EVOLUTIONARY PROGRAMMING TECHNIQUES FOR MODELING OF C2 PROCESSES

Steven M. Alexander

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
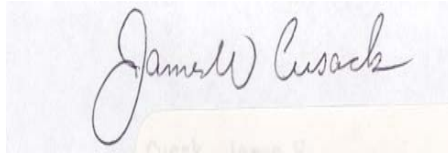**ROME RESEARCH SITE**
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-283 has been reviewed and is approved for publication.

APPROVED:

STEVEN D. FARR, Chief
C4ISR Modeling and Simulation Branch
Information Systems Division
Information Directorate

FOR THE DIRECTOR:

JAMES W. CUSACK, Chief
Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE January 2002	Final In-House, Jan 2000 – Oct 2000

4. TITLE AND SUBTITLE
EVOLUTIONARY PROGRAMMING TECHNIQUES FOR MODELING OF C2 PROCESSES

5. FUNDING NUMBERS

PE: 61102F
PR: 2304
TA: ER
WU: B1

6. AUTHOR(S)

Steven M. Alexander

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/IFSB
525 Brooks Road
Rome, NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER

AFRL-IF-RS-TR-2001-283

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/IFSB
525 Brooks Road
Rome, NY 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2001-283

11. SUPPLEMENTARY NOTES
AFRL/IF Program Manager: Steven M. Alexander/IFSB, (315)330-4304
E-mail: alexanders@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

This report contains a summary of the work done on this effort over the past twelve months. Two projects were performed, one of which investigates the basic ideas of evolutionary programming (EP) techniques, the other is an attempt to apply EP to an air campaign simulation. The first project provided evidence that EP can find the ideal solution in a complex space. It also seemed to show that a faster convergence to a good solution may be obtained by keeping the various parameters involved in EP within certain limits. Although there was insufficient time to complete the second project, it did show that it is possible to use neural networks (NNs) to control a two-dimensional entity in two-dimensional space, and that even randomly initialized (untrained) NNs can exhibit interesting behavior.

14. SUBJECT TERMS Evolutionary Programming, Genetic Algorithms, Command and Control, C2			15. NUMBER OF PAGES 13
			16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Introduction

Evolutionary programming is a computational technique pioneered by Dr. Lawrence Fogel. The following passage is taken from an abstract written by Dr. Fogel on the topic of using Evolutionary Programming (EP) to solve military simulation problems.

> Real-world military problems are extremely complex: they contain the prediction of uncertain events, the control of incompletely understood processes, and the management of extensive distributed resources in the face of an intelligently interactive OPFOR. The constraints are nonlinear and the objective function changes as the situation develops. To make matters worse, there are far too many feasible solutions ... alternative ways of getting the job done. An exhaustive search to find the best solution is clearly impossible.
>
> The conventional approach relies on simplification. Each challenge is broken into component problems that can be more easily addressed, but local optima do not add up to a global optimum unless the components are independent ... and they rarely are. We call upon linear programming even when the constraints are known to be nonlinear. Steepest descent is used even when the response surface may have multiple modes, is discontinuous, noisy, or in the limit, has no gradient. Spectral analysis and Markov processes are used to predict time series even when the actual environment is known to be non-stationary. These methods often yield the right answer ... to the wrong problem! (Fogel [1])

EP is an optimization tool used to search for a viable solution in the solution space of a given problem domain. It is modeled after the process of evolution observed in nature, using the concepts of Darwinian evolution to perform its search of the solution space. Each solution within the solution space consists of a set of information called a genome. This genome consists of many individual genes, which may be represented by any appropriate alphabet; including binary digits, alphanumeric characters, and real numbers. Within the constraints of a specific model, this genome determines the behavior of its corresponding solution. In turn, the fitness of a particular solution within its environment is determined by evaluating the behavior of that solution against a fitness function. This implies a direct relationship between the information describing the solution and the distance of that solution from the optimal solution.

