The Core: Evolving Autonomous Agent Control

Matt Parker
Computer Science
Indiana University
Bloomington, IN, USA
matparker@cs.indiana.edu

Gary B. Parker
Computer Science
Connecticut College
New London, CT 06320
parker@conncoll.edu

***Abstract* – The Core is a unique learning environment where agents compete to evolve controllers without the need of a fitness function. In this paper we use it with a cyclic genetic algorithm to evolve agents in the network game Xpilot, where the agents are engaged in space combat. The agents interact locally through tournament selection, crossover, and mutation to produce offspring in the evolution of controllers. The system is highly parallel, can be easily distributed among a network of computers, and has an element of simple co-evolution as the environment (population of agents) evolves to continually challenge individual agents evolving in the environment.**

I. Introduction

COMPUTER games can be used for experimentation with learning in autonomous agents since they provide an environment that is complex and challenging, often modeling relevant characteristics of the physical world. The computer game Xpilot offers several levels of complexity while requiring minimal in-game graphics. Xpilot is a well-established 2D internet game that was very popular with Unix users before the current age of 3D internet games came into existence. It is still in wide use today with several hosts sponsoring ongoing games. The user, with internet access and an installed free client program, can join a selection of arenas set with a variety of objectives and environmental conditions. The user controls a spaceship (represented as a triangle) that can engage in combat (through shooting bullets) with other ships. At this point in our research our area of concentration is close combat. Learning the correct behaviors to successfully beat an opponent in this space simulation is very challenging. Human controllers come in a variety of skill levels and experience. Our eventual goal is to have our autonomous agents learn to compete with humans in the online world of Xpilot.

Several researchers have used evolutionary computation for learning in games. Most of this work has been done on board games [1,2,3] and thought games, such as the prisoner's dilemma problem [4]. Research has also been done on learning controllers for interactive computer games such as Counter-Strike [5], a first person shooter game, Pac-Man [6], and Tron [7]. Parker et al [8] introduced the use of Xpilot for the learning of complex control behaviors for autonomous agents. Modifications were made that allowed researchers to write programs that get information about the environment and reply with control responses generated by an artificially intelligent agent. The system was tested by using a standard genetic algorithm (GA) to learn a simple neural network controller [8] and a rule-based controller [9] and by using a cyclic genetic algorithm (CGA) [10,11] to evolve a multi-loop control program [12]. These systems worked well for learning autonomous control, but required programmer involvement in the form of the creation of combat opponents and appropriate fitness functions.

The Core, a highly parallel learning system where agents co-evolve within a common environment without need of a fitness function was introduced and used with a modified CGA to evolve agent controllers [13]. Although successful, the controller was overly complicated due to the modifications made to the CGA and the learning system required altering the game play to reduce wall collisions. In this paper, we discuss the Core further, add a new parameter called the "age of adolescence" to avoid altering game play, and show that successful controllers can be evolved without the added modifications to the CGA.

II. Cyclic Genetic Algorithm

The original CGA [10] was developed to learn a single loop control program for locomotion in hexapod robots. It is a modification of the standard GA in that the genes (logical groupings of bits) represent tasks to be completed in a set amount of time as opposed to traits of the solution. Using a CGA, single-loop control programs with variable lengths could be evolved. In further research [11], the CGA was expanded to be capable of evolving multiple loop programs (allowing conditionals) and was employed to learn the control program for a robot with light and touch sensors. Controllers for agents in the Core need to be reactive in that control will change when events warrant (bullet approaching) yet have sufficient structure to allow more than one move in response to a stimuli (such as develop a circling pattern with an approaching enemy ship). CGAs can provide this level of control since they can be used to learn multi-loop programs. In this light we determined to use CGAs and developed one that was, in effect, a finite state machine (each conditional had a specific loop associated with it) with multiple instructions at each node. Although successful, we felt that this type of controller was too restrictive and did not utilize the full capabilities of the CGA.

