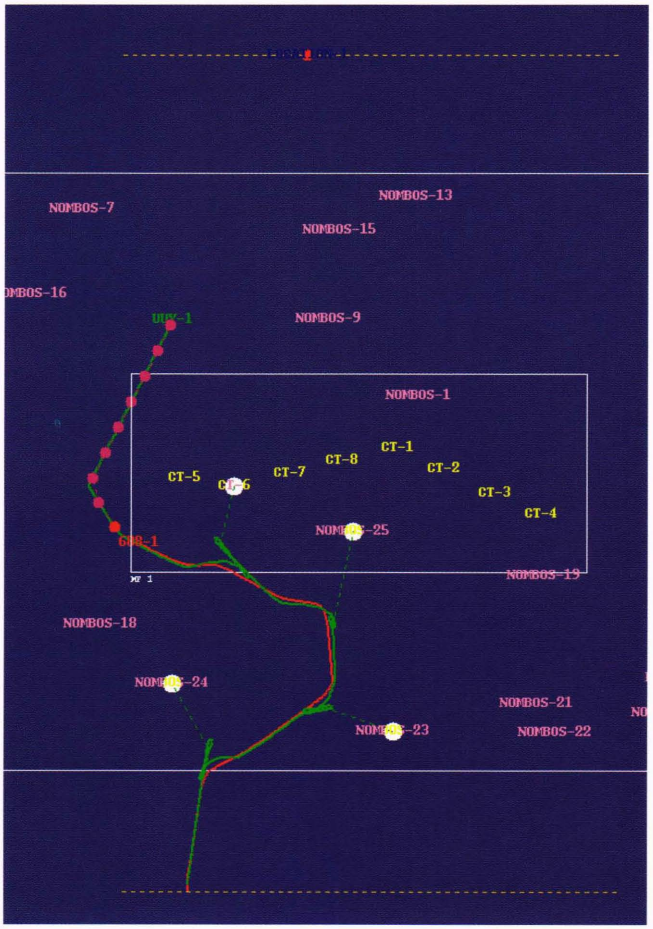


Figure 10. Simulation behavior using SOMSS tactician parameters shows the UUV operating with desired motion as it plans a safe path to keep itself and its host vehicle (the SSN-688) from coming too close to any mines (denoted as CTs) and nonmine bottom objects (NOMBOS). The red and magenta dots along the safe path following the UUV represent points that the SSN-688 follows to ensure that it is safely following the path.

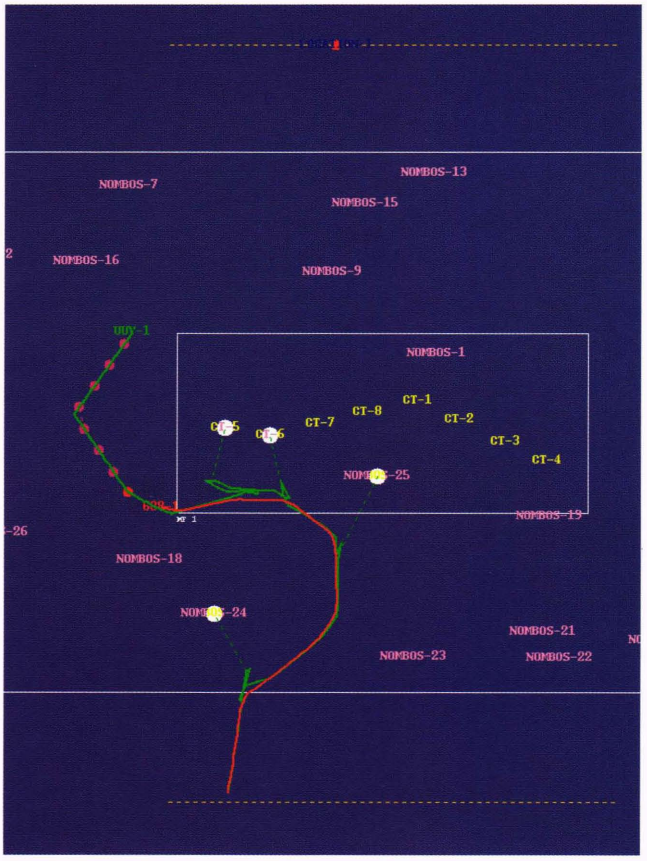


Figure 11. GLOS was used to recommend parametric settings to allow the UUV to operate in a motion similar to that observed for the SOMSS tactician. The simulation was run using the identical minefield configurations used by the tactician. The behavior observed from the GLOS-recommended parameters resulted in a similar desirable motion, allowing the UUV to plan a safe path around all mines (denoted as CTs) and nonmine bottom objects (NOMBOS). Again, the red and magenta dots along the safe path following the UUV represent points that the SSN-688 follows to ensure that it is safely following the path.

Table 5. Recommended parametric settings for GLOS simulation speed-up test.

