Using Evolutionary Strategies for the Real-Time Learning of Controllers for Autonomous Agents in Xpilot-AI

Gary B. Parker, *Member, IEEE*, and Michael H. Probst

Abstract - **Real-time learning is the process of an artificial intelligence agent learning behavior(s) at the same pace as it operates in the real world. Video games tend to be an excellent locale for testing real-time learning agents, as the action happens at real speeds with a good visual feedback mechanism, coupled with the possibility of comparing human performance to that of the agent's. In addition, players want to be competing against a consistently challenging opponent. This paper is a discussion of a controller for an agent in the space combat game Xpilot and the evolution of said controller using two different methods. The controller is a multilayer neural network, which controls all facets of the agent's behavior that are not created in the initial set-up. The neural network is evolved using 1-to-1 evolutionary strategies in one method and genetic algorithms in the other method. Using three independent trials per methodology, it was shown that evolutionary strategies learned faster, while genetic algorithms learned more consistently, leading to the idea that genetic algorithms may be superior when there is ample time before use, but evolutionary strategies are better when pressed for learning time as in real-time learning.**

I. Introduction

The problem that most artificial intelligence agents face, specifically scripted ones, is that rules are good when encountering expected situations, but they can be rather lacking when it comes to knowing what to do in unforeseen situations. This applies even to agents that learn behaviors in test situations before entering the real world, as they must make numerous attempts before there is an ingrained response. The reason that this is a problem is because of the chaotic nature of the world; it is unlikely for the exact same situation to occur, leading to a requirement for an agent to be able to learn to blur the lines separating rules.

It is for this reason that real-time learning is a highly important field in modern research. Real-time learning is the general idea that, while doing something, an agent is learning what about its approach works and what does not, meaning it is highly adaptable to changes in its environment. For example, learning in real-time would allow for a robot that can change its gait when it moves to different terrain. Learning in real-time would allow for an educational artificial intelligence to determine the appropriate pace to move a lesson along for the students. Learning in real-time would allow for interactive stories in virtual reality, according to Bates [1]. Learning in real-time basically means an agent is capable of dealing with situations that were not foreseen when it was given instructions to carry out, meaning it does not require nearly as much supervision.

Video games provide an excellent source of testing arenas for real-time learning agents. Scripted opponents provide regular opposition that that does not tire, making continuous learning possible. However, before agents can learn in real time, they need to be equipped with a learning system. Previous work on evolving game artificial intelligences includes Yannakakis and Hallam's work with Pac-Man [2], the work by Cole et al. on agents in Counter-Strike [3], and Priesterjahn et al.'s work on Quake [4]. In addition, work on games created and modified by the researchers in order to facilitate learning has been done by Miles and Louis [5], and neural networks have been used for learning by Stanley et all in the past [6].

Evolving neural networks with genetic algorithms has been shown to work in the game Xpilot in previous research [7]. However, maintaining a population of solutions that go through selection, crossover, and mutation for each generation is possibly too much overhead. Evolutionary strategies use mutation and only mutation in order to make changes from previous states. Given that a feed-forward neural net will tend to have everything interconnected, it makes sense to require many small changes as opposed to one big change which is what crossing over often does. The structure of a neural net and evolutionary strategies' use of repeated small changes seem to be perfectly compatible in theory, so it was decided to test them in practice in Xpilot. To establish a baseline it became necessary to create a genetic algorithm version of the controller for purposes of comparison. In this regard, we considered the research done by Yao on the various ways evolutionary algorithms and neural networks can be combined [8], and the research by Mandischer into evolutionary strategies being used as weight training for neural networks [9].

II. Xpilot-AI

Xpilot-AI [7] is a creation for researchers in artificial intelligence to have a good testing arena for control learning systems. The interactive Internet game Xpilot is the training battleground and the combatants are the various controllers created by researchers, the packaged AI that is part of the game itself, and the occasional human pilot who comes on the server.

The authors are in the Computer Science Department at Connecticut College, New London, CT 06320 USA (corresponding author: 860-439-5208; e-mail: parker@conncoll.edu).

