

Using Genetic Algorithms to Aid in a Vulnerability Analysis of National Missile Defense Simulation Software

Eric S. Imsand

Computer Science & Software Engineering
Auburn University
imsanes@auburn.edu

Gordon Evans

Booz | Allen | Hamilton
Missile Defense Agency
Gordon.Evans.ctr@mda.mil

Gerry Dozier

Computer Science & Software Engineering
Auburn University
gvdozier@eng.auburn.edu

J.A. Hamilton, Jr.

Computer Science & Software Engineering
Auburn University
hamilton@eng.auburn.edu

The National Strategy for Homeland Defense, published by the then U.S. Office of Homeland Security in July 2002, directs all U.S. government agencies to conduct vulnerability analyses of their sensitive systems. This policy applies to Department of Defense systems, including the simulation packages used to design and exercise national missile defense.

Many, if not most, of the previous vulnerability analyses of missile defense simulation platforms have utilized traditional reverse engineering techniques along with a review of all documentation and publicly available sources. These experiments have produced some useful information, though the amount of platform specific data recovered has been limited.

The use of genetic algorithms (GAs) has been shown to be an effective method of performing boundary analysis and parameter optimization. In this paper, we show how GAs can be used to extract information concerning how particular parameters affect heavily parameterized missile defense simulation system performance. This information would be very valuable to researchers as part of a greater vulnerability analysis of national missile defense software.

Keywords: Genetic algorithms, simulation, missile defense, vulnerability analysis

1. Background

In July of 2002, the then U.S. Office of Homeland Security published the National Strategy for Homeland Defense. Part of this strategy required all governmental agencies to conduct vulnerability analyses of all critical systems. The goals of these analyses were to discover any vulnerability that might allow a hostile organization to not only sabotage sensitive systems, but also compromise

them for the purpose of information extraction.

To comply with the newly issued strategy, the Missile Defense Agency ordered that a vulnerability analysis be conducted of their missile defense simulation packages. These simulations were used to design and develop modern defense strategies for the United States. To develop modern strategies, these missile defense simulation packages were required to simulate highly sensitive military hardware. To enhance national security, the developers of these simulation packages made them highly parameterized, meaning that no performance data was stored in the software packages

themselves. Any performance information needed for the simulation of a system had to be provided by the end user. However, it was feared that while the simulation packages did not contain any performance data, the packages might contain some inherent information necessary to perform the simulations, such as behavioral characteristics. Any information of this type that could be gained from the simulation packages still could be considered a vulnerability.

2. Previous Research

Analysis of national missile defense simulation software is a relatively new endeavor. Vulnerability analysis, though, is not a new field. There even has been considerable work performed using genetic algorithms (GAs) as part of vulnerability analyses.

Many, if not most, of the previously published studies regarding vulnerability analysis of national missile defense simulation packages have utilized traditional vulnerability analysis techniques. These techniques have included traditional reverse engineering practices [1] and documentation review [2]. These studies also made use of publicly available information as part of the vulnerability analysis [2]. The amount of information recovered from these experiments was considerable [1]. This large amount of data resulted in a desire to determine which pieces of data were more important to the execution of the simulation models.

Vulnerability analysis research has been performed using GAs; the research domains vary considerably. For example, GAs have been used to perform vulnerability analysis of resource constrained tactical networks [3] and have also been used to conduct vulnerability analysis of tactical battlefield environments [4].

Though GAs have not been used to find vulnerabilities in missile defense simulation software, they have been utilized in the analysis of other types of simulation packages. Researchers at the Naval Research Laboratory have had success using GAs to perform vulnerability experiments on intelligent controllers for autonomous vehicles [5]. It should be noted that the term “vulnerability” carries a different meaning in the context of the Naval Research Laboratory experiments. The goal of those experiments was to utilize a GA in partnership with a vehicle simulator to discover sets of data that would cause the intelligent controller to fail. This differs with the meaning of vulnerability in the context of this experiment. In this case, the term vulnerability is used to describe any potentially sensitive information that may potentially be extracted from the missile defense simulation packages.

3. Evolutionary Computation

Evolutionary Computation (EC) [6,7,8,9] is the field of study devoted to the design, development, and analysis of problem solvers based on simulated evolution. Evolutionary computations (ECs) are biologically-inspired systems that have been successfully used to solve a wide range of problems in the areas of robotics [10], constrained optimization [9], machine learning [8], network and computer security [11], and vulnerability analysis [5], just to name a few. ECs are different from most traditional problem solvers in that they operate on a population of candidate solutions (CSs) rather than on just one CS. They have been particularly successful in solving large complex problems for which traditional problems solvers yield unsatisfactory results.