The process of EP mimics that of biological evolution, although the two processes are not necessarily identical. Many variations of EP have been used to solve a vast array of computational problems. A general description of the process of EP is given here. First a population is generated, in which each individual's (solution's) genome is randomly initialized. Next, the fitness of each individual is determined by evaluating the individual against a fitness function. This fitness function assigns favorable fitness values to individuals that exhibit desired behavior, and poor fitness values to individuals that show unwanted behavior. The researcher generally determines the definition of "desired behavior." Individuals that receive higher fitness scores are more likely to be selected. Selection of the best n% individuals occurs. The individuals who are not selected are generally thrown away. The selected individuals are then used for "mutation" and/or "crossover", so that much of their genetic material is propagated to the

next generation of individuals. Mutation consists of altering a very small percentage of the genes in an individual's genome. This introduces new information and consequently new solutions into the population. Information is kept from previous solutions if found favorable during fitness evaluation. Crossover consists of swapping genetic information between two solutions, resulting in new combinations of genetic information. This injects new individuals into the population; some of which may represent favorable solutions. After mutation and crossover have occurred, the new population is put through a fitness test, and the entire process repeats. For a more complete explanation of EP, see (Fogel [2]).

Evolutionary Programming has been used in many applications. It has been used to generate adaptive behavior in a platoon-level engagement of tanks where the mission of one platoon is changed on-the-fly (Fogel [3]), to evolve a checkers player that attained master level over time while competing against human players in an on-line checkers game room (Fogel[4]) and for many other interesting and important applications (Fogel [5]).

The objective of this project was to explore evolutionary programming techniques and choose a campaign or mission-level problem domain in which to illustrate the applicability of evolutionary programming for decision support. The final product was to be a demonstrable program showing application of evolutionary programming to air campaign assessment.

During the course of this investigation of EP, work was performed on two separate but related experiments. Both consisted of Java 2 code developed in-house on a PC under Windows 98. The first of these was an all text-based program, with user input and program output occurring through an MSDOS console window. The second experiment utilized a graphical user interface (a Java applet) in which the user could input the required parameters and press on-screen buttons to activate the program. Output was represented visually in the applet. Both of these experiments represent incremental progress toward the ultimate goal of this effort.
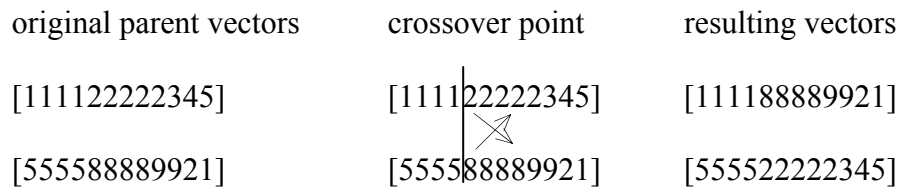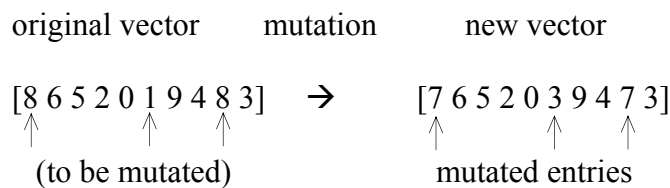
Project 1: Vector Evolution

Methods, Assumptions, and Procedures

The concept for the first project was to design code which would optimize the entries of a 100 dimensional vector whose possible values were contained in the set $S = \{0, 1, \ldots, 9\}$ using EP techniques. It was set up in the following way. A population of vectors is randomly initialized, so that each vector in the population has 100 entries x_1, x_2, ... x_{100}, where x_i is an element of S. The user determines the number of vectors in the initial population.