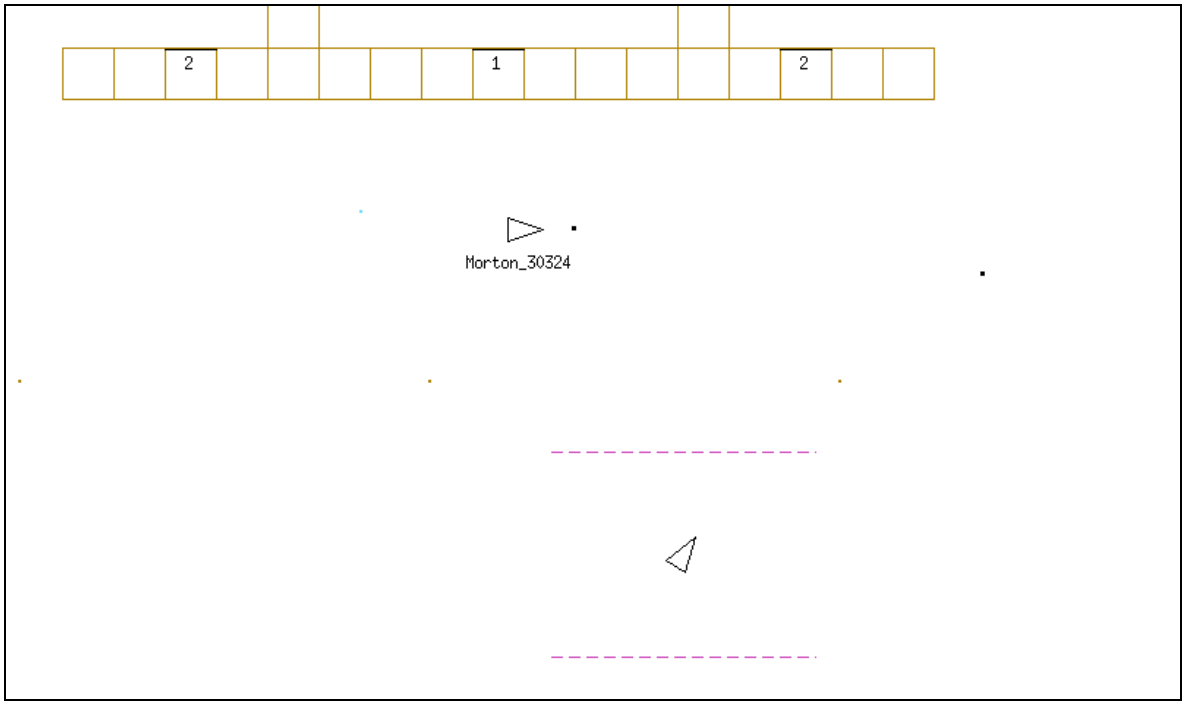

Fig. 1. A typical game of Xpilot in progress. The outlined triangles are the ships and the black dots are the shots. The gridded squares at the top are part of the stage, and the dashed lines above and below the lower of the two ships is part of the HUD (Heads-Up Display) to identify the player's ship. The colors are inverted from normal Xpilot graphics, as the normal game has a black background.

Xpilot is a multiplayer two-dimensional space combat game (Fig. 1). It is open-source, which allowed modifications to the interface to enable a scripted controller to log in and play without the need for human assistance issuing commands at the keyboard after telling it to start. It also facilitated the researchers' creations of functions for allowing the agent to control the ship (turn, thrust, and fire) and read in data from the area around the ship. The main structure of Xpilot is in two parts: the server and the client. Xpilot-AI modifies the client, so technically, any agent made using Xpilot-AI can easily connect to any server it has Internet access to, assuming the correct commands are used.

The game uses a consistent and relatively realistic, but modifiable physics setup, which specifically depends on the map being used. Since when a server is started is when a map is selected, this means the environment is not subject to change for a non-discernable reason. However, this does mean that any learned patterns on one map will not necessarily transfer well over to a different map.

