# Varying Sample Sizes for the Co-Evolution of Heterogeneous Agents

Gary B. Parker and H. Joseph Blumenthal
Computer Science
Connecticut College
New London, CT 06320
parker@conncoll.edu and hjblu@conncoll.edu

***Abstract* - The evolution of a heterogeneous team is a complex problem. Evolving teams in a single population can retard the GA's ability to specialize emergent behavior, but co-evolution requires a system for evaluation at trial time. If too few combinations of partners are tested, the GA is unable to recognize fit agents; if too many agents are tested, the resultant computation time becomes excessive. We created a system based on punctuated anytime learning that only periodically tests samples of partner combinations to reduce computation time and tested a variety of sample sizes. In this paper, we present a successful method of varying the sample sizes, dependent on the level of fitness, using a box pushing task for comparison.**

## I. Introduction

The objective of our work is to define a method for evolving a team that simultaneously reduces computation time while maximizing accuracy. A powerful method for producing cooperative behavior is essential because of its potential for application in today's world. Some applications for teams include toxic waste cleanup and search and rescue teams. These tasks are often too dangerous for humans and therefore fertile ground for the employment of teams of robots. We chose a box pushing task to test the strength of our method because it represents a simplified task related to our method's possible applications. Evolving heterogeneous behavior for teams has been approached in several different ways.

Luke and Spector researched methods for increasing specialization and cooperation in team behavior [1]. The applied problem in their research was the Predator-Prey scenario. The simulated scenario was a terriodal (continuous) space where four agents representing "lions" chase a randomly moving "gazelle". Using the success rate of capture for comparison, Luke and Spector concluded it was advantageous to consider the whole team as a single GP individual. Though this method produced heterogeneous behavior, we believe evolving team members in separate populations will further increase specialization because the GA can concentrate on making the best individual to do the task. When a team is represented by a single GP individual it is hard to obtain an individual's fitness without it being overly influenced by the team members to which the agent is bound within the same chromosome. This issue highlights the greatest challenge of co-evolution using separate populations, how to pair teams at trial time. Random selection of teams could result in fit members going unrecognized by the GA because of a poor member of the team.

Potter and De Jong developed a method to address the co-evolution of separate populations [2]. Their method, referred to as cooperative co-evolutionary algorithms (CCAs), tests an individual's fitness by pairing it with a single member from each of the opposing populations. These chosen members from each of the opposing populations are the fittest members within their own population from the previous generation of training. Using this system Potter, Meeden, and Shultz co-evolved artificially intelligent agents to heard sheep into a corral [3]. Their results showed this to be a successful method of producing heterogeneous behavior. Even though the CCA proved to be an efficient system of co-evolution, it still limits an individual's fitness calculation to a single partner.

Wiegand, Liles, and De Jong, investigated which factors influenced co-evolution using different implementations of the CCA [4]. In their research they focused on collaborator selection issues. The first implementation tested (CCA-1) used the best individual for pairing at trial time, while the second (CCA-2) had varying collaborator pool size, the number of individuals chosen for comparison at trial time. They concluded that the most prominent factor for co-evolution was the collaborator pool size. Wiegand, Liles, and De Jong, also point out that as the pool size increases so does the computation required for generational training. Taking this concept to the extreme, the most accurate solution would be to test all members of a population against all other possible partners in the opposing populations. Using this method a problem involving $N$ populations with $I$ members in each population, would require $I^N$ comparisons for any generation of training. This level of computation is unacceptable.

In this paper, we discuss a method that maximizes accuracy while minimizing computation time. This method involves a periodic selection of a single individual from each population that represents the overall nature of its population. These selected individuals are referred to as *alpha individuals* and the generations of selection are called *punctuated generations*. At trial time, it is possible to attain the fitness of any individual by constructing a team using the alpha individuals from the opposing

populations. We continue to optimize our algorithm in terms of accuracy and reducing computation time by varying the spacing between punctuated generations and the collaborator pool size used for alpha selection. We choose a box-pushing task to show the success of our method.

## II. Box-Pushing Task

The task is designed to require the cooperative behavior of two hexapod robots. The robots, which start from one corner of an enclosed square area, are to walk to and push a box that is in the middle of the area to the opposite corner.