For the work done in this paper, we use a multi-loop CGA that has conditionals with learned jump locations. The chromosome used (Fig. 1) is divided into 16 sections, which represent loops; each loop has 8 genes, which

represent the instructions; each instruction has 9 bits. Two types of instructions are used. One type describes an action to be taken by the Xpilot agent in the next frame and one describes whether to jump to a new loop. An action instruction determines how the agent should turn, whether or not it should thrust, and whether or not it should shoot. A single bit enabled for each indicates whether or not to thrust or to shoot. For turning, three bits are devoted to the quantity of turn and three to where to turn, such as to the nearest ship, away from the nearest wall, towards the most dangerous bullet, etc. A jump instruction consists of a four bit number corresponding to a conditional, such as "if enemy distance < 100" or "if self velocity > 20", and a four bit number representing to which of the sixteen loops the program should jump if the conditional is true. If it is false then the jump is ignored and the program goes on to the next instruction. If the end of a loop is reached, then the control loop goes back to the start of that loop. The program will continue through conditionals (jump genes) until an action gene is found and executed on that frame. On the next frame, the program begins again where it left off.

One of two types of crossover is applied at each mating. There is equal chance the crossover is either single point, cutting the chromosome only between two genes, or it uniform crossover across the entire chromosome. The mutation is a 1/300 chance of flipping per bit.

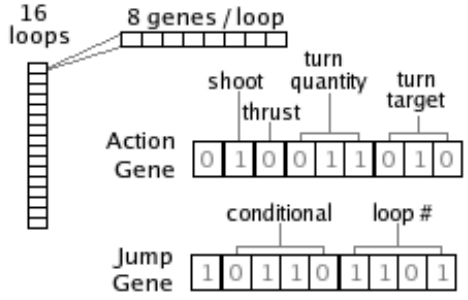

Fig. 1. Chromosome for the CGA, consisting of 16 loops, 8 genes per loop. Each gene can be either an action gene (0), which performs some combination of shoot, thrust, and turn, or a jump gene, which jumps to "loop #" if "conditional" is true.

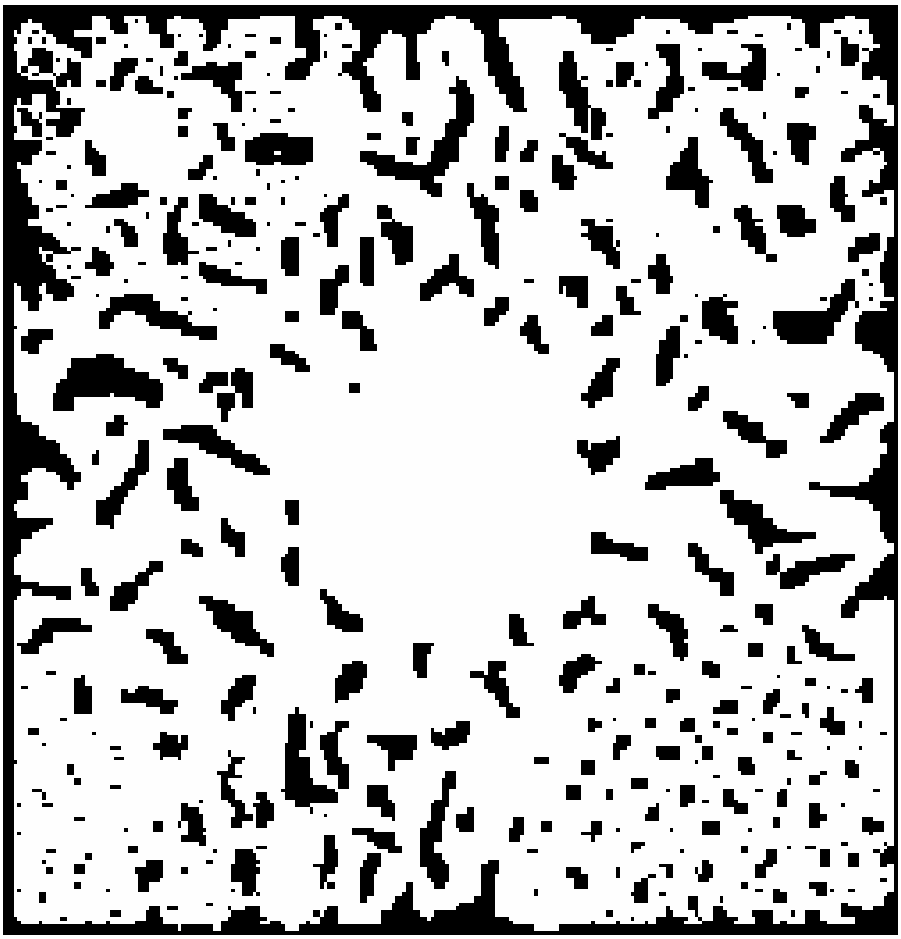

Fig. 2. The Core (black/white inverted). This is a map layout; the white area is free space and the dark areas are obstacles. Starting bases are scattered throughout each quadrant.

