SFTE Silver Symposium
Abstract

1994

The Use of Genetic Algorithms for Flight Test and Evaluation of Artificial Intelligence and Complex Software Systems.

Elizabeth Davies
John McMaster
Mary Stark
NAWC-AD Patuxent River MD 20670
(301) 826-6353
(301) 826-6381 (fax)

Artificial Intelligence (AI) technologies are emerging from research laboratories into industry, particularly into the field of aviation. One such application is the ARPA/USAF Pilot's Associate, a set of cooperating expert systems designed to aid the tactical pilot in a combat situation. There are currently no rigorous methods for the test and evaluation of such tactical airborne expert systems. The focus of this program, sponsored in part by the Office of Naval Technology, is the development of a test and evaluation facility for tactical airborne systems which incorporate artificial intelligence. In particular, the subject of this paper is the development of a genetic algorithm (GA) which will automatically find those test cases for which the system under test performs most poorly.

The search space of possible test cases for an expert system such as the Pilot's Associate, or indeed for a conventional software system such as the F/A-18 digital flight controls is practically infinite. Therefore a technique which could automatically find the most difficult test cases would provide a vast savings in time and resources during flight testing and provide a safer product for the fleet. Genetic algorithms have the potential to provide such a benefit. A genetic algorithm is a search algorithm based on the mechanics of natural selection. They have proven to be effective in optimization problems such as surface-to-air missile evasion in computer simulations. Our goal is to determine whether a genetic algorithm can successfully generate difficult test cases.

For initial development, the genetic algorithm was written in C++ on a Macintosh computer. The first optimization task given to the GA was an abstract geometric problem which examined less than 2% of the entire state space before the global optimum was achieved.

Encouraged by the performance of the GA on an abstract problem, the next task was more realistic. The GA was designed to generate an optimum flight path of a hostile interceptor against a friendly fighter escort. This work centers on the development of GA techniques to exploit the weaknesses of the fighter escort by proposing hard "test cases" consisting of flight paths of the hostile interceptor. Data from these experiments and a computer demonstration will be presented. Conclusions as to the overall merit of genetic algorithms for flight test and evaluation of artificial intelligence systems and complex conventional software systems will be made.

THE USE OF GENETIC ALGORITHMS FOR FLIGHT TEST AND EVALUATION OF ARTIFICIAL INTELLIGENCE AND COMPLEX SOFTWARE SYSTEMS

ELIZABETH DAVIES
JOHN MCMASTER
MARY STARK
NAWC-AD PATUXENT RIVER MD

Abstract

Artificial Intelligence (AI) technologies are emerging from research laboratories into industry, particularly into the field of aviation. One such application is the ARPA/USAF Pilot's Associate, a set of cooperating expert systems designed to aid the tactical pilot in a combat situation. There are currently no rigorous methods for the test and evaluation of such tactical airborne expert systems. The focus of this program, sponsored in part by the Office of Naval Technology, is the development of a test and evaluation facility for tactical airborne systems which incorporate artificial intelligence. In particular, the subject of this paper is the development of a genetic algorithm (GA) which will automatically find those test cases for which the system under test performs most poorly.

A computer demonstration of the genetic algorithm will be presented. Conclusions as to the overall merit of genetic algorithms for flight test and evaluation of artificial intelligence systems and complex conventional software systems will be made.

Introduction

The search space of possible test cases for an expert system such as the Pilot's Associate, or indeed for a conventional software system such as the F/A-18 digital flight controls is practically infinite. Therefore a technique which could automatically find the most difficult test cases would provide a vast savings in time and resources during flight testing and provide a safer product for the fleet. Genetic algorithms have the potential to provide such a benefit.

A genetic algorithm is a search algorithm based on the mechanics of natural selection. It simulates biological evolution by representing possible solutions as chromosomes and using an evaluation function, analogous to natural selection, to determine the worth of that solution . By repeated iterations, a simulated evolution occurs and the population of solutions improves. Eventually a "good enough" solution is achieved.

Genetic algorithms have proven to be a powerful technique for optimization problems such as surface-to-air missile evasion in computer simulations. They have also been used for submarine detection in a noisy environment and as a potential scheduling tool for the Space Station. Although genetic algorithms have existed as an AI technique for optimization since the mid 1970s, they have not previously been used for testing. Our goal was to determine whether a genetic algorithm could be run "in reverse" to automatically generate the most difficult test cases for a system under test.

For initial development, the genetic algorithm was written in C++ on a Macintosh computer. The first optimization task given to the GA was an abstract geometric problem which examined less than 2% of the entire state space before the global optimum was achieved.

