AN APPLICATION OF REAL TIME EVOLUTIONARY ALGORITHMS

David R. Pratt
Science Applications International Corporation
Orlando, Florida

ABSTRACT

Historically, CGF systems have made runtime decisions via the use of prescriptive mechanisms such as Finite State Machines (FSMs) and Rule Based Systems (RBS). The FSM and RBS mechanisms are the result of a complex and time-consuming process of Knowledge Acquisition and Knowledge Engineering (KA/KE). The artifacts of the KA/KE process are then turned over to the programmers to implement. The result is often a large, complex, and brittle set of hard coded behaviors. The advantage to these approaches is that the entities execute the pre-programmed behaviors faithfully and fairly efficiently. The downside is that it is often quite difficult to modify the behaviors to account for new events, stimuli, or situations. To address these issues we have looked at the realm of machine learning, specifically the use of Evolutionary Algorithms (EA), to make decisions that have historically been hard coded in the FSM or RBS constructs. The use of EA is not new to the Computer Generated Forces (CGF) community. However, the vast preponderance of its use has been in *a priori* offline runs to develop a rule base, plan of attack, or path. This has largely been due to the computational costs of using EA. While the increase in processing speed has not made performance considerations irrelevant, they have fundamentally changed the dynamics of the development vs. runtime cost equations. It is with this in mind that we chose to investigate the use of EA to make selected decisions at runtime. Specifically, we developed a proof of principle system to select the engagement rules and target priorities for a tank platoon in a given tactical situation. Rather than having the prescriptive determination of the engagement process, the EA subsystem randomly generates a set of shooter / target pairings and weapon selection. The EA subsystem evaluates a set of possible engagements using a polynomial function comprised of the proximity and obscuration of the entities, supporting fires, and lethality. The highest rated engagements, and newly generated modifications of them, are carried forward to the next generation by the EA subsystem. The process of the evaluation of a best engagement scenario is then repeated for a given number of generations. At the end of several generations, or when a figure of merit is reached, the best engagement scenario is chosen as the course of action. The main advantage to this approach is a relatively small amount of code needed to encode the EA mechanisms and the evaluation function which can be easily changed to account for new weighting of factors. Thus, a whole series of target selections can be made with a relatively compact flexible code base. This paper covers the development of the proof of principle system and the results from the test runs. Specifically, we focus on three factors: the number of engagement scenarios created per generation, the number of generations, and the evaluation function. Through the interaction of these three factors, we show how the engagement scenarios evolved to suit the tactical scenario. Key among the considerations is the time it takes for the system to come up with a viable target list. From these results, we make extrapolations to where it is appropriate to use EA as a means of developmental cost reduction and code simplification. This is the second in a series of papers that addresses the use of EA in real-time simulation systems. The first paper focused on the ability to change formations based upon the detection of a threat.

ABOUT THE AUTHOR

Dr. David R. Pratt is a Chief Scientist/Fellow at SAIC's Applied Software Systems Engineering Technology (ASSET) Group. As the Group's Technical Lead, he coordinates the internal research activities of the group and establishes the technical directions. Prior to joining SAIC, he served as the JSIMS Technical Director, a tenured Associate Professor of Computer Science at the Naval Postgraduate School, and was a Captain in the Marine Corps. Dr. Pratt received a Ph.D. in Computer Science in 1993 and a Masters of Science degree in Computer Science in 1988 from the Naval Postgraduate School.

AN APPLICATION OF REAL TIME EVOLUTIONARY ALGORITHMS

David R. Pratt
Science Applications International Corporation
Orlando, Florida

INTRODUCTION

Historically, CGF systems have made runtime decisions via the use of prescriptive mechanisms such as Finite State Machines (FSMs) and Rule Based Systems (RBS). The FSM and RBS mechanisms are the result of a complex and time-consuming process of Knowledge Acquisition and Knowledge Engineering (KA/KE). The artifacts of the KA/KE process are then turned over to the programmers to implement. The result is often a large, complex, and brittle set of hard-coded behaviors. The advantage to this approach is that the entities execute the pre-programmed behaviors faithfully and fairly efficiently. The downside is that it is often difficult to modify the behaviors to account for new events, stimuli, or situations. As we develop new systems that must adapt to a an ever-changing world, the cycle time in updating the behaviors renders modifications to the system obsolete before the system is deployed.

