10-2011

Investigating the Effects of Learning Speeds on Xpilot Agent Evolution

Gary Parker
Connecticut College, parker@conncoll.edu

Phil Fritzsche
Connecticut College, pfritzsche@gmail.com

Recommended Citation

Parker, G.; Fritzsche, P., "Investigating the effects of learning speeds on Xpilot agent evolution," Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on , vol., no., pp.2561,2566, 9-12 Oct. 2011 doi: 10.1109/ICSMC.2011.6084062

Investigating the Effects of Learning Speeds on Xpilot Agent Evolution

Keywords
Xpilot; Xpilot-AI; Punctuated Anytime Learning; Fitness Biasing; Genetic Algorithm; Real-time Learning; Video Game Learning; On-line learning; Autonomous system control

DOI10.1109/ICSMC.2011.6084062

Investigating the Effects of Learning Speeds on Xpilot Agent Evolution

Gary Parker and Phil Fritzsche
Computer Science
Connecticut College
New London, CT, USA
parker@conncoll.edu and pfritzsche@gmail.com

Abstract—**In this paper we present a comparison of the effects of varying play speeds on a genetic algorithm in the space combat game Xpilot. Xpilot-AI, an Xpilot add-on designed for testing learning systems, is used to evolve the controller for an Xpilot combat agent at varying frames per second to determine an optimal speed for learning. The controller is a rule-based system modified to work with a genetic algorithm that learns numeric parameters for the agent's rule base. The goal of this research is to increase the quality and speed of standard learning algorithms in Xpilot as well as determine a suitable speed for employing Punctuated Anytime Learning (PAL) in the Xpilot-AI environment. PAL is the learning component of an overall system of autonomous agent control with real-time learning.**

Keywords-Xpilot; Xpilot-AI; Punctuated Anytime Learning; Fitness Biasing; Genetic Algorithm; Real-time Learning; Video Game Learning; On-line learning; Autonomous system control

I. INTRODUCTION

The objective of this research is to analyze the effects of frame rates in using genetic algorithms for learning combat behavior in the Xpilot-AI game environment. An important area of autonomous agent learning is interactive games. Having the ability to learn autonomous and intelligent behavior in games is beneficial to those who develop and improve games and to researchers in other areas as well. Computer games serve as an important test bed for methods of evolutionary robotics. The real-life nature of the physics in these games allows for an easy translation to real world applications. Xpilot, with the help of the Xpilot-AI learning add-on, is an environment where evolutionary systems can be tested for learning game agent behavior and robot control.

Much work has gone into developing systems that use video game environments for the purpose of developing human-like agents through static and learning systems. When a population-based genetic algorithm is used for this learning, it is a benefit to be able to speed up the game's frames per second (FPS) rate to increase the speed of learning. In Xpilot, the normal speed of play is 16 FPS. In order to speed up learning in past research, Xpilot was run at 64 FPS as a genetic algorithm (GA) needed to perform many experiments in attempts to develop intelligent behavior. In one work, a GA was used to evolve parameters for a rule-based system [1]. The rule-based system-controlled agent was altered to allow for learning by GA. It evolved optimal numeric parameters for the rules of the control program. In addition, GAs have been used to evolve weights for neural network-controlled agents [2]. Specific abilities were learned individually. Once the agent had successfully learned how to optimally perform each task in isolation, the three resultant agents were combined together using a multi-layer neural network. In other research, a cyclic genetic algorithm was used to directly evolve a control program for an Xpilot combat agent [3].

Observationally, running at 64 FPS appeared to be the fastest the system could run without encountering adverse effects on the quality of learning. A few attempts were made at 128 FPS, but the agent's performance led researchers to believe it was not learning as well as at 64 FPS. It was determined that the higher frame rates were not effective. However, since experimental testing was not done, it was determined that tests were needed to analyze the effects of varying frame rates on Xpilot agent learning.