Each vector is then compared to the "ideal" vector, which was chosen to remain the same every time the program is executed. The vector [0 1 2 3 4 5 6 7 8 9 0 1 2...9...9] was used, which is a 100 entry vector that repeats the digits 1 through 9 in counting order 10 times. For the purposes of this paper, comparing a vector from the population with the ideal vector translates to determining the number of vector entries that are equal between these two vectors. For example, if the first entry in a given vector from the population is a 3, this would not be a match, since the ideal vector's first entry is

always 0. A score was obtained for each vector in the population by comparing it with the ideal vector and giving one point for each matching vector entry.

The population of vectors is then put in rank order by their scores. A user-determined percentage of the vectors are selected as "parents" for the next generation of vectors. The remaining vectors are discarded. Mutation and or crossover algorithms are then applied to these parent vectors, resulting in a new population of vectors equal to the size of the original population. The user is allowed to determine the mutation rate, as well as whether or not mutation, crossover, or both will be used in the population. Mutation works by changing a few randomly selected vector entries by a limited amount. Crossover works by randomly choosing the location of a single parent vector entry, then swapping all of the vector entries following that entry with the corresponding entries in another parent vector. The purpose of both algorithms is to create new individuals for the population.

original vector mutation   new vector

[8 6 5 2 0 1 9 4 8 3] →   [7 6 5 2 0 3 9 4 7 3]

(to be mutated)   mutated entries

Figure 1. Illustration of mutation

original parent vectors crossover point   resulting vectors

[111122222345]   [1111|22222345] [111188889921]

[555588889921]   [5555|88889921] [555522222345]

Figure 2. Illustration of crossover

The process of comparison, scoring, ranking, selection, and mutation and/or crossover continues for as long as the user wishes, or until 100% conformity with the ideal vector is reached by at least one individual within the population. At this point the program terminates, and may be restarted in the case that the user wishes to use different parameters.

At this point it is useful to mention that EP does not require that the optimal solution to a problem is known prior to the EP's implementation. Indeed, knowledge of the optimal solution would negate the need to use any optimization method. The solution to this problem was known before starting the search so that the researchers could follow the solutions obtained through EP directly, and to make programming easier. Having the optimal solution makes it easy to determine the fitness function for an EP program; the investigator need only compare solutions calculated by the program to the optimal solution. Starting without the best solution, one must find appropriate measures of fitness for individuals within the population of solutions, so that it is possible to determine which solutions are the best of the current generation.

