Genetic Programming and Co-Evolution with Exogenous Fitness in an Artificial Life Environment

Michael Waters
TTC Incorporated
20410 Observation Drive
Germantown, MD 20876-4023
(301) 353-1550
watersm@ttc.com

John Sheppard
ARINC
2551 Riva Road
Annapolis, MD 21401
(410) 266-2099
jsheppar@arinc.com

ABSTRACT - The study of artificial life involves simulating biological or sociological processes with a computer. Combining artificial life with techniques from evolutionary computation frequently involves modeling the behavior or decision processes of artificial organisms within a society in such a way that genetic algorithms can be applied to modify these models and enhance behavior over time. Typically, endogenous fitness is used with co-evolution. In this paper, we explore the use of an exogenous fitness function with genetic programming and co-evolution to develop individuals and species capable of competing in a hostile environment. To facilitate the study, we use a commercially available environment—AI Wars—to host the organisms and run the experiments. Results from our experiments, though preliminary, indicate the ability of co-evolution, genetic programming, and exogenous fitness to evolve fit individuals. The results also suggest the ability to assess the nature of the fitness landscape and the impact of various fitness factors on evolutionary performance.

1 INTRODUCTION

The concepts of biological evolution assume that creatures develop capabilities to adapt to their environment. For a species to persist in an environment, it must be able to compete successfully for resources against other species. For this experiment, we have two nascent species that are adapted to their environment and the resources available in their environment. The goal is to see if they will learn to compete successfully against each other as might occur in nature.

This experiment is set in the environment of the computer game "AI Wars." In a typical AI Wars game, a player hand-codes a program that will direct the actions of a virtual "bug" that competes in a fight to the death against other player-coded bugs. The game has little to do with AI, since the programs are static, human-written, and use no AI algorithms. The environment, however, is well structured for an investigation of genetic programming, co-evolution and Artificial Life.

1.1 Motivation for Competitive Co-Evolution

Adaptation to an environment is the first (and most general) requirement for a species to establish itself. The ability to secure an adequate food source in the specific environment determines whether a species stays, moves on in search of other environments, or becomes extinct. This requirement is sufficiently general that every environment on earth is replete with a variety of different species in the same area.

Competition arises in nature in predator-prey relationships or when several species compete for the same limited food source. For a species to persist in an environment, it must be able to compete successfully for resources. The success of this process in biology has lead to the natural diversity that is evident in every environment. For this experiment neither species is assigned the role of predator or prey: it is part of the experiment to see if such behavior arises "naturally."

It would be possible to evolve a single species for this experiment using the traditional techniques of genetic programming by assuming a static environment. An alternative idea is to pit a single species that is starting from a random state against a hand-written bug that is already able to compete in its environment. Although the developing species might, in time, learn to deal with the predator, it seems intuitive that the species would only learn to deal with the one particular static predator against which it must compete. To attempt to mimic the results of nature, and to promote the development of bugs with more general capabilities, it is preferable to mimic a more natural environment–one in which there is competition and where all participants in the competition evolve. For these reasons we decided to work with two species–both of whose programs start as randomly generated code–and compete them against each other.

1.2 Related Work

Previous research in evolving solutions to complex tasks indicates that competition between sets of simultaneously learning agents can produce superior solutions [AP93, AP92, FP94]. The work of Ficici, Angeline and Pollack supports the idea that if a set of learning agents simply compete against a static opponent they are likely to learn the

0-7803-5536-9/99/$10.00 ©1999 IEEE

behaviors of the opponent and come to a potentially sub-optimal solution. If two or more simultaneously learning populations are involved, an "arms race" may arise that will drive the populations towards better and better solutions.

Co-evolving populations that can avoid interaction may converge to what Ficici and Pollack called a "mediocre stable state" [FP94]. They cite, for example, situations in World War I in which opposing soldiers in the trenches developed ritualistic shows of force that would satisfy their respective commanders but expose themselves to little risk of injury or success. To deal with this, Ficici and Pollack had three subjects in their experiment; two attempting to cooperate and one that was considered the opponent. They found that this tension drove towards better solutions.