Encouraged by the performance of the GA on an abstract problem, the next task was more realistic. The GA was designed to generate an optimum flight path of a hostile interceptor against a friendly fighter escort. This work centers on the development of GA techniques to exploit the weaknesses of the fighter escort by proposing hard "test cases" consisting of flight paths of the hostile interceptor. Data from these experiments and a computer demonstration will be presented. Conclusions as to the overall merit of genetic algorithms for flight test and evaluation of artificial intelligence systems and complex conventional software systems will be made.

Approach

Looking at the task of generating an optimum flight path of a hostile interceptor against a friendly fighter escort, this work centered on developing techniques that would be applied to the GA. First a Euclidean distance function was used as the evaluation function. While the evaluation function converged rapidly, unrealistic paths were being generated as solutions, such as negative altitude values. Next, improvements were made to the evaluation function. Jane's manual was referenced to put limits on the GA parameters, such as aircraft speed. This function computed the distance to the intercept point and accepted low angle-off during the last leg of the path. This evaluation function started producing some interesting but still simplistic flight paths.

Our next approach was to tune the various parameters of the GA. The choice of parameters can effect the performance of the GA. Decisions guiding a GA generation are defined by the GA operators. They are divided into the following categories: fitness scaling, population renewal, mating selection, crossover, mutation and creep, offspring acceptance, and population replacement. We began our investigation with population renewal and replacement, and offspring acceptance categories. A population renews itself by generating offspring and integrating those offspring with the current population. Two renewal methods are provided: generational and partial. Generational renewal requires that the whole current population is to be replaced by newly generated offspring. The partial renewal requires that two offspring are to be generated then integrated into the current population. Now the issue of replacement is relevant. Two replacement strategies were used : replace-least-fit and replace-uniform-randomly. Replace-least-fit requires that the two least ranked genomes of the current population are replaced. If replace-uniform-randomly is used then two different genomes from the current population are selected and replaced by the two offspring.

Given the generation of a pair of genomes, there remains the question of whether or not to accept them to the current population. This decision is made independently of their evaluation. It is made solely upon comparing the genetic makeup of each genome with the other newly created genome and possibly the genomes of the current population. Two acceptance methods are available: accept always and accept-if-unique. Accept-if-unique guarantees that the population does not become cloned.

In the first set of experiments the red path consisted of five legs and had a population size of 20. If replace-uniform-randomly was selected, it resulted in a large evaluation function. This was due to the fact that "good genetic" information may be temporarily lost when replaced with newly generated offspring. In one of the experiments, near the end of the run, the solutions changed drastically and improvement in the evaluation function occurred. This indicated that longer generations must take place in order for "good genetic" information to be generated and be replaced back into the current population.

If replace-least-fit was selected, the evaluation function was small from the beginning and converged very rapidly. This occurred because the genomes with the lowest rank were replaced with newly generated offspring. This would limit our search space and focus on refining good solutions. (See Figure 1.)

The next two sets of experiments consisted of doubling the number of legs in the red path and then increasing the size of the population to 100. (See Figures 2 and 3.) In these two sets of experiments the evaluation function values grouped into three categories. In the first group, if generational