Parameter	SOMSS tactician	GLOS
Mine safe distance (yd)	1300	800
Turn diameter (yd)	400	400
Maximum turn angle (deg)	45	45
Test point distance (yd)	200	100
Exit count	850	600
Arc step (deg)	45	30
Safe backtrace distance (yd)	900	800

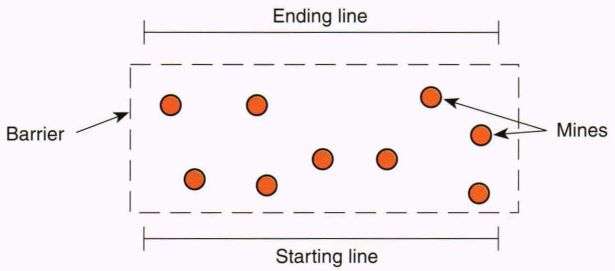


Figure 12. The test case minefield for the second evaluation is a configuration of mines randomly placed within a barrier. The UUV will begin at a randomly selected starting location, then plan a safe path to a randomly selected ending location.

parameters can be seen. The cost of using GLOS is about 8 h in computer resource time (which was actually run overnight after everyone had gone home). Even if a tactician had been available, the development cost would probably have exceeded 8 h for an expert to be able to adjust the parameters of this extremely large search space. Figures 13 and 14 show penetration paths that were generated by simply reusing

Table 6. Simulation performance for GLOS simulation speed-up test.

Learning goal	SOMSS tactician	GLOS
Percentage of runs for which SSN-688 maintained a 500-yd standoff from a detected mine	75	84
Average simulation time (h)	3:18:44	2:23:43

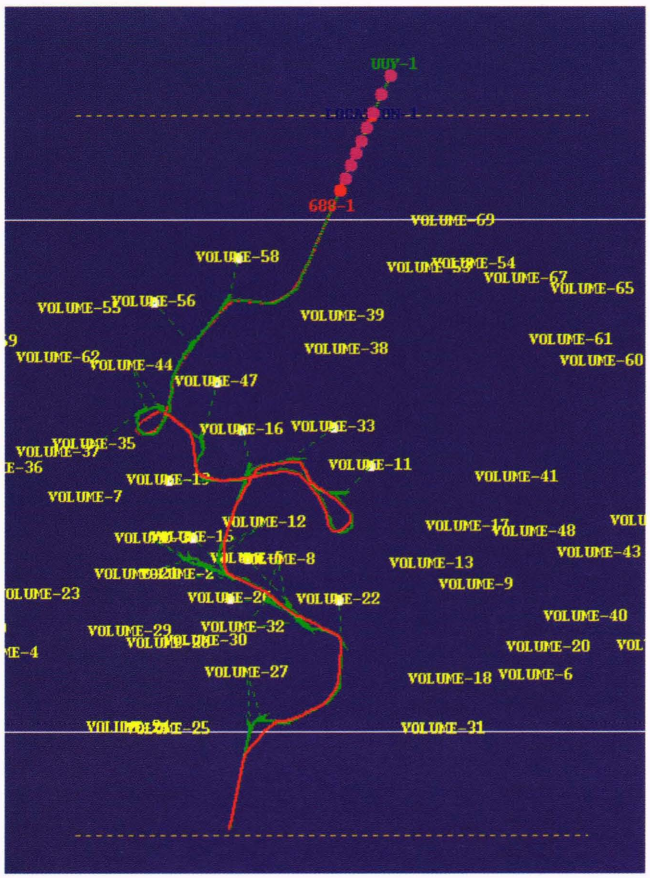


Figure 13. Simulation behavior observed when the original SOMSS tactician's parameters were reused. Notice that the UUV does plan a safe path around the encountered mines (denoted as VOLUME). Could GLOS be used to recommend some good parametric settings to get better performance?