In an ideal situation, researchers would be able to run the game at considerably higher speeds, thus improving the speed of learning while making better use of the power of modern computers. Though a game learning at 64 FPS is a four-fold increase over the human-playable speed of 16 FPS, it can still take several hours for an agent to achieve near optimal behavior. At 64 FPS, we are not taking full advantage of the power of the computer, which runs at low capacity even with an Xpilot server and multiple client agents running. In this paper, we test a genetic algorithm evolving agents at a sequence of increasing FPS to experimentally determine the realistic break point for increased speeds in Xpilot. The intent is to determine if 64 FPS is the top speed where an agent can learn with acceptable degradation.

In addition to exploring the effects of FPS rate to find the maximum feasible rate for use of a genetic algorithm, this work will also help in the development of a method of real-time learning during game play. Intelligent opponents that appear to learn, improve, and adapt are a large part of what makes a video game enjoyable. Creating artificially intelligent systems that are capable of doing so in real-time is, as a result, not only one of the most difficult aspects of video game development but also one of the most important [4]. In the DEFCON computer game, researchers used decision-tree learning and case-based reasoning combined with simulated annealing methods in order to create human-like behavior for game agents [5]. Others combined evolutionary learning techniques with neural networks by adding layers and connections to the

network slowly while the game runs to allow their agent to learn increasingly complex behavior in real-time [6].

Real-time learning is an important aspect of learning systems for game agents as well as robots. It allows the system to continually learn and update as the agent performs its assigned tasks. In previous Xpilot research, real-time learning was attempted using evolutionary strategies [7]. Though the system was capable of real-time learning, its evolution was slower than through other methods due to its reliance on mutations of a chromosome to learn. As a result, it was determined to be ineffective. In other research [8], dynamic programming based reinforcement learning, specifically Q-learning, was used to implement real-time learning for Xpilot agents. Due to the nature of reinforcement learning, the tests were done in very simple environments. Q-learning requires an accurate model of its environment to be successful. As the Xpilot combat environment is highly complicated, this technique was applied only to a single agent in a simple scenario with no opponents. Though it was a successful implementation of real-time learning, it was determined that it lacked the scalability required of a robust algorithm. As the system's complexity increased, the agent's ability to adapt rapidly deteriorated.

The research reported in this paper will help in determining an acceptable maximum speed for Punctuated Anytime Learning (PAL). PAL, which was originally developed for evolutionary robotics [9], is the learning component of a system of autonomous agent control with real-time learning. In separate research, we are using PAL in an effort to find a method for learning Xpilot agent controllers in real-time, while engaged in combat with a human player. PAL was developed as a means of linking the offline learning, taking place in a simulation, with the actual robot. An agent using PAL as a learning system is split into two parts: the "real-world" agent and the "simulation" agent. The simulation agent performs learning on a model of the real-world environment. Periodically, the simulation agent communicates new and improved control programs to the real-world agent. At all times, the real-world agent takes the best of the control programs given to it by the simulation and uses that as its primary control program. In addition, when the simulation agent checks in with the real-world agent, the real-world agent returns information as to how well the new control programs worked in the real-world. The learning system then uses this information to improve the quality of its learning algorithm. In this way it is able to learn faster than the real-world agent while still maintaining accuracy and quality.

The maximum acceptable speed for PAL learning in Xpilot can be higher than for typical GA learning since PAL is designed to compensate for discrepancies in the simulation. Determining max PAL FPS will help us in the development of a real-time learning system for Xpilot to learn controllers for game agents. The learning will take place in a simulation at increased FPS. After the agent has optimized its control program, we will compare how well the agents perform at 16 FPS, the typical speed played by humans. The analysis reported in this paper will help determine what effects the varying FPS have on the quality and outcome of the learning processes, and help us determine the maximum reasonable FPS for a PAL system.