To address these issues we have looked at the realm of machine learning, namely, the use of Evolutionary Algorithms (EA), to make decisions that have historically been hard-coded in the FSM or RBS constructs. The use of EA is not new to the Computer Generated Forces (CGF) community. However, the vast preponderance of its use has been in *a priori* offline runs to develop a rule base, plan of attack, or path. This use has largely been due to the large computational costs of using EA. While the increase in processing speed has not made performance considerations irrelevant, they have fundamentally changed the dynamics of the development vs. runtime cost equations. It is with this in mind that we chose to investigate the use of EA to make selected decisions at runtime. Specifically, we developed a proof of principle system to select the target list for a four gun artillery battery in a given tactical situation.

Rather than having the prescriptive determination of the target list, the EA subsystem randomly generates a set of target and round selections. The EA subsystem evaluates a set of possible target / round pairs using a polynomial function comprised of the proximity of the entities, target types, and probability of kill (P_k). The highest rated target lists, and newly generated modifications of them, are carried forward to the next generation by the EA subsystem. The process of the evaluation of a best target list is then repeated for a given number of generations. At the end of several generations, or when a figure of merit is reached, the best target list is chosen as the course of action. The main advantages to this approach are that there is a relatively small amount of code needed to encode the EA mechanisms and the evaluation function can be easily changed to account for new weighting of factors. Thus, a whole series of target list selections can be made with a relatively compact flexible code base.

This is the second in a series of papers where we have been exploring the uses of EA in runtime decision making. The first paper discussed the applicability of the EA as a means of determining formations for a platoon of tanks when presented with a threat [1]. This paper looks at the problem of round and target selection for an artillery battery.[1]

[1] focused a considerable amount of the mechanics of the EA and their interplay between the number of chromosomes and generations. From this research we determined that 200 of the chromosomes and 500 generations would provide a reasonable convergence to a solution, provided the evaluation function was robust and tractable.

This paper covers the development of the proof of principle systems and the results from the experimental runs. From these results, we make extrapolations to where it is appropriate to use EA as a means of developmental cost reduction and code simplification.

OBJECTIVES

The goal of the series of studies was to evaluate the applicability of the use of EA in on-the-fly behavior generation and selection. As such, there were a series of subgoals:

- Construct a suitable software testbed
- Develop a series of evaluation functions
- Conduct a series of time/space and evaluation function experimental runs
- Gather the appropriate data to evaluate the results

[1] Since [1] is not widely available, some of the introductory discussion concerning EAs and the software testbed are repeated in this paper.

- Extrapolate the results to a larger setting
- Suggest further study in this area

The following sections discuss how we went about achieving these goals.

BASIC EA CONCEPTS

While a thorough introduction to EA can be found in [1], the following discusses the fundamentals required for this paper. As the name implies, the foundation idea behind EA is that of Darwinian evolution: survival of the fittest. The concepts of genetics, with the exception of dominant and recessive genes, make this possible. In the simplest form of EA, which we used, the best chromosomes of population N are kept for generation N+1. The rest of generation N+1 is made up of chromosomes that resulted from the mating of selected chromosomes in generation N and a number of mutant chromosomes. It is through the crossover and mutation of chromosomes that new values are introduced to the gene pool. This is key because, if no completely new chromosomes are introduced, successive inbreeding can result in the propagation of undesirable traits. Hence, in every generation a limited number of new chromosomes are introduced. An evaluation function is used to determine which chromosome represents the strongest members of the species. These chromosomes are then sorted by the evaluation function and the cycle repeats.