Figure 1 provides a pseudo-code example of an EC. Initially, a population of CSs (also referred to as individuals) is randomly generated and evaluated using a user-specified evaluation function. The evaluation function assigns each individual a fitness that is representative of its “goodness.” After the initial population has been created and evaluated, the EC begins to iteratively refine the population by: 1) selecting parents based on their fitness, 2) allowing the selected parents to create offspring through crossover (sexual reproduction) and/or mutation (asexual reproduction), 3) evaluating the offspring, and 4) determining which individuals of the current population and the set of offspring get to survive to the next generation.

3.1 Genetic Algorithms

Most ECs fall into one of three categories [7]: genetic algorithms [8,9,12,13], evolutionary algorithms (EAs) [6,14] and Swarms [5]. GAs primarily rely on sexual reproduction (called recombination or crossover) to recombine CSs selected to be parents while EAs primarily rely on mutation. Swarms are EC techniques

```

Procedure EC()
{
  t = 0;
  Initialize Pop(t);
  Evaluate Pop(t);
  While (Not Done)
  {
    Parents(t) = Select_Parents(Pop(t));
    Offspring(t) = Procreate(Parents(t));
    Evaluate(Offspring(t));
    Pop(t+1) = Replace(Pop(t), Offspring(t));
    t = t + 1;
  }
}

```

Figure 1. The evolutionary process of an EC

based on social evolution. However, to date, GAs have been the most popular and widely used EC paradigm. GAs also have a long history of success on a variety of optimization problems [16]. More recently, they have been applied to a number of other problems including pattern recognition, robotics, expert systems, and biological applications [17].

When developing a GA for a given application, after a suitable evaluation function has been developed, thought must be given to the following issues: 1) the population size, 2) the representation of candidate solutions, 3) how to select individuals to be parents, 4) the type of procreation methods that will be used to allow the parents to create offspring, and 5) which individuals of the current population will be replaced with the newly created offspring.

3.2 Parent Selection in GAs

The number of individuals within the population of a GA usually varies based on the problem at hand. Once the user has specified the population size it remains static throughout the evolutionary process. Regardless of the application, the size of a population must be chosen carefully. Choosing a population size that is too large may result in far more function evaluations than needed to converge upon a solution. Choosing a population size that is too small may cause the GA to prematurely converge on a population of undesirable candidate solutions [7].

3.3 The Representation of Candidate Solutions

GAs typically are divided up into two broad categories based on the way individuals are represented. In binary-coded GAs [7,8], candidate solutions are represented as binary strings. Thus, binary-coded GAs require a mapping function that is used to map the binary string (known as the genotype representation) into the corresponding candidate solution (known as the phenotype representation) that the binary string represents. Real-coded GAs, on the other hand, operate exclusively on the candidate solution (phenotype representation). Therefore, no mapping function is needed. Although much of the theoretical work on GAs has been done using binary-coded GAs [8], most real-world applications of GAs use real-coded representations [2,9].

3.4 Parental Selection in GAs

The way in which parents are selected can dramatically affect the search behavior of a GA. If too much emphasis is placed on selecting individuals that have the best fitness, then the population will rapidly converge to a

sub-optimal solution. However, too little emphasis on the better performing individuals of a population will result in slow convergence (and sometimes non-convergence). Thus, it is important for the parent selection method to balance selection pressure (the tendency to select the better performing individuals) with diversity (the tendency to select less fit individuals as parents in hopes that they possess genetic material critical for evolving optimal or near-optimal solutions) [7,9].

There are a number of popular selection methods used by GA researchers [7,8,9,18]. The first method is known as Proportionate Selection [7,18]. Proportionate Selection assigns a probability of a particular individual being chosen to procreate based on its fitness. This probability can be represented mathematically using the formula $p_i = f_i / \sum f_j$, where p_i is the probability that the individual will be chosen, f_i is the fitness of the individual, and $\sum f_j$ is the summed fitness values of all individuals in the population.

Another frequently used method of selecting parents is known as Linear Rank Selection [7,12,18]. Linear Rank Selection selects parents based on a subjective fitness. This subjective fitness is determined by the individual's relative fitness when compared to other members of the population. The most fit individual in the population would receive a ranking of one; the second most fit individual would receive a ranking of two, etc. The subjective fitness is frequently computed using the formula $sf_i = (P-r_i)(max-min)/(P-1) + min$, where sf_i is equal to the individual's subjective fitness, P is the size of the population (number of individuals in the population), r_i is the individual's rank within the population, max is the subjective fitness to be assigned to the best individual, and min is the subjective fitness to be assigned to the worst individual.