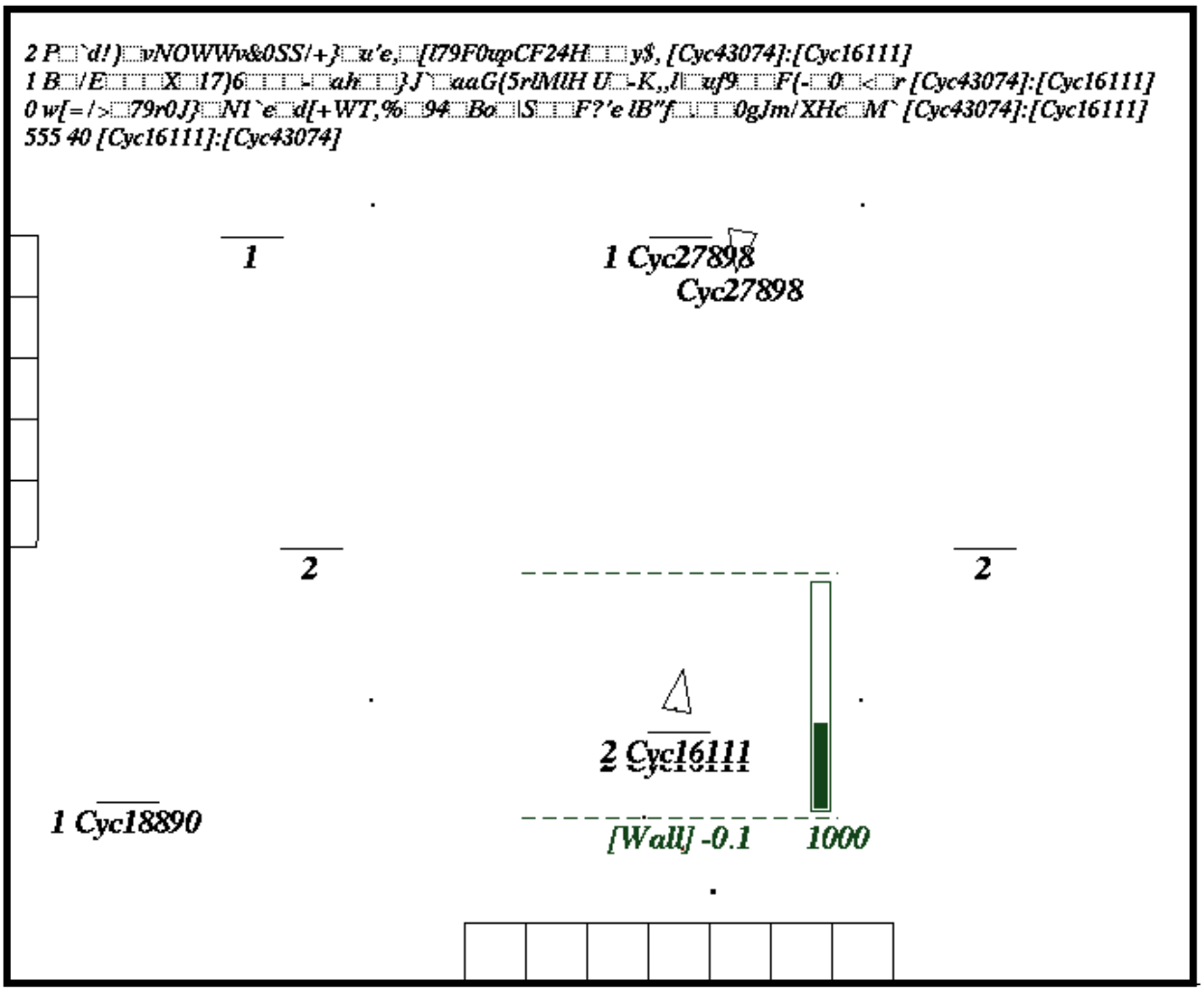

Fig. 3. Image (black/white inverted) of the combat view from an Xpilot game using the Core. It shows autonomous agent Cyc16111 fighting Cyc27898. The group of strange characters at the top of image is an encoded chromosome that was sent to Cyc16111 by its last killer, Cyc43074. In Xpilot, the empty space is black, the walls (blocks at bottom of screenshot) are blue, and ships are white triangles.