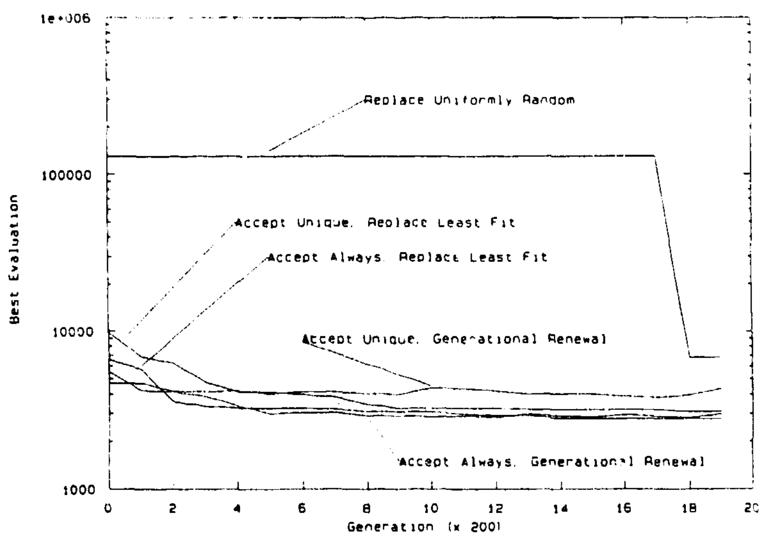

FIGURE 1

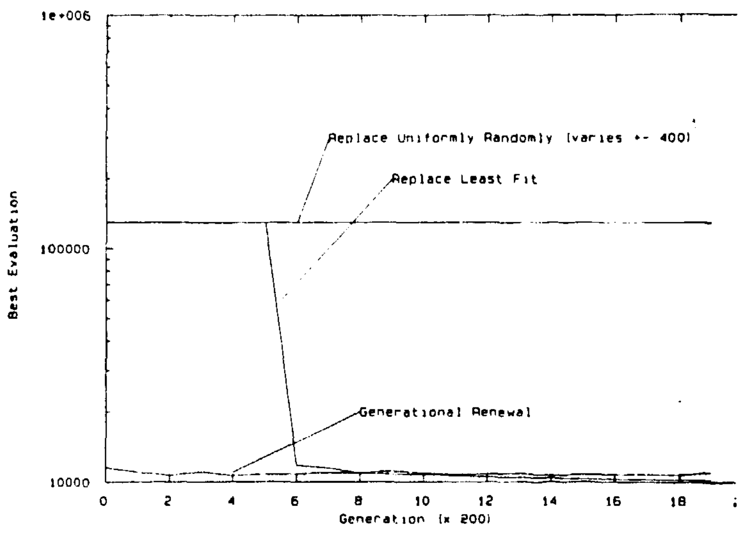

FIGURE 3

renewal parameter was selected the values of the function would jump around in the intermediate range. It would take almost twice the run time if this parameter was implemented. The second group of values occurred if the replace-least-fit parameter was selected. The function would take on very large values and then at some point in the generation cycle it would drop to intermediate values. If the replace-uniform-randomly parameter was selected the last group of function values occurred. The function values were very large and at some point the paths generated would change.

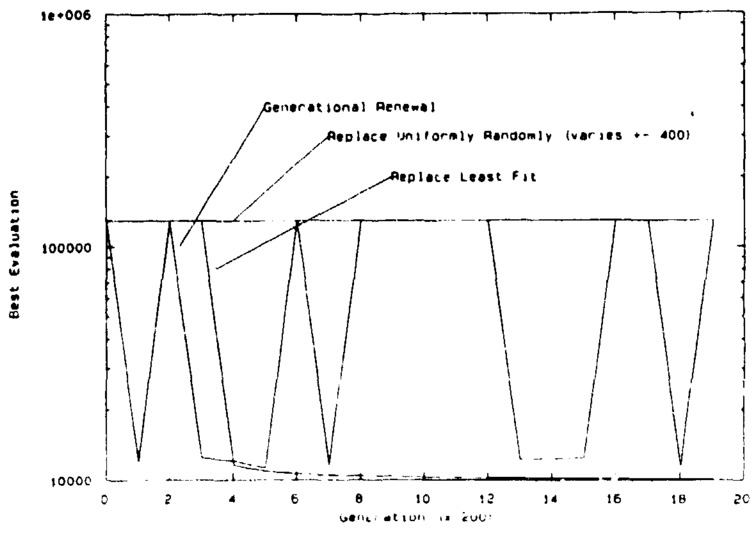

FIGURE 2

As a result of these experiments, it is recommended when running off-line tests that either generational or replace-uniform-randomly renewal schemes be used. Large population sizes are used to allow a diverse schemata available when generating test cases. For better on-line performance, a small population size is recommended because they have the ability to change more rapidly. For population renewal, replace-least-fit and for offspring acceptance, accept-if-unique should be used. Further investigation of the GA parameters will be part of our future work.

When looking at generating test cases using a GA, it is necessary to consider the types of weaknesses a test system is likely to exhibit in order to evaluate the merit of the GA approach. We reason that performance could vary in any of three ways as test inputs vary across their space:

1. As a continuous function such as the Pk Vs position in lethal cone geometry.

2. As one or more abrupt discontinuities such as a blind speed on pulsed doppler.

3. As one or more less abrupt discontinuities, that is local minima

Our investigation of the GA's ability to discover each of these types began with the use of a simple evaluation function representing the first kind of degradation of performance. This function employed the basic criteria of distance to a computed intercept point, low angle-off during the last moments before intercept and the interceptor's position within the target's 60 degree lethal cone, when the range to the target decreased below two miles. A plot of this evaluation function is shown in figure 4.

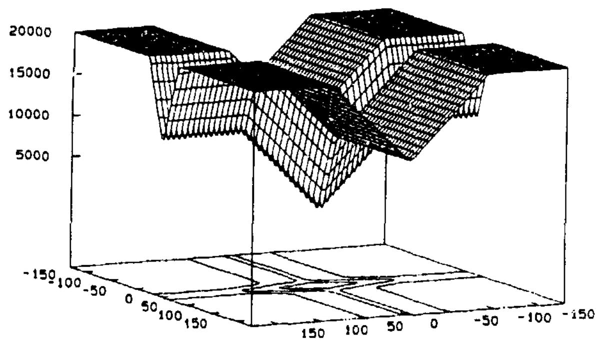

FIGURE 4

The experiments conducted with this evaluation function showed that the GA population was quick to converge on local minima, i.e. one of the "troughs" of the graph in figure 4, but did not locate the 60 degree lethal cone, in the time allowed. In essence, the GA spent its time learning that it must close on the target, rather than learning how to close on the target for optimum effect. We attribute this to our characterization of the problem.