Perhaps the most widely used selection method is Tournament Selection [9]. In k-Tournament Selection, k individuals are randomly picked from the population and the best of these k individuals is selected to be a parent. This process is repeated to select a second parent if needed. Binary tournament selection (k=2) is the most widely used selection method.

3.5 Genetic Operators

The operators used to create offspring typically fall into one of two categories: recombination and mutation [8,9]. In this section, we will briefly introduce three popular recombination (crossover) operators and Gaussian mutation. The recombination operators that we present typically are used for real-coded chromosomes (see Section 3.3). These operators differ significantly from single-point crossover used in many GA applications¹. When using real-coded GAs, the three recombination operators to be presented perform better than single-point crossover [9].

3.5.1 Recombination Operators

A number of different recombination operators are used in the creation of offspring. Most of these operators are concerned with how the CS of each parent is combined to form an offspring. The most commonly used genetic recombination operators for real-coded GAs are mid-point crossover, flat crossover (also known as BLX-0.0 crossover), and BLX-0.5 crossover.

Mid-point crossover attempts to average the attributes of both parents and assign the average to the new offspring. For example, consider a real-coded GA that attempts to find a number whose square root is less than three. Suppose that two parents have been selected, p_1 and p_2 . Each parent contains two pieces of information — its fitness and its CS, x_p , whose square root potentially may be less than three. To create a new offspring, c , that has a CS, x_c , the CSs of each parent are summed and then divided by two. This would yield the average of the two values, and would be assigned to the newly created offspring's value of X . This calculation can be represented mathematically as $x_c = (x_1 + x_2) / 2$. In this example (and the ones to follow), we use CSs that are composed of only one variable. However, to apply this recombination operator (and those to follow) to CSs that are composed of a number of variables, each of the values assigned to the corresponding variables of the parents are crossed and the result is assigned to the corresponding variables of the offspring.

A second type of recombination operator typically used with real-coded GAs is known as flat crossover [19]. Flat crossover uses the values of the genes (variables) of the two parents and assigns a value to the corresponding genes of an offspring by randomly selecting a value that lies between the values of the genes of the parents. Using the example above, $x_c = U[x_1, x_2]$, where $x_1 \leq x_2$.

The third type of crossover operator frequently used in real-coded GAs is known as BLX-0.5 crossover [20]. BLX-0.5 crossover is similar to flat crossover since the values of both parents are used in the creation of the new offspring. However, BLX-0.5 crossover attempts to expand the potential values of the offspring by multiplying the difference of the two parents by a constant factor, in this case 0.5. This operation can be represented mathematically by the following formulas, where $x_1 \leq x_2$ and where $rnd(lb,ub)$ returns a random number within the interval $[lb..ub]$:

$$\text{Let } \Delta = \Delta(x_1 - x_2), \text{ where } \alpha = 0.5 \\ x_c = rnd(x_1 - \Delta, x_2 + \Delta).$$

3.5.2 Mutation

It is sometimes advantageous to use mutation on individuals in a population to replace the genetic diversity

that may be lost due to selection and recombination [13]. Mutation is used in genetic search as a way of preventing a phenomenon known as premature convergence [21]. Premature convergence typically results when there is a lack of genetic diversity within the population. This usually will cause the GA to converge upon a sub-optimal (undesirable) solution. In real-coded GAs, mutation typically comes by way of perturbing the values of an offspring's genes by adding a normally distributed random number with a mean of zero and a standard deviation of σ . This is also known as Gaussian mutation and is denoted as $N(0,\sigma)$ [9].

3.6 Replacement

After the offspring of a generation (or iteration) have been created and evaluated, a replacement method is used to decide which individuals should be allowed to survive and which individuals should die. There are several replacement methods of choosing the survivors of a generation [6,22]. Two of the more common methods are generational replacement [12] and steady-state replacement [23]. In generational replacement, μ offspring are created and the offspring replace the μ individuals of the population. In steady-state selection, λ (usually λ is equal to one or two) offspring are created every iteration and these offspring replace the worst λ individuals of the current population. [24,25]. This replacement occurs regardless of how good the offspring are relative to the individuals that they are replacing.