By continually selecting the best of the generation, the result of the fitness function will eventually converge on the "best" solution. This might or might not be the optimal solution, if one is known, but it should be close enough to the goal. The problem of getting trapped in a local minimum is handled by the use of the mutations.

Since the results are not guaranteed to be the optimal solution, EAs are typically used on the class of problems that do not have universal solutions in predictable time or problems with a large solution spaces. In the case of our simple problem of target and round selection, there are roughly 96 possible solutions per volley. Clearly, this is too many to try using brute force techniques. Hence the use of EA to provide a guided exploration of the solution space.

One of the things that intrigued us about using the EA approach is the time adaptive algorithm nature of the process. Since each generation provides, theoretically, a better solution to the problem than the previous one, we should be able to trade off how long the EA process runs verses how "good" the solution is. We saw this as a direct parallel to real world decision making, the more time available, the more refined and, theoretically, better the decision is.

SAMPLE PROBLEM

The problem selected for this set of experiments was based upon a suggestion of research topics contained in [2]. Specifically, the use of artificial intelligence techniques for the automatic selection of targets and round types for an Artillery Battery. Since the purpose of this research was to evaluate the use of an EA system, vice provide a functioning system, the problem framing could be simplified. In doing so, we made several simplifying assumptions. Shown in Table 1 is the data used in the development of the problem. From this we can see that there were a limited number of targets and round combinations possible. Likewise, we did not consider the effects of range on the accuracy of the guns or anything short of a complete kill.

Table 1. Data used in the simplified artillery model

	Infantry	Tank	Tanker	Artillery
Max Range	10.0	0.0	25.0	50.0
Threat Value	0.9	0.0	0.5	0.7
P_k for each type of round / target pair				
HE[2]	1.0	0.8	0.2	0.7
Armor	0.1	0.3	0.8	0.3
WP	0.8	0.5	0.1	0.5

That is not to say that we did not use the range between the guns and the target as a means of doing target prioritization. As shown in Figure 1, the threat function is a range dependent function based upon the maximum effective threat range from the data file, Table 1, and the range of the actual target. It is important to note that both Artillery and Tankers had a constant threat value. This provided a means of target prioritization that looked roughly like this:

- Close in Infantry
- Artillery at any range
- Tanks within range
- Infantry within range
- Tankers, Tanks and Infantry out of range

[2] The actual type of round and P_k are made up for use in this problem and bear very little, if any, resemblance to the real values.

While it was simple enough to deduce this list from the figure, we wanted to see if the system could discover the relationship.

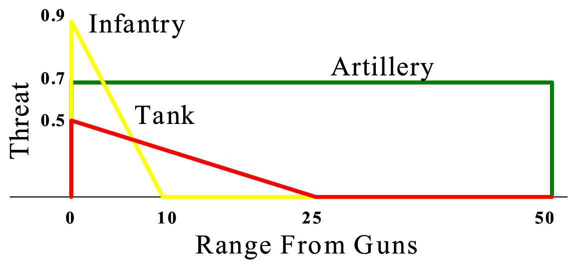


Figure 1. Graphical depiction of the threat function

Target selection was only one part of the problem. The other was the matching of the round type to the target. Once again, using the AFCS method described in [4], we can derive Table 2. However, the problem was constructed to see if the system could learn the same relationships.

Table 2. Round Selection by Target Type

Target	Primary Round	Secondary Round
Infantry	HE	WP
Tank	Armor	HE
Tanker	HE	WP
Artillery	HE	WP

The final challenge of this problem was having multiple volleys. By firing multiple rounds per gun, and weighting the effect of each volley, we could attempt to see if the system would go after the high value targets first and then shift fires to lower value ones.