The representation we chose represented values of thrust, roll, pitch and yaw, at a succession of points in time. Part of our evaluation function included criteria for closing on the target and for maintaining bearing to target. While this is a useful demonstration problem for a GA, it also allows the GA to expend effort in learning uninteresting information, i.e., that the target must be approached in the first place. In effect, the GA was able to "consider" all possible paths through the region.

It is recognized that "needle-in-the-haystack" problems, i.e., problems for which the evaluation function has only small (relative to the entire search space) and isolated minima require the most processing time for GA solution. Given a longer run time, our experiments would have found the 60 degree lethal cone, albeit inefficiently from a tester's standpoint. Moreover, the limited processing power of the Macintosh II desktop machine used for the experiments made extended runs impractical.

It is evident that in order to effectively apply GAs to testing of complex systems, the necessarily large space of possible test cases that the GA is will create must be partitioned and/or constrained in some intelligent manner to both minimize the time needed to locate performance degradation and obtain cases of performance degradation that are interesting to the tester.

It is reasonable to expect a test system's performance to degrade under conditions it was not designed to accommodate, therefore one immediate partitioning of the search space would be of realizable versus non-realizable test case parameter values. This partitioning of the search space would prevent the GA from generating and considering test cases which would not or could not be realized by the test system in the operational environment. In our own experiments, non-realizable cases (for instance, if the interceptor sought to fly below zero altitude) were punished by the evaluation function; an increase in processing speed would result if the partitioning took place at the representation level, i.e., non-realizable test cases would be detected as they are generated and not admitted into the population.

It is also reasonable to expect that certain portions of a test case would not be realistically expected to vary; to allow the GA to operate on these "immutable" portions of test cases is inefficient. One possible implementation of this idea would allow the GA to operate on codings of higher level abstractions of test cases, thus keeping the "immutable", lower-level detail out of the representation.

The search space constrained as above is still likely to be quite large and therefore a maximum of processing power is needed. Small local dips in performance may be as important to the tester as global "worst case" scenarios. Useful information about such small local minima could be lost in an extended run. A possible solution to this is the inclusion of a log facility, that would record snapshots of the population whenever a significant variation in overall fitness of the population is found.

Conclusions

In summary, our results suggest certain requirements of the GA tool and certain practices to be employed by the tester:

1. The GA tool must permit the user to partition the search space at the representation (chromosome) level. It must permit the establishment of immutable components of a test case and "reality checks" that forbid the entry of non- realizable test data into the gene pool.

2. The GA must permit the user to obtain "snapshots" of the population whenever the variation in the overall fitness of the population exceeds user defined limits. This would enable the investigation of small variations in test system performance that would otherwise be overlooked in pursuit of larger variations in performance.

Testers employing this approach should be expected to obtain and define the constraints on test cases for realizability and the components of test cases that are to be considered immutable.

Recommendations

At the time of writing, the GA library routines are being ported from the Macintosh II machine to an SGI Indigo Extreme. In addition to the extra processing power this affords, we expect to include enhancements that permit partitioning of the search space at the representation level and the ability to log and preserve snapshots of the population when significant variations in the population are detected. Investigation of the utility of these techniques will provide greater insight to the GA technique's capabilities for test case generation.

References

Booker, L. B. (1982) Intelligent Behavior as an Adaptation to the Task Environment. (Doctoral Dissertation, Technical Report # 243 Ann Arbor; University of Michigan, Logic of Computers Group). Dissertations Abstracts International, 43(2), 469B. (University Microfilms No. 8214966).

Davis, Lawrence (1991) Handbook of Genetic Algorithms. New York: Van Nostrand Reinhold.

DeJong, K. A. (1975) An Analysis of the Behavior of a Class of Genetic Adaptive Systems. (Doctoral Dissertation, University of Michigan). Dissertation Abstracts International, 36(10), 5140B. (University Microfilms No. 76-9381).

DeJong, K. A. (1980) A Genetic based Global Function Optimization Technique. (Technical Report # 80-2) Pittsburgh: University of Pittsburgh, Department of Computer Science.

Goldberg, David (1989) Genetic Algorithms in search, optimization, and machine learning. Addison-Wesley.

Grefenstette, J. J. (1986) Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man and Cybernetics, SMC-16(1), 122-128.

Schultz, A. C., Grefenstette, J. J., DeJong, K. A. (1992) Adaptive Testing of Controllers for Autonomous Vehicles. Proceedings of the Autonomous Underwater Vehicles 1992 Conference, June 1992.