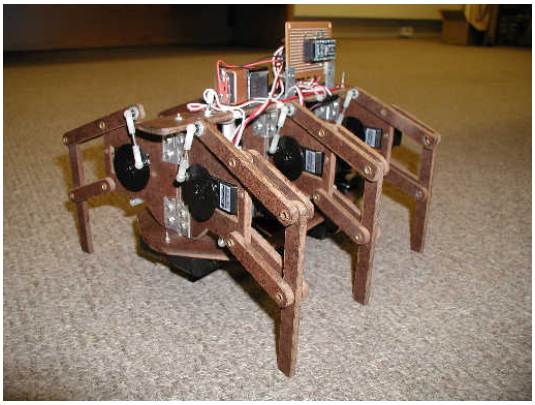

Figure 1. The ServoBot

The robots are modeled after an actual robot, which is a ServoBot. ServoBots are inexpensive hexapod robots constructed from masonite (a hard pressed wood). Each leg has two motorized servos, one oriented in a vertical capacity and the other oriented in a horizontal capacity, giving two degrees of freedom per leg (Figure 1). In order to control the motion of the robot, a BASIC Stamp II is mounted on the top of the ServoBot. The BASIC Stamp II is capable of individually addressing each of the twelve servo actuators to produce and sustain a gait cycle. A gait cycle is a timed and coordinated motion of the legs of a robot, such that the legs return to the positions from which they began. The BASIC Stamp is capable of storing a sequence of timed activations, which represent the simultaneous movement of all twelve servos.

Using a cyclic genetic algorithm (discussed in a later section), a gait cycle for a specific ServoBot can be evolved [5]. The optimal gait cycle found for the ServoBot modeled in our simulation was a tripod gait. A tripod gait maintains static stability while maximizing the walking speed for a hexapod robot. The right front and back and the left middle legs are down and moving back while the other three legs are lifting up as they move forward. Different degrees of turns were generated for the ServoBot by decreasing thrust of the legs on one side of the robot. If the left legs have less thrust than the right legs, result would be a left turn due to the drag created by the left legs throughout the duration of the gait cycle. The resultant turns were measured by tests on the actual robot giving 14 left and right turns, plus a no turn. These turns were measured in centimeters moved and degrees turned. These values were recorded and stored in a table with the addition of a “zero” value that corresponds to the robot not moving at all, giving a total of 32 total turns.

The scenario from which the task has been abstracted is a colony space in the Connecticut College Robotics Lab. The colony space is approximately an 8x8 ft area. In the colony space, the two ServoBot robots and a square cardboard box can be placed. The problem is designed for the pair to act cooperatively to force the box into the opposing corner from which the robots started. The tests done in simulation use agents that model actual robots present in the lab.

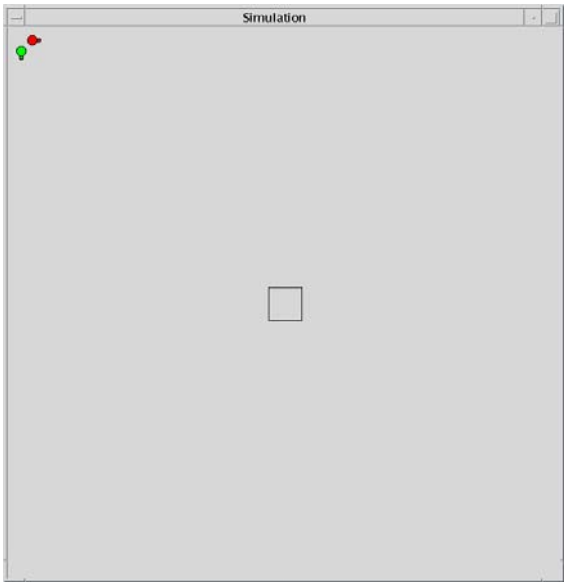


Figure 2. Simulation of the colony area with everything in its starting positions

The simulated environment used for evolving the agents was an abstraction of the colony space in the lab measuring 250x250 units. Both robots were represented as circles with a diameter of 6 units but the robots were treated as single points for the rules of contact with the box. The box was represented as a square measuring 18x18 units. In every trial the agents and the box were placed in consistent starting positions. In the simulated environment the point (0,0) is in the upper left hand corner of the area. The first robot was placed in the

location (10,5) and faced parallel to the x-axis, while the second robot started in the mirrored position (5,10). The box started in the center of the space at the point (125,125). In Figure 2 we see the starting position of the simulation where the first robot at (10,5) is shaded in dark grey, and the second robot at (5,10) is shaded in light grey.