Results and Discussion

After much debugging and rewriting of code, this program finally started working well. Within a given population, there was rapid progression towards the ideal vector. The rate of evolution was found to be highly dependent on the rates of mutation and crossover, as well as the size of the population, and the percentage of individuals selected as parents for each successive generation. The following figures (3-5) show some of the trends that presented themselves as experiments were run with varying values for the input parameters. The program was run with population sizes of 10, 50, 100, 200 and 500 individuals. The population size remains constant throughout the execution of a single run of the program. For each of these population sizes, the program ran using three different mutation rates; first without using the crossover function, then in conjunction with the crossover function. These experiments were then repeated using various values for the number of offspring selected as parents for the next generation. Figure 3 shows a comparison of population size to the average number of generations it took for a vector within the population to reach 100% conformity to the ideal vector. There is a clear trend indicating that as population size increases, the number of generations decreases. There also seems to be an exponential relationship between these two variables.
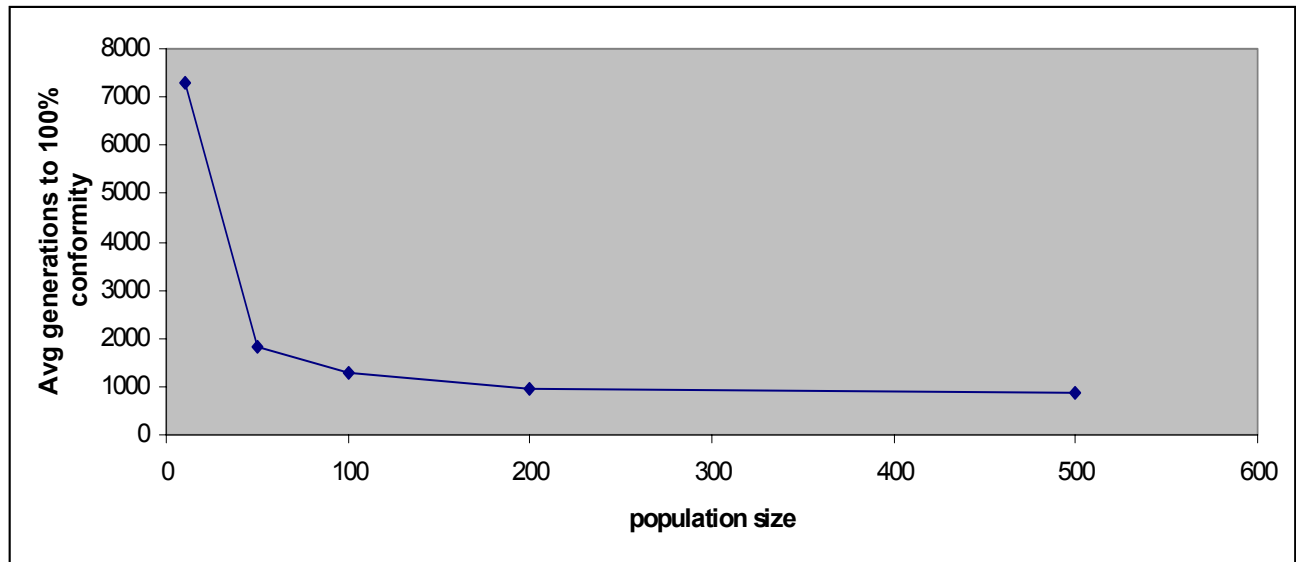