4. Commanders Analysis and Planning Simulation

The Commanders Analysis and Planning Simulation (CAPS) system is an operational missile defense simulation. The CAPS simulation system primarily is composed of two major components: the graphical user interface and the simulation engine. When examined at a macroscopic level, the graphical user interface provides an environment designed to allow the user to interpret information and initiate actions. The simulation engine is used to execute the simulation itself.

The CAPS simulation system belongs to an informal class of simulators that can be termed as being heavily parameterized. Members of this informal classification, such as CAPS and Extended Air Defense Simulation (EADSIM), execute simulations identically regardless of the sensitivity of the information provided as input [26]. The same simulation engine(s) and algorithms are used to simulate classified and unclassified scenarios. These scenarios can make use of classified or unclassified data without affecting the manner in which the simulation is executed.

To conduct a vulnerability analysis experiment using

CAPS, a more detailed examination of the simulation system must be performed. Areas examined particularly closely include the file(s) and file types used by CAPS, and how the simulation engine interacts with these files.

4.1 CAPS File Types

Several binary executable files provide the CAPS simulation system functionality. These executables serve as either the graphical user interface (GUI) for the CAPS simulation system, or as part of the simulation engine itself. The GUI for CAPS is provided by the binary executable *caps_gui.exe*. The simulation engine is composed of two different binary executables: *caps_f.exe* and *caps_s.exe*. The *caps_f.exe* executable is used to generate information concerning the operating area, defended area, and footprint. The *caps_s.exe* executable is utilized to perform risk generation and scenario generation [27].

According to CAPS terminology, each simulation is known as a scenario [27]. All of the data that is unique to a particular simulation is stored in this scenario file. This data includes locations for all tactical assets, terrain characteristics, boundaries for the areas to be defended, and battle management rules. The scenario file also contains information required by the simulation engine, such as the seed value for the random number generator and location of other input/output files.

Specific data regarding the performance of tactical systems is stored in other external files. For example, data regarding the performance of the Patriot Advanced Capability-3 (PAC-3) missile system is stored in an external file labeled "patriot.interceptor." These external files are separated into different directories depending on their function. These functions include interceptors, radars, lasers, aircraft, etc. In addition to being grouped into separate directories, the data files also carry different file extensions based on what type of system they represent. As already illustrated, interceptors are stored in files with ".interceptor" extension, while aircraft files carry an ".air" file extension.

In addition to the data contained in the input files for each tactical system, CAPS also makes use of "flyout tables" that also are stored in external files. These external files are used by the CAPS simulation engine while simulating the scenario and are capable of being generated by several different simulation platforms. For example, data generated by using the EADSIM simulation package are capable of being used to provide the flyout table values [26].

The input files passed to the CAPS simulation engine are designated with an ".in" file extension. These files are a composite of most of the data contained in the other input files, with exception of the files containing

the flyout tables. The input file includes the data pertaining to tactical system performance and all of the data contained in the scenario file. The flyout tables are not included in the input file, though. Instead, the input file contains a path to the appropriate file where the flyout information is stored.

The CAPS simulation engine produces a total of fifteen different types of output files. These files are all differentiated by the file extension assigned to them. For example, files with an ".out" file extension are designed to be a human readable form of all data points used in the simulation. Files ending in ".out.scn" are designed to be interpreted by the GUI for graphical representation. Files ending in ".c.out" containing color code information, again for processing by the GUI, while files ending in ".err" contain information concerning any errors that may have occurred during the simulation. Additional information concerning the various types of output files created by CAPS can be found by consulting the documentation.

4.2 CAPS Execution and Functionality

The Commander's Analysis and Planning Simulation (CAPS) is designed to support new system acquisition as well as operational planning. Individual System Capabilities can be modeled and evaluated against threats in user-defined scenarios. The performance of upper tier versus lower tier systems can be evaluated. Additionally, doctrine and its effect on successful engagements can be evaluated.

5. Experimental Setup and Implementation

As previously stated, the goal of this study was to show that a GA is capable of extracting information that could be used as part of a greater vulnerability analysis of national missile defense simulation software. It was originally hypothesized that a GA would be capable of solving for the values of each individual parameter used in a simulation package.

We selected the CAPS simulation system to be the target simulation system used in this study. The CAPS simulation system was chosen based on an assessment of the system's future use as well as guidance from our research sponsor. Compared to other simulations we evaluated, we found CAPS the most interesting because it was targeted to run on multiple platforms (Solaris, IRIX, and Windows). For the purposes of the GA study, CAPS' ability to input from multiple file formats made this the most straightforward simulation to evaluate. After the CAPS simulation system had been sufficiently reverse-engineered to interface with, there still was considerable work performed on the design and implementation of the GA. The most challenging part of the design was

constructing an evaluation function (for assigning a fitness to each individual of the GA population) that would not optimize parameter values, but instead maximize the amount of change each value caused on the simulation output. Scalability also was taken into account when designing the fitness evaluation function. The ability to conduct future research to determine which groups of two, three, four, ... , n parameters were most influential also was desired.