Each robot's ability to move the box on its own (without aid from its partner) was affected by an endurance factor. This endurance factor is initially set to zero and increases by one with each consecutive non-aided push of the box. If F represents the would be full force of the robot's push acting singly, and E representing the endurance factor, the force the robot may apply to the box is given by the quotient $F/2^E$. This endurance factor reduces the distance the agent can push the box by a factor of two every gait cycle without its partner. When both robots push the box simultaneously their endurance factors are both reset to zero. Both robots move simultaneously and the simulation stops when either the robots have taken 100 steps or any one of the three (two robots or the box) moves out of the simulated area.

## III. Evolutionary Methods

The agents were evolved in two separate stages using incremental evolution. The first stage of evolution defined the robot's behavior before they first touched the box, while the second stage determined their actions subsequently. The learning was split into two increments because the first stage required no cooperation while the second stage did. In the first stage, each agent's behavior was evolved without its partner's presence in the simulation, each population evolved separately. For population A, the starting point was (10,5) facing down the x-axis; the fitness of an individual was either the value of the box's y coordinate position after the trail ended or zero if the individual failed to move the box positively in the y direction. For population B, starting at the mirrored position (5,10) facing down the y-axis, the individual's score was computed similarly except the agent was charged with moving the box positively in the x direction to receive a non-zero score.

In the second increment of evolution, both agents are placed in their respective starting positions for team evaluation. The fitness score of a team of agents is the product of the positive distances the box is moved in the x and y directions. Where $X_{final}$ and $Y_{final}$ represent the box's final coordinates and $X_{start}$ and $Y_{start}$ represent its starting position, the fitness of any given evaluation would be $((X_{final}- X_{start}) * ((Y_{final} - Y_{start}))$. The team is awarded a score of zero if the product of the coordinates is negative because they failed to advance the box towards target corner of the area in the x or y direction. Since $X_{start}$= 125, $Y_{start}$=125 and the largest that $X_{final}$ and $Y_{final}$ can be is 250, the maximum attainable fitness is $125^2$ or 15625.

### A. Cyclic Genetic Algorithms

Evolutionary methods using a cyclic genetic algorithm were used both for optimal gait generation and to coordinate the partners' behavior. The CGA is a variation of a traditional GA, where the genes of the chromosome represent tasks to be completed [6]. The tasks can be anything from a single action to a sub-cycle of actions. Using this method of representation, it is possible to break up a chromosome into multiple genes with each gene acting as a cycle. Each gene or sub-cycle can contain two parts, one part representing an action or set of actions, and the second part representing the number of times that action is to be executed. The genes can be arranged into repeated sequences and a chromosome can be arranged with single or multiple cycles or even the entire chromosome can become a cycle. In the case of multiple cycles, it is possible to switch from one to the other at any point. As in the case of the gait generation, the principle cycle of the CGA can have a head and a tail. These provide the CGA with pre- and post-cycle procedures, as a set of instructions to be executed directly before or after entering the cycle. These appended instructions allow for the legs to get in position to start a gait cycle or prepare for the next set of instructions following one. The CGA was perfectly fit for the incremental evolution of heterogeneous behavior. Each increment in the learning process was given its own cycle, forming a two-part chromosome. The switch from the first cycle to the second occurred when the robot touches the box, after the completion of the gait cycle's current execution. Each of the two cycles of the CGA chromosome has nine genes. Every gene contains two 5-bit numbers, one representing a gait cycle with 32 possible turns and the other representing the possible repetitions of that gait cycle. See figure 3 for a scheme representation of a two-part chromosome.

$(((T_1\ R_1)\ (T_2\ R_2)\ldots(T_9\ R_9))$ $\mathbf{((T_1\ R_1)\ (T_2\ R_2)\ldots(T_9\ R_9)))}$

Figure 3. Scheme representation of the CGA chromosome where T is a specific turn and R is the number of repetitions of that turn. The genes that appear in bold represent the second cycle controlling the movement of an agent after it first touches the box.