This project differs from research in predator-prey evolution [NI97] and pursuit-evasion games [CM96, She97, LS96] in two substantial ways. First, both species have equal capabilities. Second, both species have the same goal. Neither species is assigned the role of pursuer/evader or predator/prey. It is left to the process of evolution to determine if these roles arise.

1.3 AI Wars and A-Life

The game AI Wars [Red98] does not use artificial intelligence; it can serve as a readymade Artificial Life laboratory. Much A-Life work is involved in the fundamental task of developing software representations that behave in a manner similar to biological organisms. Some of this work involves co-evolution, competition, parasitic behavior, and symbiotic behavior [Ray91]. In AI Wars we are not experimenting with the biology or morphology of our creatures. We are given two species with identical capabilities, and we are attempting to determine whether they can develop the high-order reasoning capabilities that will allow them to use their physical capabilities to survive in a competitive environment.

Much of the work in Genetic Programming (GP) involves attempts to grow computational functions from low-level constructs (individual programming-language words and variables). Examples include the development of adders, sorters, and pattern-matching routines [AP92, Koz92, Koz91, Koz92, Spe96]. This project starts with relatively high-level constructs assembled in random order, with random parameter values. The AI Wars programming language already includes complex constructs such as "if damage is > 90% then..." so it is not necessary to evolve them. The manipulation is in the placement of the statements within the program and in the manipulation of the consequent (the things that come after the "then..."). The intent is to grow complex general-purpose independent agent behavior.

In all other respects it follows the concepts of GP. Programs are copied and modified. Subroutines are developed and shared. The difference is that complete, valid lines of code–not words within the line–are the basic unit being manipulated. The GP mutation operators employed are similar to those described in [Koz96].

2 HYPOTHESIS

The goal of the experiment is to evolve bug programs using co-evolution that perform better than randomly generated code. A more ambitious goal is for one or both of the species to evolve bug programs that are capable of competition against each other and that are general enough to compete against hand-coded bugs. We hypothesize that the endogenous fitness assessment represented by the game score will be insufficient to drive a vigorous evolutionary process.

We refer to the fitness function applied by AI Wars as *endogenous* and the fitness function we develop and apply in an attempt to drive evolutionary towards particular behaviors as *exogenous*.

Assessing progress (and testing the above hypotheses) in a co-evolutionary environment is problematic unless the specific behaviors evolved are examined. Thus tracking the progression of individual or species fitness by itself is somewhat uninformative, especially when using endogenous fitness. For this reason, we examine evolved behaviors as well as track the progression of exogenous species fitness.

3 APPROACH

3.1 Program Representation

Initially, several bug programs are generated at random. The bug programs consist of blocks of code where a block consists of a named subroutine with one or more lines of code. Blocks can branch to other blocks but always return to the calling block. Each bug program consists of a mainline program and zero or more subroutines. The mainline and subroutines are all code blocks.

3.2 Initial State

The initial population consists of randomly generated legal programs. Invalid programs are not generated, thus enforcing the interaction of viable individuals and eliminating the possibility of either species containing inert individuals. The bugs are initially generated as a set of integer and string tokens representing valid commands. These tokenized programs are then translated into text versions. A code fragment of a bug program is given in Figure 1. The text version of the bug is then run through a proprietary encryption algorithm to produce a file format acceptable to the AI Wars program.

```
author waters
iff code s1
name bg12s1
generate random
top:
if scan found nothing then lay mines on
if no ammo then scan forward
if random is 4 then turn right
if scan found barrier then scan position 6
math vu = #strat_x / #damset11
if x coordinate is = 80 then scan right
lay mines on
gosub izdpx
if scan found enemy then move backwards
goto top
```

Figure 1. **AI Wars Code Fragment**

For each generation, there are two sets of tournaments–intramural and interspecies. A tournament consists of a series of battles, with up to 10 bugs per battle. Battles take place in a discrete 2-dimensional arena. Objects in the arena include flags, mines, and barriers. Flags repair damage and replenish fuel and ammunition. If a flag is taken when the bug has no damage, the result is an overload that damages the bug. Mines are passive enemies: whoever steps on one is damaged. Barriers prevent movement and detection but can be maneuvered around.