The GA used in this study was a real-coded GA, using a modified steady-state survivor selection algorithm, with a fixed population size of 40 individuals. Parental selection was performed using tournament selection, with each set of parents creating only a single offspring. A population size of 40 was chosen after a review of research concerning population sizes for similar GAs. Some research indicated that a population size of 50 was optimal [28], while other studies showed a population size of 30 to be preferred [28]. Averaging these two different population sizes yielded a population size of 40.

5.1 Simulation Setup

We first constructed a scenario inside of the CAPS simulation system using the standard graphical user interface provided by CAPS. When configuring the scenario, it was decided to use the third generation Patriot missile system, also known as PAC-3, as the defensive system. The offensive system selected was a generic medium range ballistic missile model provided by CAPS. Since the data concerning the defensive weapons systems were of far more interest to the study, it was decided that the use of a generic offensive weapon was acceptable. To minimize any potential impact that terrain configuration might have on the performance of either system, it was decided that the geographical location should be relatively level. Though the exact location of the offensive and defensive weapons system was unimportant, it was decided that the experiment should approximate a likely combat scenario. For this reason, the offensive weapons system was placed just north of the Iraq-Kuwait border, while the defensive weapons system was placed in the middle of Kuwait. The total distance between the two objects was approximately 67.72 miles or 108.98 km.

5.2 Genetic Algorithm Architecture

As previously stated, evolutionary computation used in this study was a real-coded GA using a modified steady-state survivor selection. In order to determine which parameter(s) had the greatest impact on the simulation, it was necessary for each individual in the population to store two pieces of information for every variable used

by the simulation. The first piece, a floating-point value, indicated how much mutation potentially could be applied to a particular simulation variable. The second variable, a Boolean value, indicated whether the pre-specified mutation should be applied.

The survivor selection algorithm employed by the GA was a modified steady-state algorithm. The original steady-state algorithm requires that the offspring replace the parents, regardless of their fitness relative to the parents. The modified steady-state algorithm used in this study purposefully allows the 40 best individuals to survive, regardless of whether they are newly created offspring (this is known as a $(\mu+\lambda)$ -GA). Tests showed this survivor selection method allowed population evolution to occur more rapidly with no noticeable tendency toward premature convergence. Further formal study on this survivor selection algorithm is planned.

The other critical attributes of the GA used in this study were conventional. As previously mentioned, a population size of 40 was chosen. Parents were chosen using a tournament selection [9] algorithm and the BLX-0.5 [20] procreation operator was used in the generation of offspring.

5.3 Fitness Evaluation

In order to use a GA for this type of study, a consistent mean of evaluating an individual's fitness had to be developed. Based on specific guidance from the research sponsor, we assumed that the designers of defensive weapons systems, such as the PAC-3 system, would desire to intercept an incoming threat as soon as possible. Using this assumption, it was decided that the best indication of an individual's fitness would be the amount of time required for the defensive system to disable the offensive system. For example, an individual that is capable of intercepting the incoming threat in 17.8 seconds would be considered to have greater fitness than an individual capable of interception in 20.2 seconds.

The CAPS simulation system produces several output files after executing a simulation. The file carrying an ".out" extension is particularly useful for the purpose of determining an individual's fitness. Files with an ".out" extension contain human readable output of each data point generated by the simulation engine. The data in this file was capable of being easily parsed and all pertinent data extracted.

After analysis of the ".out" output file, it was discovered that any time the defensive weapon initiated an engagement, the phrase "Shot X:" was recorded, where X represents a method of identifying which shot was being tracked. On the same line as the phrase "Shot X:" was data indicating what time the shot(s) had been launched, and the expected intercept time. The launch time could then be subtracted from the expected intercept

time to yield a value indicating how many seconds were required for interception. The following is a pseudo-code representation of the output file processing function:

```
while (not end-of-file(output_file)) do
  buffer = read_line(output_file);
  if ("Shot 1:" in buffer) do
    tokens = tokenize(buffer, " ");
    total_time = tokens[intercept_time]-
                tokens[launch_time];
    return total_time;
  end if;
end while;
```

The algorithm described provided the basis for the previous parameter optimization experiment. However, the goal of the experiment described here was to determine which parameters in the CAPS simulation system had the most impact on the simulated performance of the PAC-3 missile system. To accomplish this goal, it became necessary to devise a slightly different fitness function.