III. The Core

The Core (Figures 2 and 3) was developed as a means of learning in which reasonable agents would be provided as opponents. They needed to be progressively more skillful as our learning agents developed. Co-evolution between two populations was an option, but would require additional computation time to evolve the second population. The Core addresses this issue with no additional computation time by creating an environment where the other agents in the learning population are opponents who are constantly increasing in their capabilities. In this way, the environment as a whole is co-evolving with the individual agents who are learning in it. Another motivation was that we wanted to avoid developing a fitness function to achieve the desired behavior. The setup of the Core allows us to use how well the agent does in the environment as the fitness function. All of these factors combine to make a unique learning environment that produces excellent results. Another aspect of the Core is that it provides an environment where the agents can be tested in parallel. Xpilot sessions involve periods of combat that require time enough for the opponents to interact in positioning, firing bullets, and evading bullets. Although we determined a way to speed up the simulation to seven times its normal speed, any additional speedup could result in lost frames and controllers not suited for actual play. Running several combat episodes in parallel greatly speeds up the evolution. In addition, distributing the clients among several computers allows for additional speed increases in fitness computation. To maintain the desired parallelism, we wanted the evolution (selection, crossover, mutation) to be decentralized. This allowed the system to be asynchronous, which was important since some trials take ten times longer than others.

A. Characteristics and Related Research

The Core is a parallel system, which allows for any number of participating computers. The only limitation is the number of clients connected to one server, which is placed on the system by the hardware and operating system

on which the Xpilot server runs. However, as hardware becomes faster, more clients will be able to connect (currently we connect about 120). Xpilot servers are designed to accommodate a wide variety of capabilities so the client computers do not have to be equivalent in capabilities. The computer capability does dictate the number of clients that can be run concurrently on one computer, but since each client is an autonomous entity in itself, no coordination in the number of clients run on each computer is required. It is plausible, with a fast connection, that computers anywhere in the world could be contributing clients to the Core. Its parallelism would most closely identify it with a fine-grained parallel GA that is asynchronous [14,15]. It has a spatially-distributed population with parameters that can be adjusted to keep newly produced offspring in the local area or disperse them anywhere in the environment.

The Core is a co-evolving system, although there are not two distinct populations. Instead it has an environment that is constantly evolving to be more hostile to the individual agents within it. This is not competitive co-evolution in the classic sense. Significant work has been done in this field by several researchers [16,17,18,19,20]. One of the issues often discussed is the problem of having one population dominate the other [16,20]. This is not an issue with the Core since the body of agents is the co-evolving training environment for the individual agents and it cannot evolve to be constantly superior to the individual agents within it. In addition, no individual agent can dominate because if it evolves to be superior, its genes will eventually be spread throughout the population.

The Core has similarities with works in Alife where fitness is not predefined, but a property of the simulation [21,22,23]. It also has similarities to *embodied evolution* [24] used in evolutionary robotics. In this system, robots attain a virtual energy level that corresponds to their performance in completing the assigned task. They transmit their chromosomes with a rate proportional to their energy level and receiving robots accept it at a rate inversely proportional to their energy level. Upon acceptance, the receiving robot crosses it with its own chromosome producing a new controller, which overwrites its current controller. In the Core, when an agent kills another agent, its chromosome is passed to that agent's client. The client crosses the victor's chromosome with that of the defeated agent's chromosome, producing a new chromosome for the agent, which is sent back into the Core for testing. Tournament selection using only two opponents with the least fit being replaced by the recombination of the two was found to be a successful strategy (microbial method) by Harvey [25].