COMPUTATIONAL PLATFORM

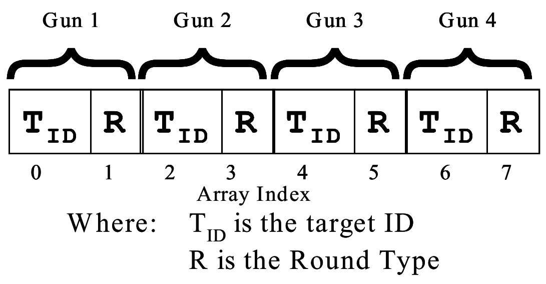
Since the timing aspects of this study are an important part of the overall assessment, the computational platform plays a role in the evaluation. The hardware platform was a Dell Latitude CPx laptop with a Mobile P-III 650Mhz processor. Additionally the system had 128Mb of RAM. The operating system was Microsoft Windows 98 Second Edition build 4.10.222A. The Java environment was: Java (TM) 2 Runtime Environment, Standard Edition (build 1.3.0-C) with Java Hotspot (TM) Client VM (build 1.3.0-C, mixed mode). Based on our previous work [5], we felt that this was a representative computational platform for this type of study.

THE EA SOFTWARE TESTBED

Since we plan on using the testbed for follow-on studies, we made the conscious choice to implement the system in the most portable manner possible. The four major classes are discussed below. They are linked together by means of a Driver class that controls the number of chromosomes and generations based on user input. It is also the Driver that calls the visualization at the end of every generation to help monitor the evolution.

The genes and chromosome

The gene is the foundational element in an EA system. It is here where all of the knowledge is captured. The genes are then grouped in a chromosome for manipulation and propagation. In this experiment, we used a simple integer array to represent the values as a single logical entity. Using a hierarchal model, we grouped the genes representing the Target ID and Round type by gun, and then, by volley. While we kept the battery size fixed at four guns, we varied the number of volleys between one and four. This allowed us to use a table driven input scheme, thus ensuring the consistency of the results. The resulting chromosome for a single volley run is shown in Figure 2.



Gun 1
Gun 2
Gun 3
Gun 4
T_{ID} R T_{ID} R T_{ID} R T_{ID} R
0 1 2 3 4 5 6 7
Array Index
Where: T_{ID} is the target ID
R is the Round Type

Figure 2. The Elements of the Chromosome

The use of an array, vice a record structure, provided a decoupling of the semantics of the data used in the evaluation function from the syntax of the data used in the EA Solver.

Simulation

Each chromosome was used as the input parameters for a simple artillery simulator. For each run of the simulator, a set of 8 targets was generated randomly. The type of target was chosen from the four categories listed in Table 1. The location was randomly chosen between the minimum range, 0, and the maximum range, 50. All the targets were marked as alive. The firing guns were assumed to be at location 0. Each gun in a volley would fire independently and once a target was destroyed, it stayed dead for the remainder of the run.

The target selection for each gun was done via the (T_{ID},R) pair from the chromosome. For each (T_{ID},R) pair, a random number was drawn between 0.0 and 1.0 using the standard Java Random Number Generator. If the number was less than the value from the input P_k table, the entity was marked dead. After a volley was fired, the live entities were counted. Likewise, using only the live entities, the threat value was computed. Once a volley had been fired and the results computed, the next volley would fire.

It is important to note that there was no changing of the chromosome during a run of the simulation. Thus, even if there was a high probability of kill and the random number came up against the shot, the entity could not be retargeted. Likewise, it is possible that an entity might be targeted more than once in a volley and a dead entity targeted and fired on.

Evaluation function

One of the advantages of using the EA approach is that it is possible to evaluate the chromosome along several different metrics concurrently. This is done by breaking the evaluation function down into a series of subfactors. The results of the subfactors are then summed to provide the overall figure of merit (FoM) for the chromosome. Since each of the subfactors might be on different scales, the result of each can be weighted using a scalar or an expression. The resulting FoM expression looks like this:

$$FigureofMerit = \sum WeightingFactor_i * SubExpression_i$$

To capture the effect of multiple volleys, we used an enhanced FoM function that weighted the previous volley's FoM higher then the current volleys. This resulted in the following equation:

$$FoM_i = 2 * FoM_{i-1} + FoM_{current}\ .$$

This approach worked well since we were trying to minimize the overall FoM.