The new fitness function was required to know which CAPS variables had been modified, and how much modification had been applied to them. The solution to this problem was to devise a new fitness function, incorporating parts of the old one. The new fitness function first calculated the amount of time needed for a baseline set of parameters to cause the simulated PAC-3 missile to intercept the incoming threat. Next, the new fitness function applied any mutations called for by the instance of class *GeneLinkage* being evaluated. A pseudo-code version of the algorithm can be expressed as:

```
TimeBaseline = InterceptionTime (Baseline);
TimeModified = InterceptionTime (Modified);

if (numberOfAttributesModified <= K)
  divisor = 1;
else
  divisor = numberOfAttributesModified;

Fitness = |TimeModified - TimeBaseline | / divisor;
```

The value of K is a constant, equal to the number of attributes that are being solved for. In other words, if $K = 1$, then the algorithm will search for the most influential single parameter. If $K = 2$, then the algorithm searches for the two most influential parameters, and so on. Calculating the fitness in this method allows the GA to maximize the amount of difference between the two-parameter sets, while minimizing the number of attributes modified. The GA then can successfully solve for the parameter that has the most impact on the simulated performance of the PAC-3 missile system. Notice that the value, K , only provides a preference for those CSs that use K or fewer attributes; however, it is

still possible for the best solution to be one where greater than K attributes are considered. The value of K may also be set to the total number of attributes. This will remove the preference for considering a smaller number of attributes and would result in a final population containing the 40 best individuals discovered regardless of the number of attributes an individual considers.

5.4 Random Number Generation

The GA used in this study makes heavy use of pseudo-randomly generated numbers. These numbers were generated using the C++ library function *rand()*. Pseudo-random number generator performance has been shown to have an affect both on the speed of a GA, and the quality of solutions generated [29]. For this reason, the pseudo-random number generator *rand()* included with Microsoft Visual C++ 6.0 was evaluated. This evaluation was conducted using the frequency test, Kolmogorov-Smirnov test, and gap test. The level of significance, α , was 0.05 for all tests performed.

The library function *rand()* generates pseudo-random numbers distributed uniformly. To confirm this, 10,000 pseudo-random numbers were generated and stored in an external file. The numbers were on the interval [0, 100). The data generated by *rand()* successfully passed all three tests for pseudo-random number generation.

5.5 Algorithm Execution

In order to determine how influential a simulation variable might be, some manner of comparing datasets must be employed. In order to make a comparison, a baseline value must first be obtained. This baseline is established by running the simulation using a default dataset and measuring its fitness using the fitness function developed for this study. This fitness value was used as a basis for comparison for the remainder of the experimental run.

After determining the baseline fitness to compare against, the population of candidate solutions was created. Each member of the population was initialized randomly. Generating a random number between 0 and 1 initialized the Boolean values. If the random number was less than 0.5, the Boolean variable was assigned a value of false. Otherwise it was assigned a value of true. The floating-point value used to apply mutation to each simulation variable typically was, though not always, within $\pm 10\%$ of the value used in the baseline dataset. To better understand this process, consider the CAPS variable "int_max_tof." This variable was assigned a value of 55.9885 in the baseline individual. The corresponding mutation amount was assigned a random value from the range $\pm 10\%$ of 55.9885.

After the population was created and randomly generated, it was evaluated using the modified fitness

function. This evaluation determined each individual's fitness based on the amount of change between it and the baseline fitness value, and the number of attributes that had been mutated.

After each member of the population had been evaluated, the population was randomly divided into two different tournaments. From these two tournaments, two individuals were chosen to create a new offspring. The new offspring then was placed into the population using the modified steady-state survivor selection. Stated differently, the new offspring replaced the worst individual in the population, regardless of whether the offspring was more fit than the individual being replaced.

For the sake of data collection, each individual of the population stored its data in an external file after having its fitness value set. This provided a way to track the population's evolution, since each individual created stored its data in an external file.