B. Description

The Core is a large virtual arena in which a massive population of agents can live, fight, die, and reproduce. A reasonably fast computer, with a reliable operating system, running a dedicated Xpilot server, is able to reliably host a very large number of Xpilot clients. For our dedicated Xpilot server, we modified the normal Xpilot-NG server, which accepts a maximum of 32 clients, to accept a virtually limitless number of clients. We also reduced to a minimum the number of informative messages that the server sends to its clients, such as those about who killed who, or about who switched teams, etc. because they are not important for the agents in the Core, and they use extra network bandwidth.

The environment in which the agents compete is a large 256x256 tiled map (Fig. 2). Tiles are about 3 times the area of a ship. Each of the four quadrants of the map has a unique terrain in order to represent multiple combat environments. The first quadrant (upper left) has fingers of walls with an open space in the center, the second (upper right) has fingers with no large space, the third (lower right) is cluttered with small asteroid-like chunks of walls, and the fourth (lower left) has just a few dots of walls, and mostly open space. The four quadrants collect towards the center where there is a large circle of empty space. Bases, where a player or agent may appear at a new life, are scattered in the center of each quadrant.

A closer view of an Xpilot session is shown in Fig. 3. The agent's ship is the triangle at the bottom of the screen with the label "Cyc16111". The ship above is another agent and they are engaged in combat with one another (the black dot below Cyc16111 is a bullet fired from Cyc27898). The boxes to the south and east are walls, and the short horizontal lines with numbers are possible starting bases.

The setup of the events of the Core is as follows. When a client first joins the Core, the client generates a random chromosome to control the agent. When the Core is first loaded, therefore, the entire population is made of agents with random chromosomes. Whenever an agent dies, its ship disappears from the map for about 32 frames, then it reappears in a random starting location, and the agent lives again. An agent may die four ways: collision with a wall at a high speed, collision with another agent, killed by its own shot, or shot by another agent. Of these four deaths, the first three simply result in the agent's death and reappearance in another location; its chromosome is unchanged. However, when it is shot by another agent, the dead agent sends a request to the killer agent, the killer sends its chromosome to the dead agent, and the dead agent performs crossover and mutation between its own chromosome and the chromosome of its killer. The messages requesting the chromosome and the chromosome itself are coded into ASCII and sent directly between the agents using the player-to-player text messaging system native to Xpilot (Fig. 3).

By this system, the agents who are more capable of killing other agents spread their genes to a larger number of agents than do those agents who are less capable of killing. In turn, the weaker agents who are killed slowly evolve to become as strong as their killers, increasing the abilities of the entire population. By mutation and crossover, new traits are formed in agents, and the traits that make the agent stronger are slowly spread across the entire population. In this way, the agents continually increase their abilities and sharpen one another.

C. *Age of Adolescence*

One of the goals for the final evolved behavior in this test is that the agents should be good at general combat, both in the Core, and in other Xpilot environments. A difficulty with regular genetic algorithms is that the evolved agent often finds ways to exploit the fitness function so that it receives a good fitness and yet does not do what was intended. The Core does not have a "fitness function", yet the setup of the map and rules of play have an enormous impact on the evolved behavior of the agents. For example, our first tests of the Core gave the agents almost no reason to avoid crashing into the walls. The agents developed the unintended strategy of thrusting straight ahead and shooting randomly. Any agents who might have been actually aiming were unlikely to hit these fast agents who whizzed past them and they were eventually killed off by a random and stray shots. The fast-flying agents always ended their lives crashing into the wall, but since there was no penalty, crashing into a wall merely transported them to a new location in the map, making it a good strategy. The entire population soon became fast-flying wall-smashers.

To fix this we tried mutating the chromosome of those who smashed into a wall, but the mutations were usually not beneficial and only increased the undesirable behavior. In another attempt [13], we penalized a wall-smasher upon rebirth, such that it was forced to wait 20 frames of game play as a vulnerable target, unable to move or control itself. This worked well, but with the negative aspect of changing the rules of the game play.

For the test reported in this paper we handle the wall-smashing by giving the agents an "age of adolescence" of 40 frames; a minimum amount of time that an agent must live before it is able to send its chromosome to other ships. This helps to make the wall-smashers less potent, as most of their kills in their short lives occur within 40 frames. Further penalization of wall-smashing, while it may more quickly reduce the behavior, evolves overly-cautious bots who are unlikely to use their thrust for any purpose. With this method, wall-smashing still occasionally flared up amongst the population, and was sometimes very prominent, but there evolved many ships who were able to both thrust and aim effectively, and these eventually dominated.