For this experiment, we used scalars as the weighting factors, although, on different runs we modified the relative values to examine the interplay between the subexpressions. The subexpressions are detailed in the following sections.

It is essential to note that not all subexpressions were used in all trial runs. By changing the weighting factors associated with the various subexpression, we were able to emphasize one factor at the expense of the others. This points to the "art" in the use of EA approach, that of crafting the evaluation function and the appropriate weightings.

Number of Live Targets

Given that we had a fixed number of targets at the beginning of the run, the number of targets killed and the number of live targets are simple inverses of each other. Since we are trying to minimize the overall FoM, the number of live targets provided the logical choice to measure the effectiveness of the fires.

Threat Value

The other subexpression was a measure of the enemy threat the firing battery was exposed to. As described above, the four types of targets each had different effective threat values. In the cases of the Infantry and Tank threats the effective range played a part in the determination of the overall threat. The algorithm used to determine the effective threat value is shown in Figure 3.

```
If (Artillery or Tanker) Then
   Return (ThreatValue)
Else if (in Range) then
   Return (ThreatValue * (1 – Range / MaxRange))
```

Figure 3. The Threat Subexpression

EA solver

As shown in Figure 4, the EA solver was an iterative process. The first step was the creation of the first generation of chromosomes. The number of chromosomes in a generation, as well as the number of generations, was controlled by a parameter provided to the constructor. The values for the genes in the first generation would be generated randomly. Once the first generation was run, the evaluation function was run on all of the chromosomes to generate the subsequent generations.

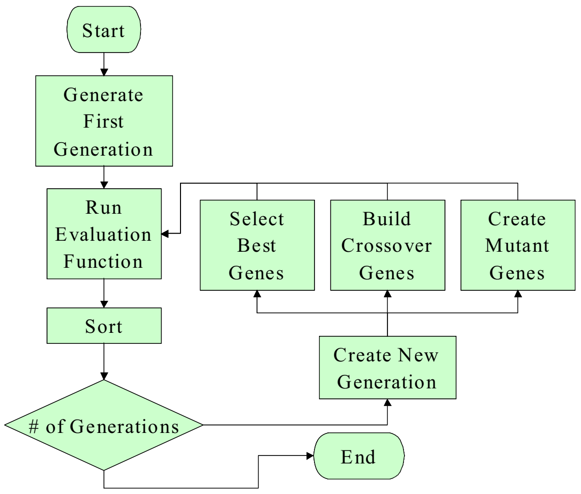


Figure 4. The EA solver flow of control

In keeping with the object oriented nature of Java, the evaluation function was modeled as class with a single public function. The various subexpressions were developed as protected functions. This ensured the encapsulation of the evaluation process. Once the entire generation was evaluated, the chromosomes were sorted in ascending order of the Figure of Merit (FoM). At this point, if the maximum number of generations was reached, the system would halt and report the "best" candidate.

If this was not the final generation, a new empty generation was created. To populate this generation, the top quarter of the previous generation was brought forward. The second and third quarters were made of the combination of two adjacent chromosomes (even probability of each one contributing a selected value). The final quarter was comprised of randomly mated chromosomes. These were added to prevent the population from getting too homogenous. Both mating functions had a low, five percent, mutation rate. The mutation rate was higher than in some other trials to reduce the dependence on the initial population. Once this process generation was completed, the generation was evaluated and the process started over.

TEST RUNS

A series of experiments were run to characterize the solution space and investigate the interplay between run parameters. To reduce anomalies, each test was run ten times and the results were averaged. In this section we discuss the individual experiments and their setups, present averaged results and draw conclusions from them. With the exception of the timing runs test case, all test were run with 200 chromosomes and 500 generations.