II. XPILOT-AI

Xpilot-AI is based on Xpilot (Figure 1), which is an open source multiplayer two-dimensional space combat game consisting of two main components: the server and the client. The server is used to configure settings for a game. For example, it can change the number of frames per second (FPS) in a game and the map being used. It is also the server's responsibility to track the players playing the game, their scores, and other information. The users control the client to play the game. Specific keystrokes allow the users to control their ship by thrusting, turning, and shooting. Xpilot contains relatively realistic physics. Agents explode if they run into a wall too fast, but will merely bounce off and lose some speed if they run into a wall at a very slow speed. Since the environment is in a space setting, an agent can glide without thrusting and still maintain constant velocity.

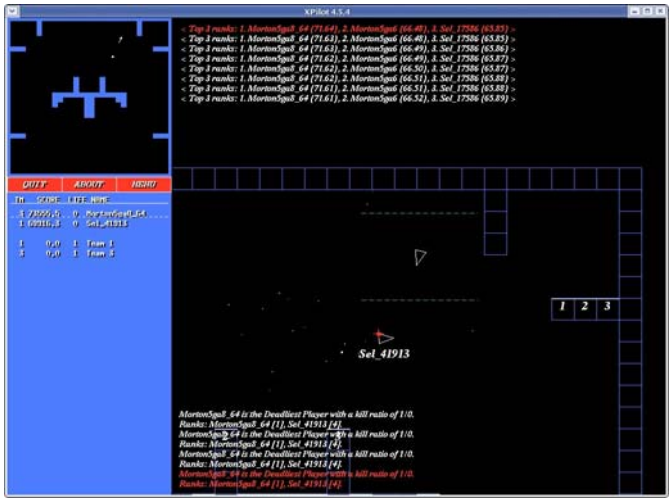

Figure 1. The Xpilot environment. Morton5ga8_64 is the player's ship in combat with Sel_41913. The triangles represent the ships, walls are in blue, and the numbers (1, 2, 3) show base locations. The top left box is the radar view that shows the entire map with dots representing the locations of the ships. Below this, the current score is displayed.

Xpilot-AI is an add-on to the Xpilot game that allows users to control Xpilot agents by writing scripts in any of a number of different programming languages. These scripted agents can play alongside other scripts, humans, or server-controlled robots on any standard Xpilot server. The scripts can be used to control the agents, which allows the researcher to use the environment to train the agents and develop systems to learn intelligent behavior. As a result, Xpilot-AI has become a powerful environment for researchers of artificial intelligence to test algorithms and learning systems.

The game provides an interesting opportunity to test autonomous agents in that it allows the researcher to run their scripts against other computer-controlled agents as well as human players. Seeing the agent in action in both scenarios

- span – the angle between the line from the agent's nose to a target location and the edge of the nearest wall. Used to determine if the agent is blocked from a bullet by a wall.
- offset_inc – indicates the increments used to determine the optimal direction to turn to avoid crashing into a wall
- same_spread – the difference allowed between the distances returned by two wall feelers which would result in considering them equal.
- wall_span1 – the angle off the ship's track used to feel for the closest wall.
- wall_span2 – the angle off the ship's track used to feel for the second closest wall.
- vd_bullet_dist – determines the bullet alert value required to consider the bullet very dangerous.
- d_bullet_dist – determines the bullet alert value required to consider the bullet dangerous.
- vd_dodge_bullet_angle – the angle the ship will turn away from a bullet considered very dangerous in order to dodge it.
- d_dodge_bullet_angle – the angle the ship will turn away from a bullet considered dangerous in order to dodge it.
- close_wall_speed – the speed of the ship in relation to the distance to the closest wall. Used to determine if the ship should take action to avoid the wall.
- medium_wall_speed – similar to close_wall_speed, but for walls that are farther from the agent.
- c_angle_before_thrust – the angle of the ship's heading away from the closest wall before the ship will thrust.
- m_angle_before_thrust – similar to c_angle_before_thrust, but used in a rule with lower priority
- wall_avoid_angle – how small the angle has to be between the ship's heading and its desired track to avoid a wall before it will thrust.
- screen_thrust_speed – if the ship's speed is lower than this and it is turning to attack an enemy on the screen, it will thrust.
- radar_no_thrust_speed – if the ship's speed is lower than this and it is turning to attack an enemy on radar, it will thrust.
- ship_error_to_shoot – the maximum angular difference between the desired aim direction and the ship's heading before it will shoot at an enemy on the screen.
- radar_error_to_shoot – the maximum angular difference between the desired aim direction and the ship's heading before it will shoot at an enemy on radar.
- wall_turn_angleR – the angle between the ship's track and heading that the ship turns to avoid colliding with a wall when responding to a right feeler indicating a wall that is too close.
- wall_turn_angleL – the angle between the ship's track and heading that the ship turns to avoid colliding with a wall when responding to a left feeler indicating a wall that is too close.
- wall_turn_angleB – the angle between the ship's track and heading that the ship turns to avoid colliding with a wall when responding to an equal distance from both walls.
- shoot_dir_rand – the angular range that the ship will use to randomly affect its direction to aim.