Xpilot's controls are simple in that there are only three actions being controlled, at least in the basic format. However, good behaviors are complex and dogfights in the game are chaotic in the mathematical sense of the word (minute initial differences leads to large final differences). This is therefore an excellent learning environment for neural network controllers, as neural networks take in a number of inputs and provide a limited number of outputs.

III. Evolutionary Strategies

Evolutionary strategies (Rechenberg and further developed by Schwefel [10]) are good for evolving a population when the only thing known about a situation is that it requires using numbers to manipulate whatever the objective is, and that there is a known fitness function. Evolutionary strategies involve the use of an initial population, mutation, and a fitness function. In the general case, evolutionary strategies take the initial population, mutate its members to create at least as many children, and then test the children's fitness using the fitness function.

After that, one of two things happens. In normal evolutionary strategies with a larger population, the children with the highest fitness scores completely replace the original population, meaning there is a vulnerability to backsliding. In other versions, if there are children with higher fitness than members of the initial population, the weaker initial population members are removed and replaced with the children. If the children are inferior to the initial population, then the children are ignored. In either case, this is called an epoch. If new individuals made it into

the population, then it is a generation. For clarity, in the first type of evolutionary strategies, there is no distinction between an epoch and a generation, because the parents are replaced every epoch by the children. After that, the cycle begins again with the new initial population in the case of a successful generation, or with the unmodified old initial population in the other case.

In 1-to-1 evolutionary strategies, the type used in this project, the initial population is exactly one individual, and one child is created and tested each epoch, with only a successful child replacing the parent. This technique, while efficient in terms of guaranteeing progress barring flukes in the fitness function, is not necessarily as fast as other types of evolutionary strategies when the fitness can be quickly attained. However, given that the goal for real-time learning is to always have the best version being used, using a population, which will not always showcase the individual with the highest fitness, when it can be avoided seemed to be an unnecessary complication. Due to this, 1-to-1 was deemed the most appropriate choice.

IV. Genetic Algorithms

Genetic algorithms (Fraser [11]) work well at finding solutions in problems where not much is known about the solution barring that a fitness function exists and that the fitness function is not something which can be optimized easily. In many respects, it is similar to evolutionary strategies in where it can be applied, though it cannot be applied to every situation where an evolutionary strategies solution can be and vice versa. Genetic algorithms involve the use of an initial population, usually with a minimum size of fifty or greater, then using selection, cross over, and mutation to get the next generation of the population. Fitness of each individual in the population is determined by the fitness function.

Selection can be done on a basis of the best individuals mating with their closest counterparts and producing all the children or by stochastic selection, where selection is technically random but having a greater fitness increasing the likelihood of a specific individual being selected. In order to promote diversity, stochastic selection tends to be the preferred method. Crossing over is the idea of swapping part of one selected individual for the corresponding part of another selected individual. The goal of this is to propagate superior parts throughout the population, theoretically leading to a better population as a whole. Mutation is then done to the resulting crossed over individuals at a low probability, as mutation in genetic algorithms tends to reduce scores. It is still done, however, to help avoid a stagnant population.