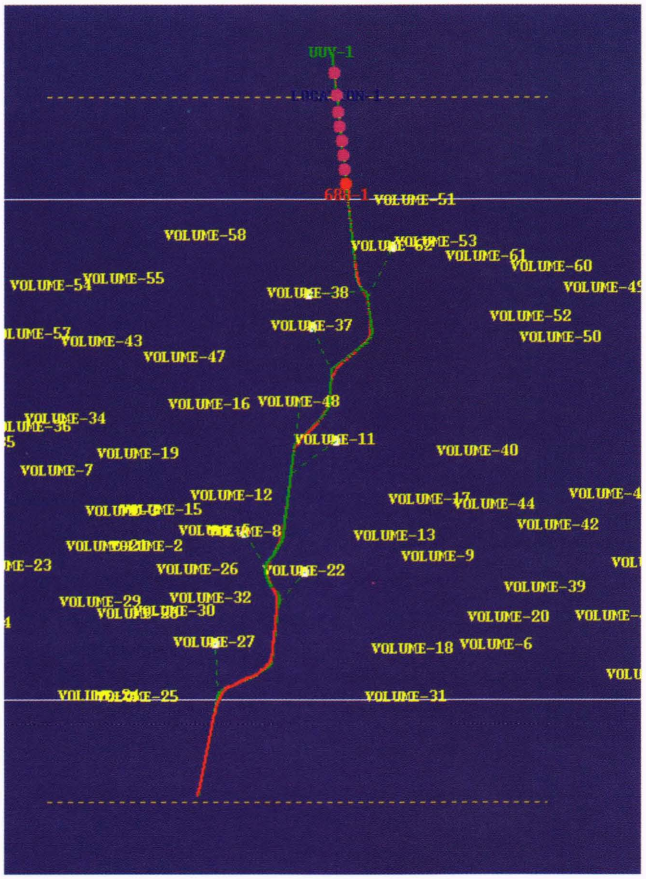


Figure 14. GLOS was run over an 8-h learning session to recommend some good parametric settings. The behavior observed from using these GLOS parameters resulted in a more desirable motion and allowed the UUV to plan a safe path around all mines (denoted as VOLUME).

the old tactical parametric set and by using GLOS, respectively.

CONCLUSION

We do not suggest in this article that artificial intelligence tools such as GLOS should replace the expert tactician; rather, GLOS can be used to augment the expert's resources. That is, an expert's time can be more efficiently allocated to perform more demanding tasks, while a tool such as GLOS can be applied in less critical areas where it simply is not cost-effective to use the expert.

GLOS has much to offer the user in terms of ORBIS simulation development and speedup. It gives the user a tool to quickly locate areas within a complex, arbitrary, multiparametric search space that can provide high-fidelity simulation performance. The authors hope future users find it helpful.

REFERENCES

[1] Evans, R. B., and Sanders, R. D., "ORBIS: A Tool for Simulation Development," in *Proc. Summer Simulation Conf.*, San Diego, CA (Jul 1994).

[2] Srinivas, M., and Patnaik, L. M., "Genetic Algorithms: A Survey," *Computer* **27**(6), 17–26 (1994).

[3] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA (1989).

[4] Davis, L., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York (1991).

[5] Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor (1975).

[6] Goldberg, D. E., *Optimal Initial Population Size for Binary-Coded Genetic Algorithms*, Report 85001, The Clearinghouse for Genetic Algorithms, University of Alabama (Nov 1985).

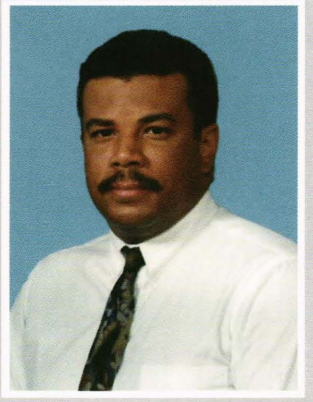
[7] Cavicchio, D. J., *Adaptive Search Using Simulated Evolution*, Ph.D. Thesis, University of Michigan, Ann Arbor (1970).

[8] Frantz, D. R., *Non-Linearities in Genetic Adaptive Search*, Ph.D. Thesis, University of Michigan, Ann Arbor (1972).

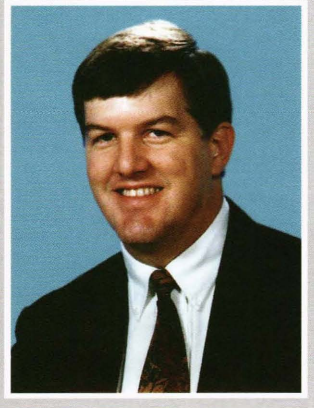
[9] DeJong, K. A., *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Thesis, University of Michigan, Ann Arbor (1975).

[10] Benedict, J. R., "Countering Mines in Littorals—Operational and Technical Implications for U.S. Submarines," in *Proc. Submarine Technology Symp. at JHU/APL*, Laurel, MD (1994).

THE AUTHORS



LLOYD A. BEST received B.S. and M.S. degrees in electrical engineering from North Carolina Agricultural & Technical State University in 1988 and 1991, respectively. He is a software engineer in the Advanced Combat Information Technologies Group of APL's Submarine Technology Department and a member of the Associate Professional Staff. Since joining APL in 1991, Mr. Best has worked on FBI fingerprint recognition, rocket plume simulation, verification and validation techniques for expert systems, object-oriented computer simulation development, and machine learning applications involving genetic algorithms. His e-mail address is Lloyd.Best@jhuapl.edu.



ROBERT D. SANDERS received a B.S. degree from Tulane University in 1982 and an M.S. degree from The Johns Hopkins University G.W.C. Whiting School of Engineering in 1992, both in computer science. From 1982 to 1987, he served in the Navy's nuclear submarine community, leaving as a lieutenant. Mr. Sanders then worked at the David Taylor Research Center, where he developed and verified thresholds for the Trident submarine monitoring subsystem. In 1991, he joined APL's Advanced Combat Information Technologies Group and is currently involved in a variety of ORBIS-based simulation efforts. He is a member of the Senior Professional Staff and specializes in computer simulations, tactics analysis, and technology assessments. His e-mail address is Robert.Sanders@jhuapl.edu.