Figure 2. A list of the parameters from the control program that the genetic algorithm learned for the Xpilot combat agent

allows researchers to increase the quality of learning algorithms by better understanding how the scripts perform against all types of opponents.

III. Experiment

In order to determine the best FPS for learning, a standard genetic algorithm was used to do learning at different speeds. The genetic algorithm optimized parameters in the rules for a rule-based system used to control an Xpilot combat agent. These parameters are shown in Figure 2. This type of learning was shown to be successful in the past [1] with the model agent running at 64 FPS. To calculate the fitness for a given control program, each agent was allowed to engage in combat for a set time on an Xpilot server against a robust hand-coded rule-based agent named Sel. During its time in battle, the learning agent would receive one point of fitness for every frame it was alive and gain 1000 points of fitness for every time it killed its opponent. In addition, whenever it died, it would lose 1/6 of its total time.

The genetic algorithm was run on various Xpilot servers set at varying frames per second, starting at 16 FPS up through 1024 FPS. Each agent was allowed to learn for 115 generations. Once learning was completed, the best agents from the 50th and 100th generations were individually taken and placed against Sel while running on a 16 FPS server. This was done to determine if the results stayed consistent when playing on a normal server after learning. If the agents only appear to learn less while running at higher frame rates but still perform equally well as those that learned at slower frame rates when placed in matches at the same FPS, then Xpilot would only have the appearance of a degradation of quality at higher frame rates. On the other hand, if those agents that appear to learn poorly at higher frame rates also perform poorly when placed in lower frames per seconds, then it is clearer that the higher frame rates have a negative effect on the quality of learning.

IV. Results

Five test runs using five randomly generated populations at each of the tested frames per second were run for a total of 115 generations each. Tests included agents running at frames per seconds ranging from 16 through 1024. Figure 3 shows the fitness of the agents over time as they evolved at their respective frames per seconds. Based on the data, it can be observed that there was a great discrepancy in the quality of learning as the FPS changed. As the FPS increased, the amount learned by the agents decreased. For the agents running at 16 FPS, their fitnesses grew at an average of 18.6 per generation. Meanwhile, the agents running at 1024 FPS actually lost fitness by the end, shrinking at a rate of 0.213 fitness per generation on average as can be observed on the Figure 3 graph. The agents learning at 128 FPS and above all performed a minimal amount of learning.