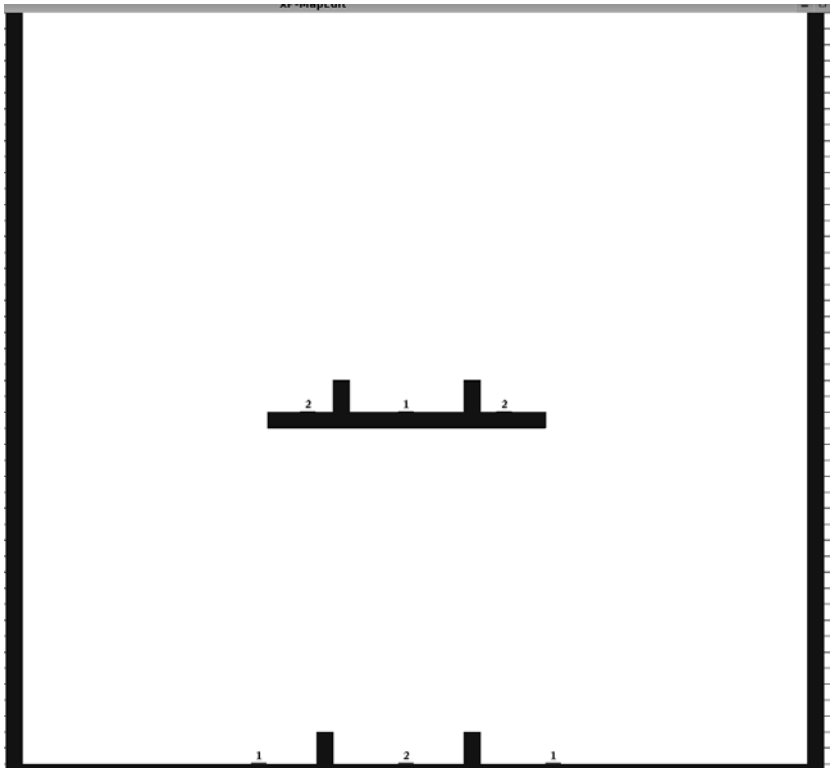

Fig. 2. Slightly modified image of the map, modifications include changing wireframe boxes to solid ones and color swapping from black to white, white to black, and bright blue to black.

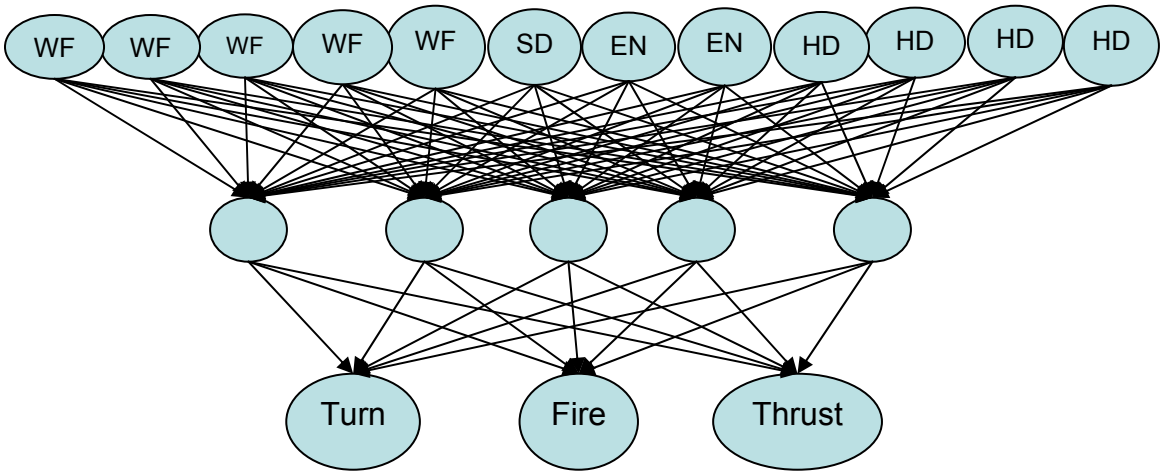

Fig. 3. Graphical representation of the controller. WF stands for Wall-Feeler, SD stands for Shot Danger, HD stands for Heading versus Track, and EN stands for Enemy Direction.

V. Method

A. Combat Situation

The goal in our combat situation in the game Xpilot is rather simple: eliminate the enemy before the enemy eliminates the agent. Although the goal is simple – its achievement is not. This is made more complicated by the fact that either ship can be annihilated by running into a wall, as well as its own bullets and the other ship. The main objective to be learned is a controller for the agent that allows it to avoid being destroyed before the opponent is destroyed. While the simplest way to do that is to fire at and hit the opponent, forcing it to collide with a wall because it was a choice between death by the wall and death by bullets also works.

The map created for learning was designed specifically for this purpose. Built to be as simple as possible while still providing a useful learning environment, the map uses basic physics rules and is very unforgiving of hitting other objects. To be exact, the map was designed to facilitate unbiased, one-on-one dogfights. As for the map's layout, the best way to describe it is a small platform in the center, with a large open space surrounding it, all boxed in. Figure 2 is an approximate screenshot that gives a good idea of the map's structure

The actual game uses a black background and white number markers, with wireframe boxes as opposed to solid ones; the figure is modified to make it easier to deal with visually and to save black ink. The number markers in the middle of the map and on the bottom edge are the different spawn points for the agents. The middle one of the three in the center and the outer two of the one on the bottom are the spawn points for the scripted opponent. The other three spawn points are the learning agent's spawn points. Each of the agents started anew from a random one of their three spawn points whenever either of them died.