IV. Results

Because there is no specific fitness function in the Core, it is difficult to measure the success of the evolution. Visually it is easy to see that the agents have from their ignorant beginnings drastically improved and developed simple and effective strategies. However, their success cannot be measured by their time alive nor by their number of kills in the Core because as an individual agent evolves and increases its ability to kill, so do its competitors increase their ability to kill it and to avoid being killed.

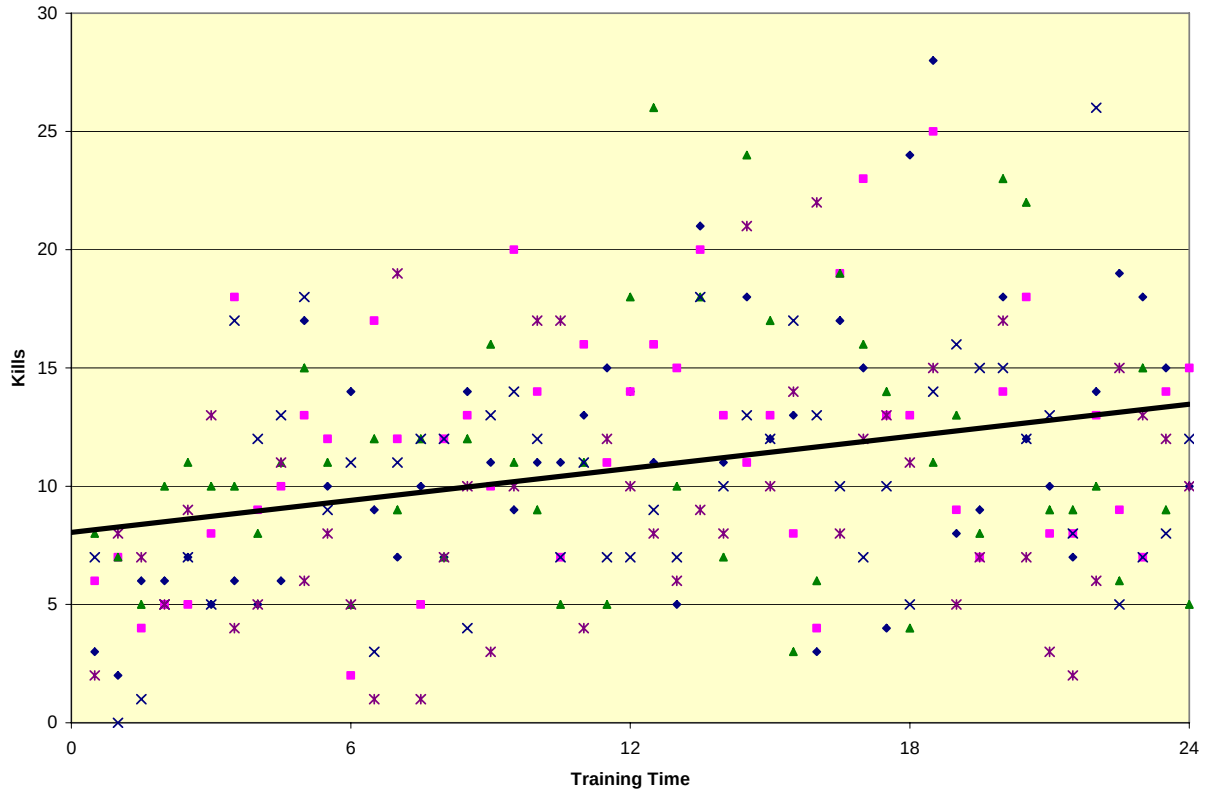

Fig. 4. Graph showing the fitness growth of 5 agents evolving for 24 hours in the Core with a least squares fitness line.

The fitness of the agents from the Core must therefore be judged by their ability to fight some standard opponent, whose behavior remains constant over time. However, care must be taken when placing an agent from the Core into some new type of environment. In the Core the agents learn to fight against multiple opponents and around rough terrain. Placing them into a simple square arena verses a specific bot does not adequately measure the increase in fitness over time, as behaviors that worked well in the Core may not work in that specific setting.