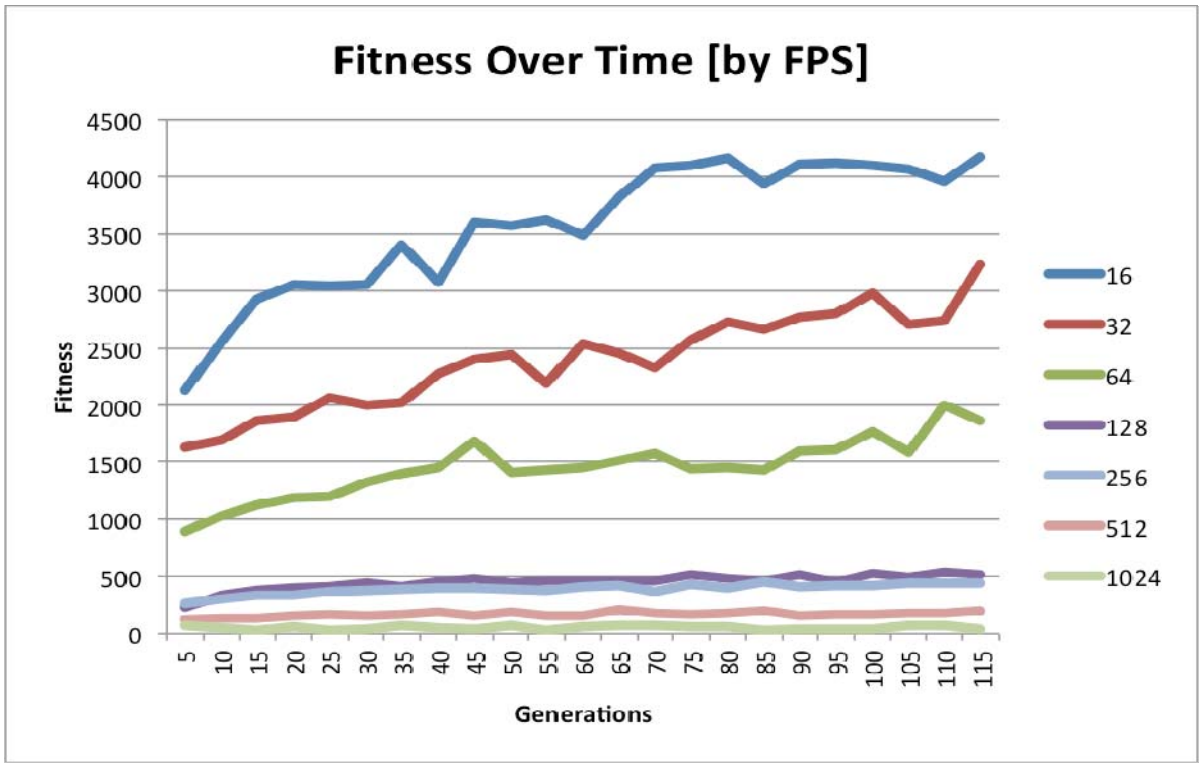

Figure 3. Growth curves for the GA learning with the game running at varying frames per second. The average of the population for five tests at each speed are shown.