The opponent for the agent was a relatively simple-minded, but still effective hand-coded agent by the name of Morton. It and its code are available on the Xpilot-AI (www.xpilot-AI.org) website for any who wish to see it. Morton is a better opponent than the server-controlled ships that come with the game, capable of competing with the average human player assuming the human does not abuse blind spots of the AI.

B. Controller Implementation

The neural network used as a controller had twelve input nodes, five hidden nodes, and three output nodes (Figure 3), one for each of the three commands of turn, thrust, and fire. The input nodes included five separate "wall-feelers," which detect how many frames it would take at the current velocity to smash into a wall. Each of these five goes out a different direction from the current velocity, with one pointing straight ahead, two going out at a fifteen-degree angle to either side from the velocity, and the last two being forty-five degrees away from the current velocity. The next node gives an estimate of how dangerous the closest shot to the ship is, which is determined by time to intercept and whether or not the bullet actually can hit the agent. The two indicators after that indicate where the enemy is relative to the ship's heading, one for port and one for starboard. When the opponent is to port, the starboard indicator is zero, and vice versa. All of these indicators are scaled from anywhere between zero and one, to account for varying conditions. The last four, which are not scaled, are indicators of which direction the nose is pointed relative to the current velocity, allowing the agent knowledge of how important the various wall-feelers are assuming the agent decides to thrust this

frame. To be specific, each of the four heading indicators has its own direction, which it checks to see if the nose is pointed in relative to the velocity. If the nose is pointing to that side (which does include overlap between different indicators), then the node returns one. Otherwise, it returns zero.

The evolutionary strategies for the weights of the neural network are done in a different manner than normal. Normally, the mutation is done with the same amount of mutation allowed per generation from beginning to end. This is not the case here. Earlier versions of the project used a static mutation amount. Ones with a small mutation amount took too long to figure out how to get out of their initial bad habits, resulting in little to no improvement in the two-day learning cycle. The versions with a large mutation amount had the problem that, after escaping the initial bad habits, they could not zero in on the right habits, and so ended up settling for mediocrity. Therefore, it was decided to try a variable approach. From the original range of mutation of 0.25 on the total range of weights from -1 to 1, after the first ten successful generations the range is cut in half, then reduced to a third of the original after the next five successful generations, with the pattern continuing from there. This allows for both the early advantage of a large mutation amount for rapid evolution, and the later advantage of precision of a small mutation amount. Prior research has been done on having a decreasing mutation rate for an evolutionary algorithm by Bäck and Schütz [12].

The disadvantage that this approach suffers from is a worsening of the already existing weakness to the "hill climbing to a local maximum" problem. This occurs when a learning agent, in the process of maximizing its fitness function, finds a local maximum that is not the global maximum. Due to the limited size of the possible perturbations at this point, it will be unable to leave said local maximum, even though it is inferior to the global maximum, because from its point of view, there is nothing better as an alternative. 1-to-1 evolutionary strategies are already rather susceptible to this problem due to their standard of only taking the best fitness. The decreasing mutation range method exacerbates the problem because, while a normal evolutionary strategies agent has a small chance at finding another peak in the fitness function within its mutation range at later levels of learning, this method results in an ever-decreasing range to do so.

The genetic algorithm variation of the control learner had identical methods for a controller, the only differences allowed were for the evolutionary process and the population requirement the technique imposes. This was done in order to make the comparison as simple as possible. A population of sixty-four individuals was used. Each individual was given control of the agent for the same time frame each epoch of the evolutionary strategies learner had, meaning a single generation of the genetic algorithm took the same amount of time as sixty-four epochs of the evolutionary strategies. The actual evolutionary process used stochastic selection with a five percent chance of mutation per weight, with the cross-over being done by switching entire groups of weights. Each actual mutation was done identically to the mutation of the evolutionary strategies, without the reduction of mutation range, so there was no set change value beyond being inside a certain range. A genetic algorithm already has enough modifications going on that changing the mutation value was felt to be unnecessary.