Because it was unclear as to the number of iterations the GA would require to find a solution, it was initially decided to allow the GA to run for a total of 10 hours. Five different runs of ten hours were performed, and the resulting data were examined. After examination of the data, it was discovered that the GA always derived its best solution prior to 800 function evaluations. Stated differently, the GA always found the best solution prior to the generation of the 800th offspring. After discovering this information, the GA was then executed 100 times, allowing each execution run to generate a maximum of 4,000 offspring. The number 4,000 was chosen for two primary reasons. First, and most importantly, the amount of time required for the GA to generate 4,000 offspring is very small, typically requiring about two minutes. Second, the maximum value of 4,000 offspring provided each execution of the GA far more time to find a solution than was indicated as being necessary by the initial five experimental runs.

6. Experimental Results

As previously stated, the GA was executed a total of 100 times in an attempt to discover the single most influential parameter. During each execution, the GA was allowed to create only 4,000 offspring. Initial experimental runs showed that the GA determined its solution by the creation of the 1,000th offspring. For this reason the number of offspring the GA was allowed to produce was limited to 4,000. This allowed the GA a reasonable amount of extra execution time if necessary. Of the 100 runs, 73 runs indicated that the single most influential parameter was the CAPS parameter *int_max_tof*. Though the CAPS documentation never directly describes what this parameter does, it is surmised that it indicates the maximum time of flight for the interceptor,

in this case the PAC-3 missile. Further more, another 23 experimental runs indicated that the parameter *int_max_tof_flag* was the single most influential parameter. Considering the apparent naming convention used by the developers in CAPS when labeling variables, it is reasonable to assume that the variable *int_max_tof_flag* is a Boolean value indicating whether the data stored in *int_max_tof* should be considered while performing the simulation. If this is the case, then it is reasonable to say that the GA found that the parameter *int_max_tof* is the single most influential parameter, with a 96% certainty rate. Two other parameters, *see_to_intercept* and *int_min_tof*, also were named as the single most influential parameters, though only for a small fraction of the experimental runs. Table 1 contains the frequency with which individual parameters were found to be most influential by the GA.

Parameter Name	Occurrences
<i>int_max_tof</i>	73
<i>int_max_tof_flag</i>	23
<i>int_min_tof</i>	2
<i>shape_delay</i>	2

Table 1. Frequency of genetic algorithm results

7. Conclusions

The goal of this research was to determine which parameters were most influential upon simulation output. Once obtained, this information could be used as a part of a broader vulnerability analysis of the simulation package. The GA used in this study was able to determine the single parameter that most affected output from this single class of simulation package.

This is significant because when analyzing simulation software, there may be hundreds or thousands of parameters. This is particularly true in defense simulations where a desirable design feature is to have unclassified simulations whose results are only classified when classified parameters are used. By determining the parameters that most influence the outcome of a simulation, a vulnerability analyst quickly can determine which parameters to focus on as he/she evaluates the simulation software.

The findings of this study are significant because of the empowering nature of the information obtained. By allowing researchers to focus on influential parameters, a more thorough vulnerability analysis of the simulation package can be performed. Being able to prune away insignificant parameters reduces analysis time.

Requiring further study are the effects of using the modified steady-state survivor selection algorithm. This

method of survivor selection was used in this study because of its indicated potential to more rapidly evolve the population of candidate solutions. Further research should be performed to confirm the initial findings of increased efficiency, as well as explore any drawbacks.