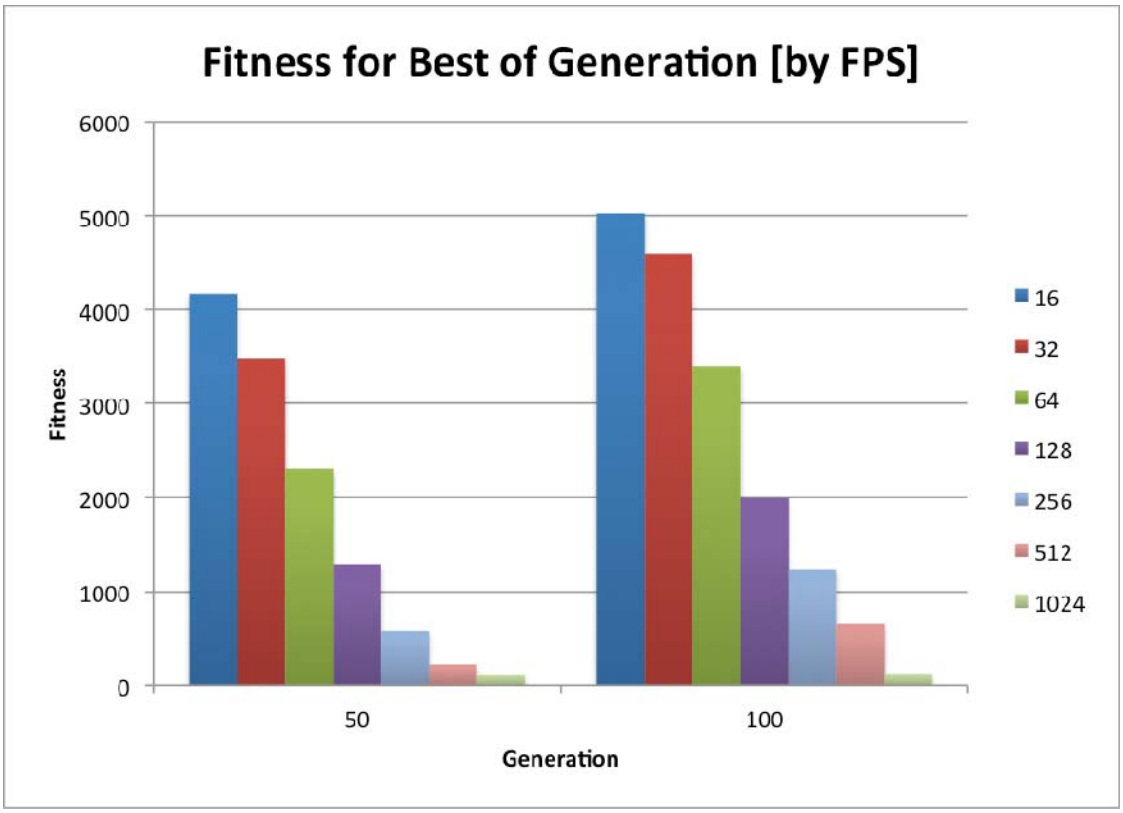

Figure 4. The average of the best individuals produced by the genetic algorithm after 50 and 100 generations. Each column represents the average of five agents, each of which were evaluated 30 times, for a total of 150 data points.

In addition, observations of the agents' behaviors were made from 16 to 64 FPS. They appeared normal and behaved intelligently. At speeds above 64 FPS, it appeared that agents were incapable of reacting properly to the events occurring around them. They would have a higher chance of a running directly into walls they would otherwise have avoided when running at slower speeds. If an enemy agent were to come into a learning agent's vicinity, it would be less likely to fire and often not seem to register that anything was different.

To further analyze the data, the best agents from each trial were analyzed after 50 and 100 generations. Each agent was evaluated in a test environment running at 16 FPS (normal game speed), regardless of the FPS the agent learned at, in the same way their fitnesses were evaluated during learning. Each agent from the five different trials was tested in 30 separate trials, yielding a total of 150 trials for each FPS learning speed at each of the two generations, 50 and 100. Figure 3 shows the averaged fitnesses of these trials. This data shows similar trends to that of the average fitnesses learned by the agents, with the agents learning at 16 FPS performed the best while the agents performing at 1024 FPS were clearly the worst. However, the best individual tests shown in Figure 4 show that the 128 through 512 FPS learning speeds may have merit. Even though they show less effective learning than the slower speeds, they still show some level of improvement. There is notable improvement from 50 to 100 generations in all cases, except at 1024 FPS. It's clear that a GA running at this speed produces little or no improvement in the Xpilot controller.

The above empirical data can also be confirmed by observation. At the beginning of the learning process for the lower FPS agents, the agents would regularly make obvious mistakes in combat. For example, some control programs would thrust while turning to avoid a wall and as a result run directly into the wall rather than avoid it or simply not turn sharply enough to avoid a bullet. However, by the end of the learning process, these mistakes were correct in the majority of the learned control programs. They exhibited intelligent behavior that was capable of defeating their opponent in the majority of situations. Meanwhile, the agents running at higher FPS generally displayed unintelligent behavior. For example, during the learning process, they would often thrust directly into the wall immediately upon spawning or aim incorrectly at opponents while firing. Even after learning was completed, these traits still remained strong in the majority, if not all, of the population.