The best fitness function for learning was, as usual with evolutionary computation, difficult to determine. The end result, however, was rather successful. Each new mutation had a total initial life span of one thousand five hundred frames in order to prove itself better than the last. Given that the servers for the learning tests were running at fifty frames per second, which is between three and four times faster than the normal human speed for the game, this meant that each agent option had a maximum of thirty seconds.

Every time the agent killed its opponent, it gained a hundred points. Every time the agent died, with no preference given to how, it both lost a point and lost an additional twenty-five frames out of its remaining life span. This allowed for the control solutions that died more often to be eliminated faster, making it more likely that a robot with survival skills would be created. The agents all started with fifty points, in order to ensure all scores were positive so as to make stochastic selection for the genetic algorithm easier to manage.

VI. Results

Three tests for each type of controller ran for approximately eighty hours real-time. The evolutionary strategies controllers all had a minimum of over nine and a half thousand epochs while the genetic algorithms all had about one hundred fifty generations. For the evolutionary strategies, the fitness at each generation (epoch with an increase in fitness) and the resulting fitnesses for each test are plotted in Figure 4. In order to compare the ES results with the GA, ES epochs were converted to their GA generation equivalents. Information on the GA was stored at each 50 generations. Figure 5 shows the average of the three runs conducted for the two methods with the GA results shown for the best individual at the recorded generations and the average of the population at the recorded generations.

In every test for the genetic algorithms, most progress per generation, measured at the population average, was slow and steady, indicating a lack of fluke runs. The cases of a jumped fitness were checked to determine if a single individual in the population in question was just lucky or