During the actual test of the Core, each agent saved its chromosome to file every 30 minutes. The Core ran in a stable condition for 24 hours, so each agent saved a history containing 48 of its chromosomes over the span of the test. A miniature map consisting of a portion of the upper left quadrant of the original Core map was developed to perform the test. It was populated with 10 enemies; each of which was controlled by a chromosome selected from a randomly chosen agent's chromosome history from the Core. These 10 enemies were chosen from the chromosome history in such a way that the full range of evolved chromosomes was represented; representatives from the first and least talented chromosome to the chromosome saved at the 24th hour. Five agents from the Core were randomly selected to test them in the miniature map. Each agent progressed through the history of all 48 of its chromosomes. Each chromosome controlled the agent's ship until it had died ten times from enemy fire. Thus, the agents could crash into a wall without it counting as a death. The total number of kills that the agent achieved while being controlled by a chromosome was recorded so that the improvement in the ability to kill could be seen over the span of the 48 saved chromosomes. To closely simulate the conditions in the actual Core, the kills that were achieved before the agent had reached the "age of adolescence" were not counted.

Fig. 4 shows the plot points for the fitness of each of the five agents tested, as well as a least squares trend line. The average fitness shows an overall increasing trend, while there are many dips in the individual fitnesses. An explanation for this is that the agents saved their chromosomes strictly every 30 minutes without regard to the health of that particular chromosome. The saved chromosome at that point was sometimes a poor one that just happened to be in play at the time of the save. However, since the good chromosomes were usually alive longer, there was a greater chance they would be saved, and therefore the trend of the five individuals tested is a definite increase in health and skill.

A very enjoyable aspect of the Core is watching the actual evolution of the agents as they play on screen. If a new agent is connected to a mature Core server, its evolution and behavioral assimilation into the population can be viewed over a span of a few minutes. When the agent first joins, its behavior is usually sporadic. It does not aim at other bots, sometimes it does not shoot, and usually it smashes into walls rather quickly after coming to life. It flies around like this until some mature agent finally shoots it. Then its behavior becomes some strange mix between the two, sometimes seeming to aim, and other times flying wildly. Each time the agent is killed, it becomes more and more like everyone else in the Core, until finally it seems to be fully assimilated.

Sometimes a new strategy is developed and it spreads like a plague across the population. For this test, after a few hours the Core was dominated by reckless wall-smashers. On the radar, which is a small view of the map, all the little dots representing the enemy ships could be seen rushing madly into the walls. Then, from some fortunate mutation or crossover, an agent learned to aim at the other ships and to not thrust. The agent playing on screen was one of the first to be turned into an aimer. It floated around, killing many of the wall-smashers, and being killed by many, but not receiving their chromosome because they had not yet reached the “age of adolescence.” It could be seen on the radar, fewer and fewer ships were thrusting madly across the map, and in minutes the wall-smashers had become endangered and aimers dominated the Core.

V. Conclusions

In the challenging game of Xpilot, it is very difficult to successfully evolve a controller that will be fit in the wide variety of combat environments available. A CGA can evolve multi-loop control programs appropriate for this level of control. However, one of the major problems with its use is building a variable environment and populating it with hand-coded opponents that reliably test the agent. Creating a fitness function that adequately tests the large number of tasks in general combat in Xpilot is also difficult. Therefore, we use the Core, with its highly parallel setup, simple single population co-evolution, and lack of a need for a specific fitness function, it is ideal for evolving robust combat agents in Xpilot.

This test shows just some of the capabilities of the Core. By redesigning the rules of play, the map, and the clients, there are many different possibilities for evolution. Some plans for future work include: localized regeneration, where areas of the Core develop specialized agents who can mix at the borders of their area (this will allow differing combat strategies to emerge within their own niche); team play, where multiple species of agents combat against one another in the Core, fighting to become the dominant population; and goal oriented play, such as destroying an enemy base.

The idea of the simple evolution and parallelism of the Core is useful in applications where agents are competing to perform a task. These applications include the evolution of non-combat agents. An example might be in the development of a race-car controller. In a population racing cars, the dominant cars can send their chromosome to the less successful cars that they pass. The Core has been shown to be an interesting and reliable method for evolving robust agents in Xpilot. Future work will determine its general applicability and use in the study of autonomous agent evolution.