Bugs all have "identify friend or foe" (iff) codes representing their species. In this way, bugs of the same species can identify each other during inter-species competitions: during intramural competition, iff-detection is disabled.

3.3 Intramural Tournaments

In intramural tournaments groups of up to 10 bugs from the same species compete against each other. Each bug is guaranteed to be in at least one battle, but there is no guarantee that every bug will be matched against every other bug in the species.

After the intramural tournament, the aggregate battle results are computed from the battle reports. Statistics computed for each bug include; number of battles, raw score (total of all scores from all battles–see Appendix), scaled score, and fitness.

Raw bug scores can range from minus infinity to plus infinity; therefore, the scores are scaled to a minimum of zero for ease of calculation. To do this the lowest raw score is captured for the species and (if it is less than zero) added it to each raw score to compute the scaled score. The fitness for each bug is calculated as (scaled score) / (sum of species scaled scores). Reproduction opportunities are then allocated for each bug using fitness proportionate selection.

3.4 Interspecies Tournaments

Once the intramural tournaments for each species are finished, the two species compete against each other. Five bugs from each species are placed into each battle. Once all the interspecies battles have been run, scoring and mating are handled in the same manner as with intramural tournaments. Mates must come from the same species.

The selection and replacement mechanism implements a steady-state genetic algorithm. Specifically, in each new generation, four new bugs (two offspring from the intramural and two offspring from the interspecies tournaments) are generated and replace the four least fit individuals of their respective species from the previous generation.

3.5 Genetic Operators

Both mutation and crossover are used. We used five mutation operators: *duplication, deletion, restoration, substitution* and *modification*.

Duplication: The selected line is copied and inserted immediately following the original line.

Deletion (Deactivation): The selected line is deactivated by being commented out. The "return" and the "goto TOP" statements are exempted from mutation since commenting out these lines would be destructive to the block structure of the programs.

Restoration: If the selected line is one that had been previously commented out (deleted), it is restored to its active state.

Substitution: The selected line is replaced with a new, randomly generated, valid line. If the random command generator generates a "gosub", an entirely new subroutine is generated.

Modification: A single value of the selected line is changed. The value to be modified is randomly selected from all the tokens in the line of code. If the random selection indicates a change to the first token - which is the primary command on that line - modification would be equivalent to Substitution. Since this is not the intent of Modification, these cases are passed through unmodified. Integer tokens are randomly increased or decreased, resulting in either new parameter values or new consequent commands.

Crossover – occurring 100% of the time in reproduction – is implemented as two-point, where contiguous subsets of code from the mainline routines of the two parents are swapped to create two different offspring. If the copied code contains a call to a subroutine, the entire referenced

subroutine is copied into the offspring. De-activated code (code that has been subject to the Deletion operator) is also copied and available for Restoration in later generations. Mutation probability is relatively high (10%). Each line of code in each new bug is subject to the possibility of mutation.

3.6 Fitness Function

The AI Wars program assigns a score to each bug that reflects the amount of damage done to enemy bugs while keeping the damage to itself low. This usually means that the last bug alive is the winner of a battle, but it is not unheard of for the bug with the highest point total to have been killed before the battle has ended. This score seems to be an appropriate fitness function since it is an integer value with an initial setting of zero, it can go negative if a bug is particularly inept and is not explicitly bounded. It reflects the bugs' ability to defend itself, to locate flags (food) and to damage competitors. The details of the scoring used by AI Wars are provided in the Appendix.