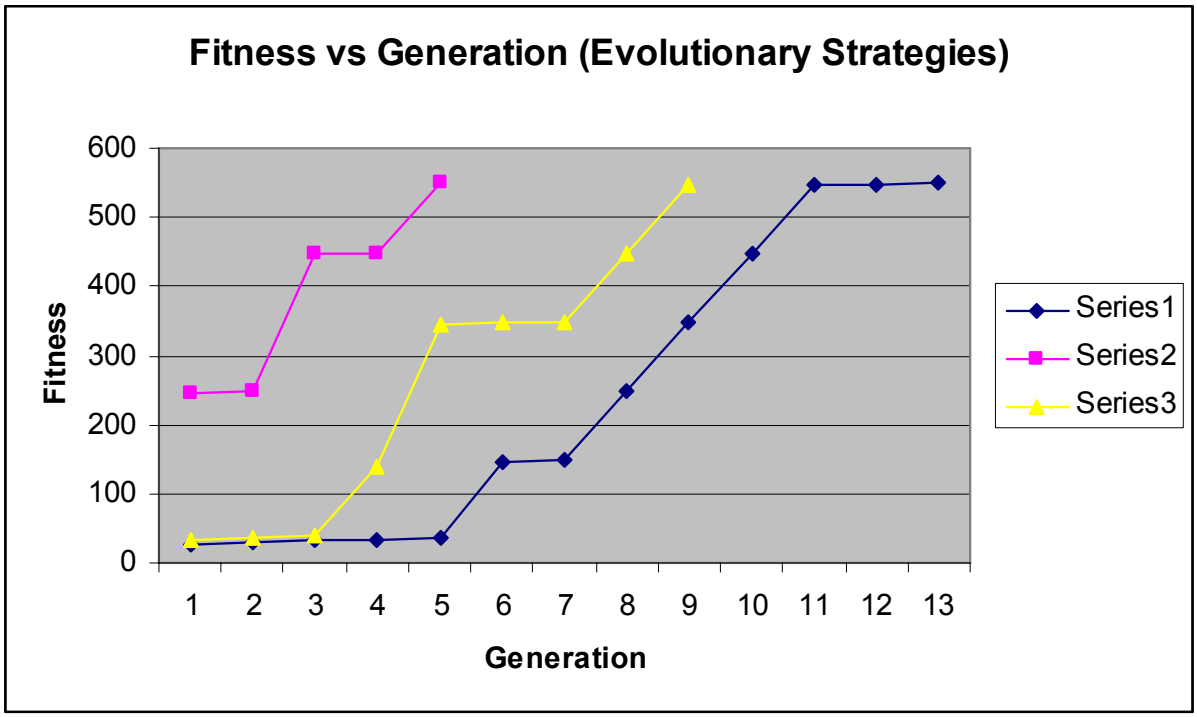

Fig. 4. Graph of fitness versus number of successful generations for the Evolutionary Strategies controller. The fitness function gives a hundred points for every kill, and loses both a point and fifty frames of lifespan for every death. Note that, even though the count of generations implies that series 2 finished far before series 1, this is an illusion. In fact, they all had the same amount of learning time, with series 3 actually reaching its final point first, followed by series 2 and then series 1. The other thing to notice is that, even when the graph may appear flat, it is not. A point's existence indicates an improvement over the previous point, though in the "flat" portions, this is an increase in survivability as opposed to number of kills.

unlucky, and for the most part, they ended as being slightly lucky, but not enough to throw off the learning. However, even towards the end of testing, there was steady evidence of progress towards a higher fitness for the population averages of the genetic algorithms, indicating they had not yet peaked.

The evolutionary strategies tests went quite well, all progressing quickly to a score where they were killing their opponent more than they were dying to anything during their half-minute runs. Though it is not visible on the graph, in each case, the majority of the generations were done in the first few hundred epochs, with fine-tuning occurring over a much greater period of time. Even the fine-tuning, however, tended to end before the genetic algorithms had finished their sixtieth generation. An example of this fine-tuning would be the final four points of series 1. The fourth to last point of series 1 happened on the thirty-second epoch, with a score of 448, indicating four kills of the opponent while only dying twice. The third to last point happened on the three thousand nine hundred twenty-sixth epoch, with a score of 546, indicating five kills with four deaths. The next point happened slightly over two hundred epochs later, with a score of 548, indicating an improvement of two fewer deaths, leaving the agent with five kills to two deaths. The final point of series 1, however, took an additional eighteen hundred epochs to come about. The improvement was to a score of 549, meaning that the agent killed its opponent five times while dying exactly once.

Due to the fitness function so heavily rewarding killing the opponent, it comes as no surprise that most behaviors were focused on hitting the enemy. What is interesting about this is the fact that, even the individuals that started off simply trying to stay alive changed to a more aggressive stance. Further thought on the subject lead to the realization that, in the scenario where testing was occurring, when one of the combatants died, the situation was reset. This meant that the best defense was a good offense for the agents. The fitness function fulfilled its job admirably in this case, as all three of the evolutionary strategies agents proceeded to end the testing with five kills to anywhere between one to four deaths from all sources.

The genetic algorithms agents had more varied controller results, with some individuals seeming to decide on a pacifistic stance at the start, meaning they refused to fire,

which led to them not having acceptable scores to pass on to the next generation, leading to populations that, while somewhat diverse, all had a habit of shooting first and aiming later. Some of the time this actually worked, by filling the space with enough shots the opponent was incapable of dodging all of them, but it was a costly few generations before aiming first became a dominant behavior among the population.

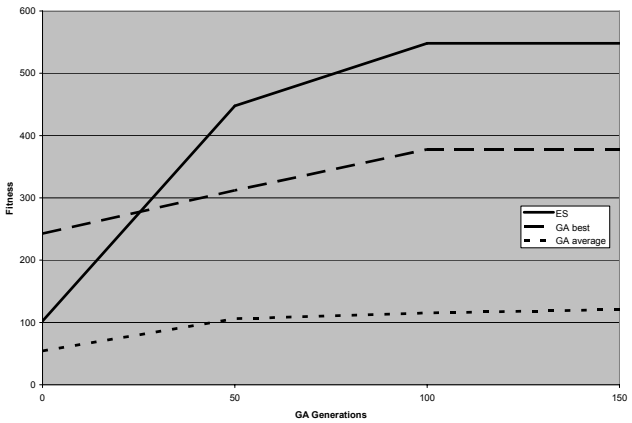

Fig. 5. Graph showing the average fitness for the ES and GA through the equivalent of 15 generations of GA evolution. The ES resulted in the best eventual performance. The learning curves for the population average and best individual are shown. The standard deviation at the end of training for the three sets of data are: ES: 2, GA best: 52, and GA average: 14.