REFERENCES

[1] Fogel, D. *Blondie24: Playing at the Edge of AI*, Morgan Kaufmann Publishers, Inc., San Francisco, CA., 2002.

[2] Konidaris, G., Shell, D., and Oren, N. "Evolving Neural Networks for the Capture Game," *Proceedings of the SAICSIT Postgraduate Symposium*, Port Elizabeth, South Africa, September 2002.

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

[4] Hingston, P. and Kendall, G. "Learning versus Evolution in Iterated Prisoner's Dilemma," *Proceedings of the International Congress on Evolutionary Computation 2004 (CEC'04)*, Portland, Oregon, 20-23 June 2004, pp 364-372.

[5] 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.

[6] 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.

[7] Funes, P. and Pollack, J. "Measuring Progress in Coevolutionary Competition," From Animals to Animats 6: *Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*. 2000, pp 450-459.

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

[9] Parker, G., Doherty, T., and Parker, M. "Evolution and Prioritization of Survival Strategies for a Simulated Robot in Xpilot," *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005)*, Edinburgh, UK., September 2005.

[10] Parker, G. and Rawlins, G. "Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots**,"** *Proceedings of the World Automation Congress (WAC '96), Volume 3, Robotic and Manufacturing Systems.* May 1996.

[11] Parker, G. and Georgescu, R. **"**Using Cyclic Genetic Algorithms to Evolve Multi-Loop Control Programs**,"** *The 2005 IEEE International Conference on Mechatronics and Automation (ICMA 2005)*, Niagara Falls, Ontario, Canada, July 2005.

[12] Parker, G., Doherty, T., and M. Parker (2006). "Generation of Unconstrained Looping Programs for Control of Xpilot Agents," *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006)*, July 2006, Vancouver, BC, Canada.

[13] Parker, M. and G. Parker (2006). "Learning Control for Xpilot Agents in the Core." *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006)*, Vancouver, BC, Canada.

[14] Cantu-Paz, E. "A Survey of Parallel Genetic Algorithms," Technical Report 97003, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana, Illinois, 1997.

[15] Tomassini, T. "Parallel and Distributed Evolutionary Algorithms: A Review," *Evolutionary Algorithms in Engineering and Computer Science*, pages 113–133. J. Wiley and Sons, Chichester,1999.

[16] Bongard J. and Lipson, H. "Nonlinear System Identification using Co-Evolution of Models and Tests," *IEEE Transactions on Evolutionary Computation*, 2004.

[17] de Jong, E. "The Maxsolve Algorithm for Coevolution." *Proceedings of the Genetic and Evolutionary Computation Conference*, 2005.

[18] Popovici, E. and De Jong, K. "Relationships between Internal and External Metrics in Co-evolution," *Proceedings of the Congress on Evolutionary Computation -- CEC-2005*.

[19] Rosin, C. and Belew, R. "New Methods for Competitive Coevolution." *Evolutionary Computation*, 1997, 5(1):1–29.

[20] Williams, N. and Mitchell, M. "Investigating the Success of Spatial Coevolutionary Learning," *Proceedings of the 2005 Genetic and Evolutionary Computation Conference, GECCO-2005* , New York: ACM Press, 523-530, 2005.

[21] Ventrella, J. "Attractiveness vs Efficiency (How Mate Preference Affects Locomotion in the Evolution of Artificial Swimming Organisms)," *Artificial Life VI*, MIT Press. 178-186, 1998.

[22] Werner, G. and Dyer, M. "Evolution of communication in artificial organisms." *Artificial Life II*, Addison-Wesley. 659-687, 1991.

[23] Thearling, K. and Ray, T. "Evolving Parallel Computation." *Complex Systems*, Volume 10, Number 3, 1997.

[24] Watson R., Ficici S., and Pollack J., "Embodied Evolution: Embodying an Evolutionary Algorithm in a Population of Robots". *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 335-342, 1999.

[25] Harvey I. "Artificial Evolution: A Continuing SAGA," In *Evolutionary Robotics: From Intelligent Robots to Artificial Life*, Takashi Gomi (ed.), *Proceedings of the 8th International Symposium on Evolutionary Robotics (ER2001)*. 2001.