Another consideration of the data from Figure 4 is a comparison of fitnesses from differing learning speeds at 50 and 100 generations (Figure 4). One can compare 16 FPS after 50 generations with 32 FPS after 100 generations (Figure 5), as the learning will take about the same time. With this in mind, consider 16 FPS at 50 generations versus 32 FPS at 100 generations. Since the fitness of 32 FPS at 100 generations is higher than 16 FPS at 50 generations, it can be concluded that 32 FPS is a better learning speed than 16 FPS. Now consider 32 FPS at 50 generations and 64 FPS at 100 generations (Figure 6). These fitnesses are close to the same with 32 FPS slightly better than 64 FPS. Considering 64 FPS at 50 generations and 128 FPS at 100 generations shows that 64 FPS has the clear advantage (Figure 7).

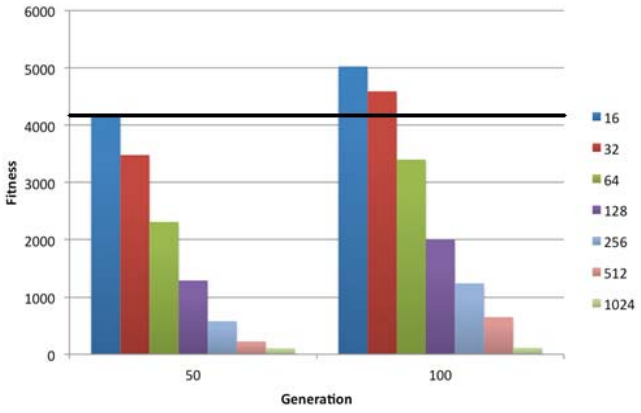

Figure 5. Comparison of GA run at 16 FPS versus 32 FPS. Since twice as many generations can be completed in the same amount of time, 100 generations at 32 FPS is equivalent in training time to 50 generations at 16 FPS. Running at 32 FPS produces better results.

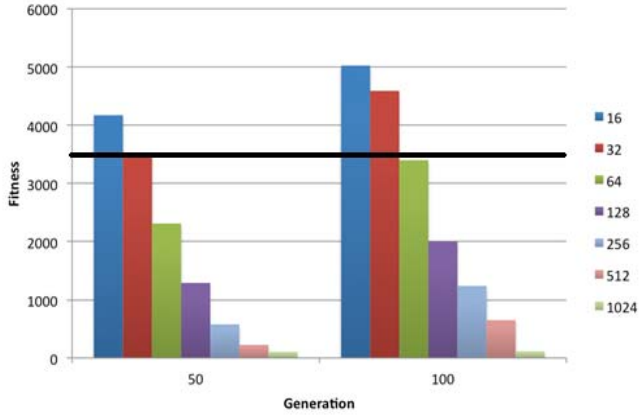

Figure 6. Comparison of GA run at 32 FPS versus 64 FPS. Since twice as many generations can be completed in the same amount of time, 100 generations at 64 FPS is equivalent in training time to 50 generations at 32 FPS. The two speeds are roughly equivalent, although 32 FPS produces slightly better results.

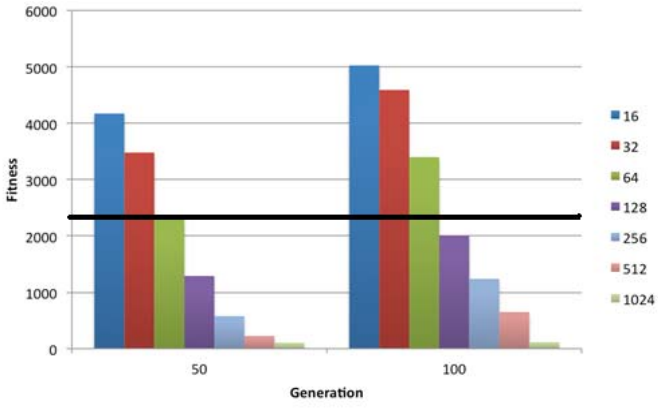

Figure 7. Comparison of GA run at 64 FPS versus 128 FPS. Since twice as many generations can be completed in the same amount of time, 100 generations at 128 FPS is equivalent in training time to 50 generations at 64 FPS. At this point the breakdown at higher speeds shows its effect. Running at 64 FPS produces better results.