The other noticeable thing about the genetic algorithms was that while, occasionally, the best members of the populations could approach or even top the fitness of evolutionary strategies agents from the same time, this ability was not passed completely on to their population. While the genetic algorithms agents most definitely were progressing, it was not at the same prodigious rate the evolutionary strategies agents did at the start. However, by the end of testing, the best individuals in the populations were competitive against Morton and the average of the population was still progressing.

VII. Conclusions

Our use of evolutionary strategies and genetic algorithms to learn the weights for a multilayer neural network controller in a real-time environment was successful. The evolutionary strategies agents made quick improvements; within a few hours were capable of beating their unchanging opponents. However, they reached a score plateau rather quickly, with few changes occurring once they had hit that ceiling. The genetic algorithms agents did not learn as quickly, however their rate of progress was more constant and still progressing, so we speculate that they could have ended up equaling if not exceeding the evolutionary strategies agents in due time. In either case, the resultant controller was competitive with the hand coded agent. Due to the quick learning aspect of evolutionary strategies, we find that they are a viable option for real-time learning in the Xpilot video game.

In future research, the ES learning system will be used to adapt to new playing strategies. An agent trained on Morton will be put in the arena with a new agent to see if it can adapt its play to remain competitive in real-time.

References

[1] Bates, J. The nature of character in interactive worlds and the oz project. Technical Report CMU-CS-92-200, School of Computer Science, Carnegie Mellon University, October 1992.

[2] Yannakakis, G. and Hallam, J. "Evolving Opponents for Interesting Interactive Computer Games," Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB'04); From Animals to Animats 8, 2004, pp 499-508.

[3] Cole, N., Louis, S., and Miles, C. "Using a Genetic Algorithm to Tune First-Person Shooter Bots," Proceedings of the International Congress on Evolutionary Computation 2004 (CEC'04), Portland, Oregon, 2004, pp 139–145.

[4] Priesterjahn, S., Kramer, O., Weimer, A., and Goebels, A. (2006). "Evolution of Human-Competitive Agents in Modern Computer Games." Proceedings of the 2006 IEEE Congress on Evolutionary Computation (ECE 2006), Vancouver, BC, Canada, July 2006.

[5] Miles, C. and Louis, S. (2006). "Towards the Co-Evolution of Influence Map Tree Based Strategy Games Players." Proceedings of the 2006 IEEE Symposium on Computational Intelligence and Games (CIG 2006).

[6] Stanley, K., Bryant, B., and Miikkulainen, R. (2005). "Evolving Neural Network Agents in the NERO Video Game." Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG 2005).

[7] Parker, G., Parker, M., and Johnson, S. (2005). "Evolving Autonomous Agent Control in the Xpilot Environment," Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005), Edinburgh, UK., September 2005.

[8] Yao, X., "Evolving artificial neural networks," *Proceedings of the IEEE*, 87(9):1423-1447, September 1999.

[9] Mandischer, M., "A Comparison of Evolutionary Strategies and Backpropagation for Neural Network Training," Neurocomputing 42(1-4): 87-117, January 2002.

[10] Back, T. and Hoffmeister, F. and Schwefel, H. (1991) A survey of evolution strategies. *Proceedings of the Fourth International Conference on Genetic Algorithms.*

[11] Fraser, Alex S. (1957). "Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction". *Australian Journal of Biological Sciences* 10: 484–491.

[12] Bäck, T. and Schütz, M., "Intelligent Mutation Rate Control in Canonical Genetic Algorithms," *Foundation of Intelligent Systems 9th International Symposium,* Zakopane, Poland, June 1996.