As it turned out, this endogenous fitness function was inadequate to achieving the kinds of results desired. An exogenous fitness function that amplified the rewards for certain specific acts (subgoals) was added to the third experiment. Specifically, the original fitness function was simply the score given by the game to the bug at the end of the battle. We added to this value a reward/penalty function for specific events that took place during the course of the battle. These events are captured in the textual narrative of each battle report and include direct hits with missiles and energy weapons, indirect (shrapnel) hits, stepping on landmines, capturing flags and firing missiles with shields up. Beneficial events are given large additive rewards and detrimental events are given large subtractive penalties.

4 IMPLEMENTATION

The environment used for the experiments is AI Wars version 3.7c. Programming was done in Microsoft Visual Basic for Applications (VBA), integrated with Microsoft Excel under Microsoft Windows. The Microsoft scripting product "Scriptit" was used to coordinate the execution of tournaments and the integration of the programs. The test machines were Pentium II with CPU clock speeds > 200mHz.

4.1 Initialization

A VBA program is used to generate the initial population. Two populations of individuals are generated – one for each species. Subroutines are written to separate files to facilitate potential sharing of subroutines between individuals in later generations. The text of the subroutines is copied onto the end of the individual's code for compilation and execution.

The output of this generation process is a set of tokenized files, with integers representing commands and parameters. These files are then mapped into valid commands in separate text files (e.g. "39" → "Return"). These text files are then encrypted for execution by the AI Wars game.

4.2 Execution

A VBA program is used to run the simulations, process the results, and produce generations after the first (Generation Zero). Recognizing that manually running the battles, both intramural and interspecies, would be extremely tedious, a scripting utility from Microsoft called "ScriptIt" is used to coordinate the execution of the components. The VBA program writes scripts and invokes the script runner, which in turn invokes AI Wars. Each generation is stored in a separate directory, as is each species and each tournament, thus facilitating analysis by providing uncorrupted snapshots of the evolutionary process.

5 RESULTS

The time required to run each generation of bugs proved to be considerable. With that in mind, these results should be considered preliminary. Experiments are continuing.

Three experiments were run to a point where identifiable trends emerged. The first was with two 50-member populations run to 24 generations. The second was with two 20-member populations that converged after only 7 generations. The third was with two 20-member populations that are still running. The results in this paper represent the output of this third experiment at generation 143.

The chief difference between the first and subsequent two experiments was the range of programming constructs available. The first experiment limited the commands being used to the simplest commands available. The latter two experiments allowed the bugs to use the full range of the AI Wars programming language. The third experiment involved the use of the exogenous fitness function described in section 3.6. In all three experiments two fundamental trends emerged–self-destructive behaviors were eliminated and overall fitness increased over time.

5.1 Eliminating Self-Destructive Behaviors

Three different patterns of self-destructive behavior were eliminated in early generations. These behaviors included firing missiles with shields up, laying landmines and then immediately stepping on them, and repeatedly executing the "Discharge Energy" command.

Firing missiles with shields up results in the full missile damage being inflicted on the firing bug. Bugs exhibiting this behavior received significantly negative scores and were eliminated from the population. Individuals that fired

missiles in later generations had learned to lower their shields first.

When mine laying is turned on, a mine is laid for each step the bug takes until it runs out of ammunition (or turns mine laying off). The most common self-destructive form of this behavior was observed to be the following code fragment:

```
Lay Mines On
Move forward
Move backward
Move forward
```

This code resulted in the individual stepping on two mines it had just laid itself. In general, mine laying was not eliminated but is far less common in later generations than in early generations. Individuals that did lay mines tended to proceed in a single direction, either forward or backward, instead of turning or reversing course.

The energy discharge command inflicts one point of damage to the bug executing the command while clearing mines in the immediate vicinity. It also inflicts three points of damage to any bugs that are next to the executor. In human-written bugs that compete well, it is common to execute "Discharge Energy" three times in succession if an enemy is detected in the surrounding spaces, but only after ensuring that the procedure will not be self-destructive. Ten executions of "Discharge Energy" in a row will destroy the bug discharging the energy.