Figure 3. Average generations until convergence to ideal vector values vs. population size. Number of generations decreases exponentially as the number of individuals in the population increases.
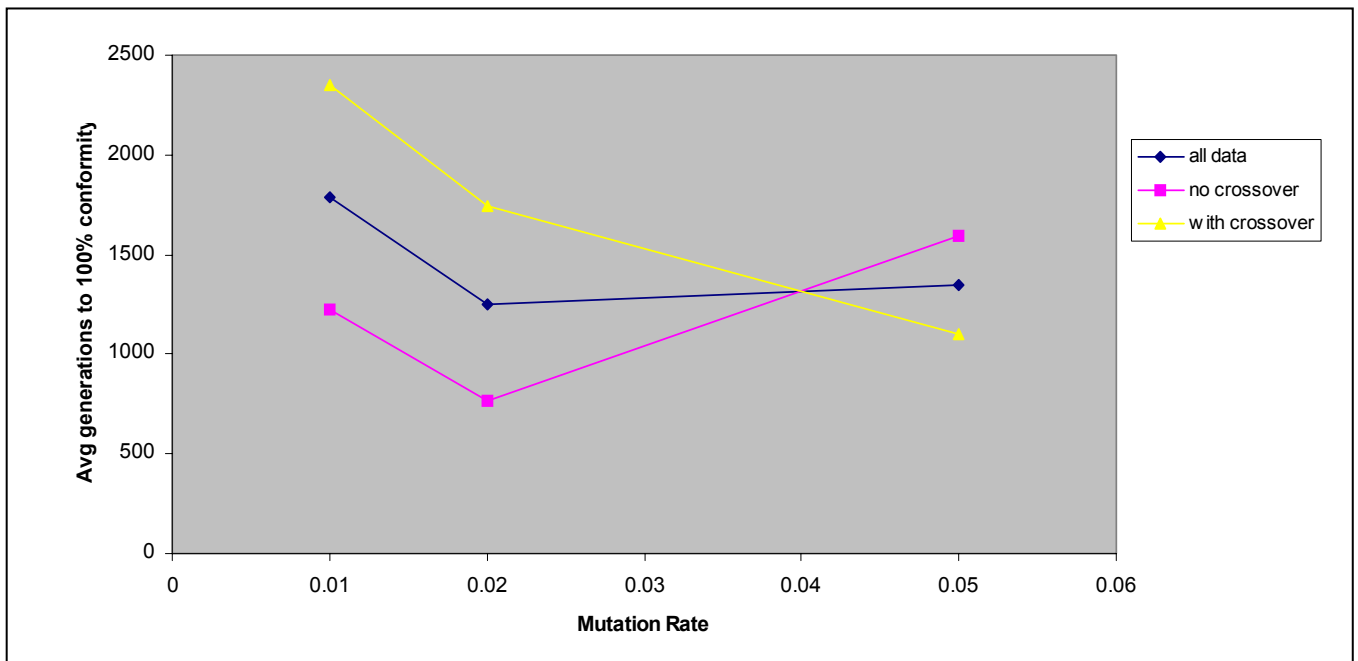
Figure 4. Average generations until convergence to ideal vector values vs. mutation rate. The fastest observed mutation rate in terms of generations to conformity is 0.02. Crossover may be beneficial at high mutation rates.
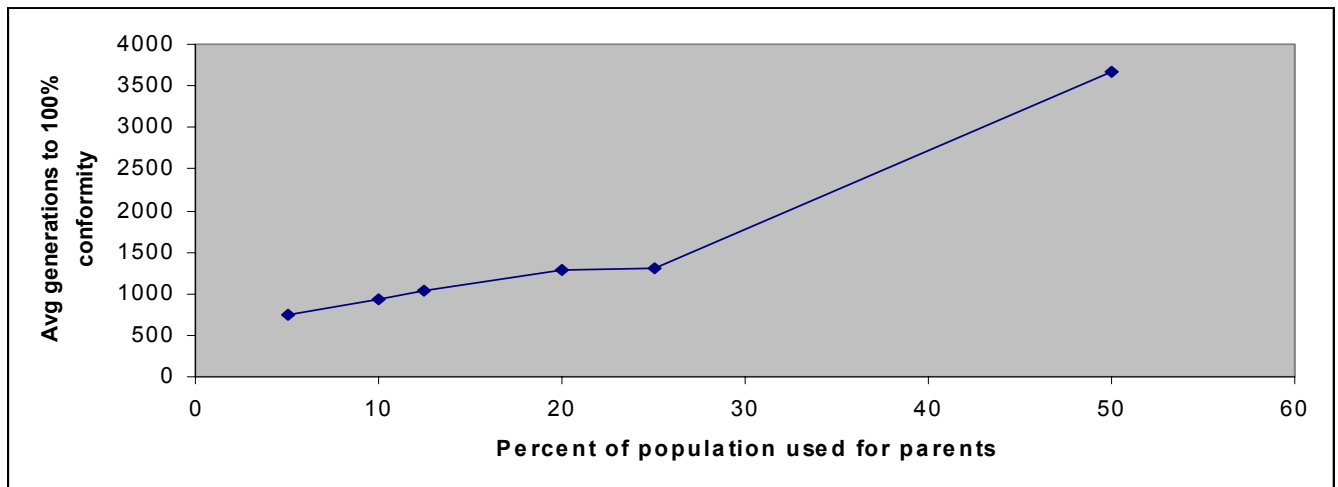

Figure 5. Average generations until convergence to ideal vector values vs. percentage of population kept as parents for next generation. More elitist selection produces fit individuals in fewer generations.

Figure 4 shows how the mutation rate affects the rate of convergence to the ideal vector in terms of generations. The three mutation rates tested were .01, .02, and .05. A mutation rate of .01 means that 1% of the entries in each vector in the population was changed during each generation; so on average, one entry per vector was changed. The values were changed by +/- 1, 2, or 3, and were constrained so that the maximum value was 9 and the minimum value was 0. This figure shows that .02 was the best mutation rate tested. Since it lies between .01 and .05, and both these values produced a slower evolution process, the optimal mutation rate is most likely somewhere between the two.