Timing Runs

We used the CYGWIN_98-4.10 as the command shell and developed a batch driver that ran the test in an automated mode. To reduce the variability in the test results, a constant random number seed was used, the computer was disconnected from the network, no other applications were running, and the screen saver was turned off. Each one of the chromosome/generation pairs were run one hundred times and the results averaged. Shown graphically in Figure 5, the results show the linear growth of the execution time as a result of the product of the number of generations and chromosomes. The slightly higher than expected result in the 10 chromosome/10 generation case can be attributed to the start up transient cost that was not amortized over the length of the run. This result is consistent with the results obtained in [1].

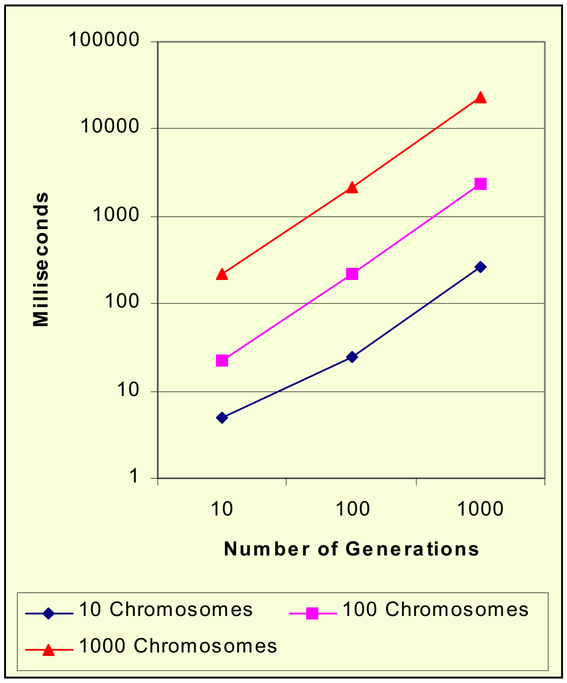


Figure 5. Results from the timing runs

Single Volley, Maximum Kills

In this experiment, and all the subsequent experiments, a different random number seed was used for each of the runs. This provided the variability we needed to provide insight in the effectiveness of the EA approach. The number of rounds per gun value was set to one, to ensure that single volleys would be fired. To ensure that the emphasis was set on maximizing the number of kills (actually, we were minimizing the number of remaining targets), the weighting factor for the threat value was set to zero.

As expected, based on the input parametric data shown in Table 1, the system selected the Infantry as the primary target. This was followed by the Tanker and Artillery. Very few Tanks were targeted. The HE round was the mostly commonly used round overall. Also as expected, (Infantry / HE) pairing was the primary target/round pair. All this is consistent with our expectations based upon the input data.

Multiple Volleys, Maximum Kills

We ran this set of test cases twice. The first time we did not have any incentive to target the easy targets first. To do this, we just ran the evaluation function after all volleys had been fired. In essence, we treated the system like a single volley of many guns. As such there was no prioritization of the targets. The easy to kill targets were spread out through out the entire chromosome.

On the second set of two runs, those with the weighting factor to reward kills in the early volleys , we noticed a different behavior. In sets, the Tankers and Infantry were targeted early on and the Tanks were saved for the later volleys. This was not as prevalent when there were only two or three volleys. We believe this was due to the cases where the easy to kill entities were not killed with the first round. In cases where there where more than three volleys, the Tanks would be targeted multiple times in the later volleys.

Single Volley, Minimum Threat

The lesson learned from this experiment was to go after the Artillery first; next, deal with the close Tanks. While the close in Infantry had a greater threat value, there were not sufficient cases for the system to learn to deal with them on a priority basis. Likewise, the Tankers were completely ignored. This makes sense since they did not represent any threat and were the same as shooting things that were dead or out of range.

Multiple Volleys, Minimum Threat

We found out that many of the same things in the single volley case, apply to the multiple volley case. The Artillery was targeted first, then the close in Tanks, finally, everything else. Since out of range and dead entities had the same threat values, after two volleys there was a wide range of targeting variations. Both in terms of selection of targets and round types. Basically, they were firing to fire, there was minimal if any contribution to the FoM.