Individuals in early generations were seen to perform energy discharges until they had destroyed themselves. This behavior was eliminated. Later generation individuals that used the Energy Discharge did so with intervals between energy discharges. These intervals could include attempts at repairing damage or other benign or beneficial commands. The simple self-destructive sequence is no longer present.

5.2 Evolutionary Trends

These changes are supportive of the hypothesis that the bug programs would improve over time. The bugs in later generations were generally more fit than those in generation 0. This is true for all three experiments. Trends in the first and third experiment further support the hypothesis that the individuals were improving. The highest scores of the bugs in both species trended higher with time, as shown in Figure 2 with results from the third experiment, for interspecies competitions. Figure 2 also shows Species 1 as dominant over Species 2.

Figure 3 uses a 10-point moving average of the top scores for both species in intramural as well as interspecies competition. It is noteworthy that species 2 was actually making rapid gains early in the experiment before losing all

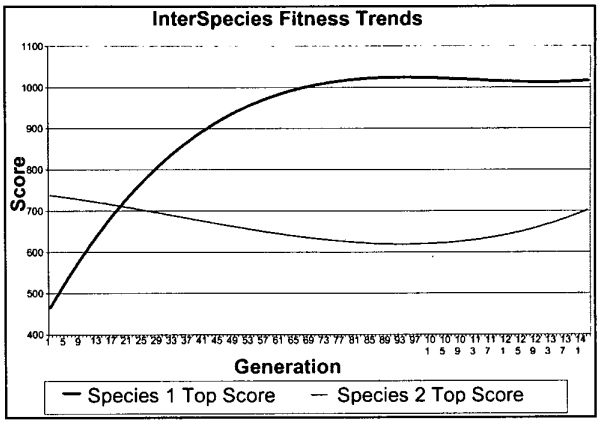

Figure 2. Interspecies Score Trends

the ground it had gained. This relapse is most likely attributable to the fact that not all individuals were guaranteed to compete against all others. It is also possible that the selection mechanism – allowing an equal number of new individuals for intramural and interspecies – magnified any deficiencies in the gene pool for species 2.

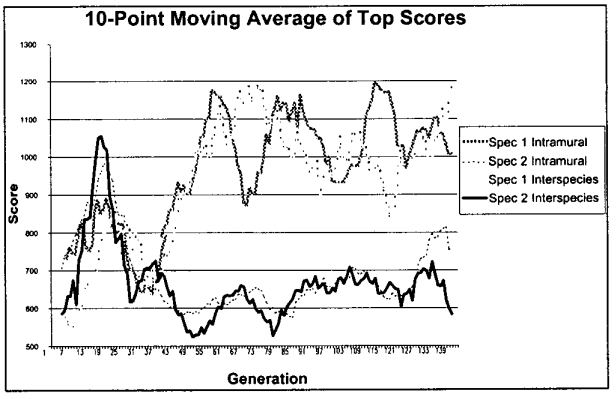

Figure 3. Moving Average of Top Scores

What is also of interest in Figure 3 is the mirrored peaks and valleys of intramural versus interspecies fitness that species 1 demonstrates. This suggests that bugs that evolved as competent in the intramural competition were less competent in the corresponding generation's interspecies tournament. This supports the motivation of running both kinds of tournaments in the hopes of evolving a generalized competitor that will compete well in both arenas (as seems to occur around generations 85 and 135).

By contrast, the intramural and interspecies tracks for species 2 are almost identical. This suggests a fairly poor homogenous population that has lost the valuable genetic material it had developed before generation 25. Lacking any vigorous individuals who might produce high-quality

offspring, mutation is insufficient to bring species 2 up to the levels achieved by species 1.

As was hypothesized, the most successful individuals in the experiment came from later generations. Specifically, bug 108 in species 1 (created in generation 23) won 17 out of the 120 interspecies tournaments (14%) that occurred after its creation. Somewhat better was bug 444 (created in generation 110), which won 5 out of the 34 interspecies tournaments that occurred after its creation (15% success). Bug 412 (created in generation 102) which won 7 interspecies tournaments of the 42 in which it could compete (17% success) achieved the best success ratio demonstrated so far. The success ratio of the remaining individuals of each species ranged from 3% to 0% (no wins). The behavior of these successful bugs was characterized by limited movement (single steps forward or backward, single turns) and aggressive firing of weapons.