Figure 5 shows the relationship between the rate of convergence to the ideal vector, and the percentage of the population selected as parents for subsequent generations. For example, if the population size is 100, with 10% chosen as parents, then each generation, the best 10 individuals from the population of 100 will be selected. These will then be mutated and/or crossed over to generate a new population of 100 individuals. This process repeats until convergence is reached or the user of the program decides to quit. The graph shows an increase in generations as the number of parents increases; which suggests that, for this problem, selecting a very small group of highly fit individuals is better than selecting a larger group which includes some less fit individuals.

It is important to remember that the results of this experiment do not necessarily imply optimal or even useful relationships among the variables described above. Factors such as population size and mutation rate will have different optimal values depending on the problem at hand. This idea is emphasized by what is called a "No Free Lunch Theorem," which states that any gain in an algorithm's performance in solving a particular type of optimization problem will result in degradation of its ability to solve some other type of optimization problem. This means that no single algorithm can be the most suitable for all optimization problems (Wolpert and Macready [6]).

Despite this theorem, there do seem to be some useful guidelines to follow when using evolutionary programming techniques. One of these is to keep the mutation rate low. The problem with a high mutation rate is that once the individuals in the population of possible solutions get near the optimal solution, mutation may cause more damage than good. Consider, for example, if a 20% mutation rate were to be used in the vector evolution program above. Assume that some individuals within a population had reached 95% conformity to the ideal vector; that is, assume that 95 out of 100 vector entries matched those of the ideal vector. At a 20% mutation rate, 20 of the vector's entries would change, most likely resulting in an individual of lower fitness than before. Even if all five of the non-matching entries were changed to matching entries, 15 other matching entries would most likely change to non-matching entries, resulting in an 85% optimized individual. One possible way to improve convergence time might be to use a relatively high mutation rate in the beginning of the evolution process, and then lower the mutation rate as the average fitness of the population of solutions increases. This way, a wide initial search of the solution space would occur, but when individuals became very fit, the good genes they had evolved would not be disrupted as easily.

Another guideline that seems generally useful is to have a large population. Depending on the problem type, a population of 20 to 100 individuals may be optimal, but a population of five will almost always be too small for optimal convergence time. With only five randomly initialized individuals at the beginning of the problem, there is only a very small chance of getting an individual with a high fitness. The more individuals there are in the population, the better chance there is of randomly initializing a somewhat fit individual. So why not have a population size of 1 million? This would make computation time so long that the added benefit of more random individuals would most likely be lost. The number of generations to a viable solution would be lower, but the computation time for each generation would be longer.

Project 2: Evolved Neural Networks for modeling UCAV control

Methods, Assumptions, and Procedures

The goal of this project was to use EP techniques to develop a computer program that could model the control of an unpiloted (combat) air vehicle (UCAV). It was necessary to vastly simplify this problem to be able to make progress, given the limited time and manpower. EP must be used in conjunction with some sort of control model. In this project, Neural Networks (NNs) were chosen as the controlling entities. The variables of the NNs, called weights and biases, are part of what determines the outputs from a NN, given a set of inputs. The structure of the NN, i.e. the number of layers, number of nodes per layer, and how the nodes connect to one another, is also an important factor in how inputs are mapped to outputs. In this project, the structure was chosen and remained constant. The weights and biases were organized into an array, and evolution was performed on populations of such arrays, as in the vector project. The key difference in this project is that the "ideal" array was not known before starting the evolution, so the fitness function had to be based on the behavior of the UCAVs being controlled by the NNs.

Although NNs are very useful for many artificial intelligence (AI) applications, they can be complex and time consuming to work with in Java code. The model and control system used in this project is described here, even though complete results were not obtained. A simpler approach, involving the evolution of rule-sets instead of NNs, is described afterwards.