Multiple Volleys, Maximum Kills and Minimum Threat

In the final experiment, we combined the two subexpressions to produce the overall figure of merit. Through manipulation of the relative values of the two weighting factors, we were able to produce results that were skewed to either of the multiple cases described above. To evenly weight the two subexpressions took some doing. The number of dead was expressed in integer values, whereas the threat value was a float value less than one. In most cases, it was below 0.5. To produce a balanced result, the threat subexpression had to be multiplied by a value roughly four times greater than the dead count subexpression. It was interesting to note that when the two subexpressions were weighted equally, the result was a seemingly random firing pattern. While the target type and round selection made sense, the selection of the targets seemed to be inconsistent with our expectations. Upon closer investigation we determined that the two subexpression were actually in conflict. The things that were easier to kill were the ones that presented less threat.

CONCLUSIONS

We caveat our findings by reiterating the limited scope of this project. First and foremost, we caveat our findings by acknowledging that the paper covers a very limited and straightforward set of problems. We are sufficiently confident in our validity work that they can be generalized and used as a point of departure on which to base system level decisions. We have developed and implemented evolutionary algorithms to address relatively simple problems in CGF entity behavior modeling, formation changing in the previous paper and target selection in this one. We examined the feasibility of using the EA approach, a traditional off-line computer learning methodology, as a replacement for the commonly used real-time finite state machine and fixed rule set approaches. EA are useful primarily in a non-time constrained mode where a directed search of the solution space is needed. While this is primarily in the offline learning mode, it can still be applicable to a case where the system is "stuck" using the current rule base or a new situation that was not covered in the rules comes up.

While the computational cost of the run time EA far exceeded what could reasonably be allocated to solving a simple problem like this one, we were encouraged to see that the system behaved as we expected and learned to perform the desired actions. In doing so, the importance of a complete understanding of the Evaluation Function and the FoM can not be overemphasized. This represent the art that is still prevalent in the design of an EA system.

One of the side benefits we found very interesting was the sorting of the final generation. In essence, what it provides is a rank ordering of the courses of action available. As such, we believe that such a system can also provide advisory information to decision makers when selecting courses of action. In doing so, the human is providing the oversight and selecting the appropriate set of actions. This is particularly important when dealing with actual vehicles where the world might not be completely modeled and the system might generate some course of action that is unsafe or impracticable for reasons not included in the model.

FUTURE WORK

Based on our findings from this work, as well as the results from our previous study, we are planning on using EA for behavior generation in cases where the desired behaviors are not known or readily apparent. Specifically, we are planning to look into the co-evolution of friendly/opposing tactics and doctrine and equipment.

ACKNOWLEDGMENTS

This work was done under SAIC internal research funding. The author would like to thank Aurora Jablonowski for her help in editing the paper.

REFERENCES

[1] Pratt, David, "Use of Evolutionary Algorithms for Runtime Decision Making", The 10th Conference on Computer Generated Forces & Behavior Representation, Norfolk, Virginia, May 2001

[2] Marshall, Henry, et al, "Fabulous Graduate Research Topics in CGF, Improving the Army Graduate Program", The 10th Conference on Computer Generated Forces & Behavior Representation, Norfolk, Virginia, May 2001

[3] Mitchell, Melanie, "An Introduction to Genetic Algorithms," MIT Press, 1999, ISBN: 0-262-13316-4

[4] Henninger, Amy, "The Limitations of Static Performance Metrics for Dynamic Tasks Learned Through Observation," , The 10th Conference on Computer Generated Forces & Behavior Representation, Norfolk, Virginia, May 2001

[5] Pratt, D., Courtemanche, A., Moyers, J., and Campbell, C., "An Empirical Evaluation of Programming Languages for Computer Generated Forces," The 9th Conference on Computer Generated Forces & Behavior Representation, Orlando, Florida, May 2001