5.3 Impact of Exogenous Fitness Function

In the second experiment, there was a complete tendency towards passivity. Both populations converged toward bugs that were coded differently but whose overall behavior was characterized by inactivity. The reason for this seem to be that although the endogenous fitness function offers substantial rewards for damaging an opponent, the likelihood of doing so when the driving program is behaving randomly is quite low. As a result, bugs conserved fuel and ammunition by staying in one place and not firing their weapons. This behavior corresponds to the "mediocre stable state" described in [FP94]. The simple fitness function failed to reward subgoals explicitly (e.g., directly or indirectly damaging enemies and collecting flags), thus encouraging passivity. The endogenous fitness function provides a view of the fitness landscape in which passivity is sufficiently rewarded and the active, aggressive behaviors in which we were interested would be extremely narrow – albeit tall – peaks.

The more elaborate exogenous fitness function explicitly rewarded the subgoals, thus succeeding in preventing a mediocre stable state and in encouraging the behavior in which we were interested. It "warped" the fitness landscape in such a way as to create a basin of attraction representing the active and aggressive behaviors we were interested in evolving.

6 SUMMARY

The data supports the basic hypothesis that the individual programs improve over time. The third experiment indicates that one species is becoming dominant over the other. We believe that a larger number of generations will reinforce this pattern. No bugs evolved that were capable of competing against human-coded programs. In particular, individuals have not yet demonstrated a robust connection between sensor input and action, nor have they grasped the substantial value of capturing flags after sustaining damage.

The use of an exogenous fitness function to drive the evolutionary process in a direction other than that indicated by the endogenous fitness function in this environment was demonstrated. The mechanisms underlying this effect are still under evaluation.

7 NEXT STEPS

The work that has been reported in this paper is preliminary. The experiments focused on evaluating complex environment for suitability in conducting A-Life experiments. In addition, the focus was on evaluating co-evolution with an exogenous fitness function, i.e., a fitness function applied as an outside influence. Most co-evolutionary research focuses on endogenous fitness since this is a more natural application of the environment and more closely fits natural evolution. The disadvantage to using endogenous fitness in A-Life studies is that it is difficult to assess the impact of the fitness landscape on evolutionary progress. Using an exogenous fitness function allows one to perform a more analytical assessment of the function's impact on evolution.

Experiments are currently under way to perform formal assessment of the impact of fitness on co-evolution. The two fitness functions evaluated for this paper provide initial indications of the sensitivity of different types of terms used in fitness on the ability to evolve valuable and interesting behaviors. We plan to focus on characterizing the nature of this sensitivity with the hope of developing guidelines for designing useful fitness functions. We also hope that the lessons learned from examining the exogenous functions will provide insight into the mechanisms inherent in endogenous fitness, thus helping to predict the types of factors that might be involved in natural evolution.

ACKNOWLEDGMENTS

We were aided by John Reder, the developer of AI Wars and Tony Dwyer, developer of the AI Wars Assistant. Both provided source code and programs to support this effort with no request for rewards other than a mention in the final paper.

Appendix: AI Wars Scoring

The point-value assignments that make up the game-assigned score are given in the following table.