The model that was developed is a bounded two-dimensional area in which targets and "obstacles" are randomly placed. An obstacle represents anything that might hinder the progress or safety of the air vehicle, such as anti-air missile structures. The actual air vehicle is a 2D object that can move in any direction on the plane. Velocity and acceleration were not modeled; only position was modeled. The goal was to evolve a behavior for the air vehicle that could successfully navigate the area, which entails avoiding obstacles and visiting each target once.

This project was written in Java, visualized through the use of an applet. The user may enter the number of targets and obstacles into text boxes in the applet, then press the "Setup" button. This causes the selected number of targets and obstacles to appear in random locations within the designated flight area. There is currently no limitation restricting targets and obstacles from overlapping each other, so it is possible for more than one target and/or obstacle to exist in the same spot. Each target is represented as a white circle within a red circle, and the obstacles are represented as blue squares. When the user presses the "Start" button, the air vehicle appears in the center of the flight area, and begins to move.

A feed-forward Neural Network (NN) was used as the controlling code for the movement of the air vehicle. There are inputs to the NN which correspond to the distance between the air vehicle and each target and obstacle. The maximum number of targets is currently set at 3, and the maximum number of obstacles is 10. There are inputs to the NN which correspond to distances to each of these objects. There are four outputs from the NN, three of which trigger a behavior, and one that determines the current focus of the air vehicle. The program is written so that the air vehicle focuses on only one of

the targets or obstacles at any given time step, and it is always performing one of three behaviors with respect to that target or obstacle. The three behaviors are seeking, fleeing, and circling. When seeking, the air vehicle moves towards the object currently in focus. When fleeing, it moves away from the focus object. When the air vehicle is circling, it should be moving in a circle around the focus object. Currently, the orbit method is not fully implemented.

After implementation of the NN and the seek and flee methods, it was determined that a less complicated approach to this problem might produce better results in less time. A proposal written by Dr. Kuo-Chi Lin outlined a way to control UCAVs by evolving rule-sets. Each rule set would consist of a binary string that maps a UCAV's sensory inputs to maneuvers, which the UCAV subsequently executes. This is a simpler, more intuitive way to accomplish the goal of UCAV control than through the use of NNs. It should require much less effort, and unlike with NNs, it will be possible to extract the control information out of the solutions obtained through evolving rule sets.

Conclusions

This effort shows that EP is a useful tool when applied to optimization problems. There are many problems within the field of EP that have not yet been solved, such as how to determine appropriate values for parameters such as population size, mutation rate, and crossover rate. At this point, the best way to do this is to experiment until good parameters are discovered, but this does not necessarily mean that the best parameters are being used.

Applying EP to the problem of UCAV control is a complicated task, but it is not impossible. Evolving rule sets may be the best way to show a proof of concept for this challenge, while more intricate methods, such as the use of NNs, may provide additional realism to such simulations in the future.

Acknowledgements

References

1. Lawrence J. Fogel, Abstract submitted to AFRL/IFSB, 1999.

2. D.B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, Piscataway, NJ, 1995.

3. V. William Porto, D.B. Fogel, and Lawrence J. Fogel, "Generating Novel Tactics Through Evolutionary Computation," *Sigart Bulletin*, Vol. 9, No. 2, pp.8-14, Fall 1998.

4. D.B. Fogel, "Evolving a Checkers Player Without Relying on Human Expertise," *Intelligence*, Vol. 11, No.2, pp. 20-27.

5. D.B. Fogel, "What is Evolutionary Computation?" *IEEE Spectrum*, pp. 30-32, February 2000.

6. D.H. Wolpert and W.G Macready, "No Free Lunch Theorems for Optimization*,"* *IEEE Trans. Evolutionary Computation*, Vol. 1:1, pp. 67-82, 1997.

List of Symbols, abbreviations, acronyms

AI – Artificial Intelligence
EP – Evolutionary Programming
NN(s) – Neural Network(s)
UCAV – Unpiloted Combat Air Vehicle