8. References

- [1] Imsand, E., A. Sachitano, J.A. Hamilton, Jr. 2003. Reverse Engineering Vulnerabilities in Simulation Software. *Proceedings of the 2003 Advanced Simulation Technologies Conference*. San Diego, CA: Society for Modeling and Simulation International (SCS), 101-107.
- [2] Peters, M., W. Chatam, J.A. Hamilton, Jr. 2003. Simulation Exploitation Using Open Source Information. *Proceedings of the 2003 Advanced Simulation Technologies Conference*. San Diego, CA: Society for Modeling and Simulation International (SCS), 114-118.
- [3] Nguyen, B. 2002. An Agent-Based Vulnerability Assessment System Intended for Tactical Digitized Networks. *MILCOM: Proceedings of the 2002 Conference*. The Boeing Corporation, Vol. 2, 1481-1484.
- [4] Conner, M., C. Patel, M. Little. 1999. Genetic Algorithm/Artificial Life Evolution of Security Vulnerability Agents. *Proceedings of the 1999 MILCOM Conference*. The Boeing Corporation, Vol. 1, 739-743.
- [5] Schultz, A., J. Grefenstette, K. DeJong. 1995. Learning to Break Things: Adaptive Testing of Intelligent Controllers. *Handbook of Genetic Algorithms* (ed. Lawrence Davis). New York: Oxford Press, G3.5.1-G3.5.10.
- [6] Bäck, T., U. Hammel, H.P. Schwefel. 1997. Evolutionary Computation: Comments on the History and Current State. *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, 3-17.
- [7] Dozier, G., et al. 2001. An Introduction to Evolutionary Computation. *Intelligent Control Systems Using Soft Computing Methodologies*, (ed. Ali Zilouchian and Mohammad Jamshidi). New York: CRC Press, 365-380.
- [8] Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*. Reading, MA: Addison-Wesley Publishing Company, Inc.
- [9] Michalewicz, Z. 1994. *Genetic Algorithms + Data Structures = Evolution Programs (2nd edition)*. New York: Springer Verlag.
- [10] Arkin, R. 1998. *Behavior-Based Robotics*, Boston: MIT Press.
- [11] Forrest, S., S. Hofmeyr, A. Somayaji, T. Longstaff. 1996. A Sense of Self for Unix Processes. *Proceedings of the IEEE Symposium on Research in Security and Privacy*.
- [12] Davis, L. 1991. *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold.
- [13] Holland, J. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press.
- [14] Fogel, L.J., A.J. Owens, M.J. Walsh. 1966. *Artificial Intelligence Through Simulated Evolution*. New York: Wiley Publishing.
- [15] Kennedy, J., R. Eberhart. 2001. *Swarm Intelligence*. Morgan Kaufmann.
- [16] Russell, S., P. Norvig. 2003. *Artificial Intelligence – A Modern Approach*. Upper Saddle River, NJ: Prentice Hall.
- [17] Chaiyaratana, N., A. Szczala. 1997. Recent Developments in Evolutionary and Genetic Algorithms: Theory and Applications. *Genetic Algorithms In Engineering Systems: Innovations and Applications: Proceedings of the conference in Glasgow, UK*. New York: IEEE, 270-277.
- [18] Baker, J. 1987. Reducing Bias and Inefficiency in the Selection Algorithm. *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*. Cambridge, MA: Laurence Erlbaum Associates, Inc., 14-21.
- [19] Radcliffe, N. 1991. Genetic Neural Networks on MIMD. Ph.D. Dissertation., University of Edinburgh, Edinburgh, Scotland, United Kingdom.
- [20] Eshelman, L., D. Schaffer. 1993. Real-Coded Genetic Algorithms and Interval-Schemata. *Foundations of Genetic Algorithms 2* (ed. L. Darrell Whitley). San Mateo, CA: Morgan Kaufmann Publishers, 187-202.
- [21] Neville, M., A. Sibley. 2002. Developing a Generic Genetic Algorithm. *Proceedings of the 2002 ACM SIGAda International Conference*. New York: The Association for Computing Machinery (ACM) Press, 45-52.
- [22] Gottlieb, J., T. Kruse. 2000. Selection in Evolutionary Algorithms for the Traveling Salesman Problem. *Applied Computing: Proceedings of the 2000 ACM Conference*. New York: The Association for Computing Machinery (ACM) Press, 415-421.
- [23] Syswerda, G. 1989. Uniform Crossover in Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 2-9.
- [24] Moed, M., C. Stewart, R. Kelly. 1991. Reducing the Search Time of a Steady State Genetic Algorithm Using the Immigration Operator. *Tools for Artificial Intelligence: Proceedings of the 3rd International Conference*. New York: IEEE, 500-501.
- [25] Poirier, E., M. Ghribi, A. Kaddouri. 2001. Loss Minimization Control of Induction Motor Drives Based on Genetic Algorithms. *Electronic Machines and Drives: Proceedings of the 2001 Conference*. New York: IEEE, 475-478.
- [26] Sparta, Inc. 2003. CAPS Database Guide. Lake Forrest, CA: Sparta, Inc.
- [27] Sparta, Inc. 2003. CAPS Training Briefing. Lake Forrest, CA: Sparta, Inc.
- [28] Schaffer, D., et al. 1989. A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. *Genetic Algorithms: Proceedings of the Third International Conference*. San Mateo, CA: Morgan Kaufmann Publishers, Inc., 50-60.
- [29] Digalakis, J., K. Margaritis. 2000. An Experimental Study of Benchmarking Functions for Genetic Algorithms. *Systems, Man, and Cybernetics: Proceedings of the 2000 Conference*. New York: IEEE, Vol. 5, 3810-3815.

Endnote

¹In Single-Point Crossover, two parents are cut at a randomly selected “crossover site” to create two children. The first child is created by recombining the first part of the first parent with the second part of the second parent. The second child comes as a result of recombining the first part of the second parent with the second part of the first parent.