Cause	Damage to Enemy (shielded/unshielded)	Damage to Self	Burn Rate Increase (s/u)	Points added to firing unit score vs. shielded/unshielded
Primary Weapon (PW) range 1	2 / 5	0/0	2/5	40/100
PW range 2	1 / 4	0/0	¼	20/80
PW range 3	0 / 3	0/0	0/3	0/60
PW range 4	0 / 2	0/0	0/2	0/40
PW range 5	0 / 1	0/0	0/1	0/20
Missile	70% / 90%	0/0**	6/9	300/500
Missile fragments	40% / 60%	40/60%	4/7	200/400
Mine	50% / 50%	0/0	5/5	0/0
Energy Discharge	3 / 3	1/1	3/3	100/100
Self Destruct	*	*	10	100/100
System Overload	0 / 0	50/50%	5/5	0/0

*Variable depending of remaining Energy Points

The percentage values in the table reflect the fact that missiles and mines actually cause this type of damage. An undamaged bug with 10 "life" points that steps on a mine will suffer 5 points of damage and increase its fuel burn rate by 5. The system overload line refers to the fact that a bug that takes a flag while at full strength (no damage) suffers system overload. Bugs do not receive points for doing damage to themselves. At the end of the battle, each surviving bug gets an additional 50 points for every life point remaining. Each "dead" bug loses an additional 50 points for every damage point it sustained over the amount needed to kill it. The score takes into consideration the final fuel burn rate for the bug. The burn rate increases when a bug is damaged through enemy action and possibly (5% probability) whenever using the attempt repairs command. Therefore the fuel burn rate is a persistent indication of damage sustained during a battle. A bug's final score equals the sum of all its other scoring measures minus its current fuel burn rate.

The exogenous fitness function applied these additional values:

Event	Reward / Penalty
Direct Missile Hit on an enemy	+1500
Taking a flag (when not at full strength)	+1500
Primary Weapon Hit on an enemy (any range)	+1000
Indirect Missile Hit (shrapnel) on an enemy	+750
Damaging enemy with Energy Discharge	+500
Firing Missile with Shields up	-500
Taking a flag (when at full strength)	-500
Stepping on a mine	-500

REFERENCES

[AP93] Angeline, P. and Pollack, J., 1993. "Competitive Environments Evolve Better Solutions for Complex Tasks." *Proceedings of the Fifth International Conference on Genetic Algorithms.*

[AP92] Angeline, P. and Pollack, J., 1992. "The Evolutionary Induction of Subroutines." *The Fourteenth Annual Conference of the Cognitive Science Society.*

[CM96] Cliff, D., and Miller, G., 1996. Co-Evolution of Pursuit and Evasion II: Simulation Methods and Results'. To appear in *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*

[Dwy99] Dwyer, T., WWW page at http://members.tripod.com/~clan_99/index.html

[FP94] Ficici, S., Pollack J., 1994. Challenges in Coevolutionary Learning: Arms Race Dynamics, Open-Endedness, and Mediocre Stable States

[Koz92] Koza, J., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. *MIT Press*

[Koz91] Koza, J., 1991. Genetic Evolution and Co-Evolution of Computer Programs. *Artificial Life II.*

[Koz92] Koza, J., 1992. Genetic Evolution and Co-Evolution of Game Strategies. *International Conference on Game Theory and Its Applications.*

[LS96] Luke, S. and L. Spector. 1996. Evolving Teamwork and Coordination with Genetic Programming. *Genetic Programming 1996: Proceedings of the First Annual Conference.*

[NI97] Nishimura, S., Ikegami, T., 1997. Emergence of Collective Strategies in Prey-Predator Game Model. *Artificial Life 3.*

[Ray91] Ray, T., 1991. Is it alive or is it GA. *Proceedings of the Fourth International Conference on Genetic Algorithms*

[Red98] Reder, J., AI Wars at http://ourworld.compuserve.com/homepages/John_Reder/ai.htm

[RB96] Rosin, C., Belew, R., 1996. New Methods for Competitive Co-evolution. *Evolutionary Computation 5:1*

[RB96] Rosin, C., Belew, R., 1996. A Competitive Approach to Game Learning. *Proceedings of the Ninth Annual ACM Conference on Computational Learning Theory.*

[She97] Sheppard, J., 1997. *Multi-Agent Reinforcement Learning in Markov Games.* Ph.D. Thesis, Department of Computer Science, The Johns Hopkins University.

[Spe96] Spector, L., 1996. Simultaneous Evolution of Programs and their Control Structures. *Advances in Genetic Programming 2.*