V. CONCLUSIONS

Based on the collected data, it can be deduced that learning at a considerably higher FPS has a significant negative impact on the quality of learning produced. That said, running at 16 FPS is not necessarily the ideal solution either. Agents that learned at 16 FPS had the best results per generation, but those that learned at 32 FPS had the best results over time, since they were effectively learning at twice the speed. In Xpilot, learning algorithms often produce a large variation in their results due to the nature of the environment. As a result, even though the average fitness within the population run at 32 FPS is noticeably lower than that of the population run at 16 FPS, the average of the best agents is quite comparable. Given the increased of speed of learning due to the faster FPS, it makes sense to use 32 FPS for learning. In addition, since the 64 FPS versus 32 FPS results are nearly equal, 64 FPS is also a reasonable choice for learning. Speeds at 128 FPS and above are not recommended for standard GA learning.

Another conclusion that can be drawn from the data is what range of FPS is acceptable and useful for learning in PAL. When determining an appropriate speed for the simulation server, the ideal FPS would still learn but need not be adequate if learning on its own. Running Xpilot at 64 FPS would be a safe bet, but higher speeds are also possible. Agents are still able to learn and improve over time, but when compared to the 16 to 64 FPS agents, do not learn nearly as well or as fast. Although running PAL at 64 FPS would be sure to yield good results, depending on the goals of the research, running the simulation at FPS between 128 and 512 would also be acceptable given that they still improve and learn over time. They show a larger disconnect from the ideal FPS of 16 and as a result would not learn effectively on their own. However, if combined with PAL they could serve well as the simulation server speed. Any agent running at or above 1024 FPS, though, is ineffective. Algorithms run at this speed show neither intelligence nor improvement over time.

These tests will help in determining the maximum FPS that we can use to test the limits of a Punctuated Anytime Learning system. PAL for Xpilot-AI agents is now being tested with the simulation running at 128, 256, and 512 FPS. As the FPS the PAL learning system can handle increases, so does its ability to improve game agents during play. The stability of the PAL learning system at higher FPS is also an indicator of the PAL system's capabilities to deal with inaccuracies in robot models for actual robot real-time learning.

REFERENCES

[1] G. Parker and M.Parker, "Evolving parameters for Xpilot combat agents," Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Games (CIG 2007), Honolulu, HI, April 2007.

[2] G. Parker and M.Parker, "The evolution of multi-layer neural networks for the control of Xpilot agents," Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Games (CIG 2007), Honolulu, HI, April 2007.

[3] G. Parker and M. Parker, "Using a queue genetic algorithm to evolve Xpilot control strategies on a distributed system," Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006), Vancouver, BC, Canada, July 2006.

[4] G. Yannakakis and J. Hallam, "Evolving opponents for interesting interactive computer games," Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB 2004), pp. 499 – 508, 2004.

[5] R. Baumgarten, S. Colton, and M. Morris, "Combining AI methods for learning bots in a real-time strategy game," International Journal of Computer Games Technology, vol. 2009.

[6] K. Stanley, B. Bryant, I. Karpov, and R. Miikkulainen, "Real-time evolution of neural networks in the NERO video game," AAAI-06, pp. 1671-1674, Boston, MA, 2006.

[7] G. Parker and M. Probst, "Using evolutionary strategies for the real-time learning of controllers for autonomous agents in Xpilot-AI," Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010), Barcelona, Spain, July 2010.

[8] M. Allen, K. Dirmaier, and G.Parker, "Real-time AI in Xpilot using reinforcement learning," Proceedings of the 2010 World Automation Congress International Symposium on Intelligent Automation and Control (ISIAC 2010), Kobe, Japan, September 2010.

[9] G. Parker, "Punctuated Anytime Learning for Hexapod Gait Generation," Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002), EPFL, Switzerland, October 2002.