The first increment of learning produced two populations with sixty-four individuals each capable of reaching the box, one population evolved for the robot starting at (10,5) and the other starting at the mirrored position. For the second increment of learning the population was seeded with the first cycle evolved and random second cycles were appended for evolution. Individuals were selected stochastically for breeding based on their fitness score and standard operators were used for the CGAs.

### B. Punctuated Anytime Learning for Evolving A Team

The idea of anytime learning was originally introduced by Grefenstette and Ramsey [7] to allow for the continuous updating of a robot's internal model. A system of Punctuated Anytime Learning (PAL) was developed by Parker [8] to strengthen offline genetic algorithms by capitalizing on the dynamic nature of the anytime learning approach. Although PAL cannot provide continuous updates of the computer's models, it updates its model every *G* generations, resulting in a period of accelerated learning. The generations in which the model is updated are referred to as "punctuated" generations. When applied to a single GA, PAL updates the computer's model every G generations by running tests on the actual robot and uses these results for fitness biasing in the GA [9] or in the co-evolution of model parameters [10].

The concept of punctuated anytime learning was applied to co-evolving populations to form members of a team [11]. In this case, the updated information that each population receives is a more accurate representation of the overall nature of the other populations. For ease of explanation, assume that an experiment has two populations, population A and population B. At every $G^{th}$ generation all individuals in population A are tested against all individuals in population B. The purpose of this process is to find the most-fit individual from each population to evolve with members of the other population during normal learning. The chosen most fit individual from each population will be referred to as the *alpha individual*. Although the best method of evolution would be to select new alpha individuals for each generation, the process of alpha selection requires significant computation. Assuming there exists *I* individuals in each population and *N* populations, computer must perform $I^N$ trials for each generation at trial time. In order to avoid this level of computation, new alpha individuals are selected only at certain consistently spaced periods of time, every *G* generations. In non-punctuated generations, the alpha individuals selected from the last punctuated generations are paired with perspective team members in the other population for fitness evaluation. This method not only ensures consistency within a generation of training but it also decreases the total number of evaluations required to find an accurate solution. To test this method we ran five separate tests each for five thousand generations of training. All but one of the five tests reached a fitness of over 14,000 out of the possible 15625.The results showed that this method was successful in evolving heterogeneous agents.

### C. Sampling the Nature of Populations

After the initial success with the PAL method of co-evolution, it was concluded that a further reduction of computation time is required to extend the method to teams of three or more agents. We realized that it was possible to select an alpha individual while testing a sample of the entire population [12]. The number of individuals used for comparison to select an alpha individual is called the *sample size*. Assuming there exists *I* individuals in each of the *N* populations with a sample size of *S*, the computer must perform $N * (I * S^{N-1})$ trials for any given round of alpha selections which requires significantly less computation than our previous method requiring $I^N$ comparisons to select alphas.

We decided to test the sample sizes of 1, 2, 4, 8, 16, 32, and 64 to determine the relative strength of the different sample sizes for co-evolution [13]. To ensure that each sample size performed an equivalent number of fitness comparisons, we staggered the spacing of the punctuated generations such that sample 1 performed alpha selection every generation, sample 2 performed alpha selection every second generation, …, sample 64 performed alpha selection every sixty-fourth generation. In addition to these alpha evaluations, each generation also incurs 128 normal evaluations (evaluations during non-punctuated generations), one extra for each of the sixty-four members of each population.

The performance of the fittest combination of partners for each of the seven different sampling rates were recorded through 10240 alpha evaluations. Fitnessses were recorded at 0, 64, 128, 256, 512, 1024, 2048, 5120, and 10240 alpha evaluations. The tests showed that all seven sample sizes reached reasonably accurate solutions and the sample 64 ultimately reached the highest average fitness of any sample size by the $10240^{th}$ alpha evaluation. The lower sample sizes showed relatively superior performance in the earlier generations of training, but inferior performance in later generations. The opposite was true for the higher the sampling sizes where they were inferior to lower sizes in the earlier generations, but better in the later generations of training.

Although the lower sample sizes did not produce the best end result, they were better in the initial stages of learning. This quality of the smaller sample sizes shows their application for optimizing the method. They exhibit their accelerated growth up to the $128^{th}$ alpha evaluation. This is rather intuitive because by the sixty-fourth generation, the sample one has evolved with sixty-four different pairs of alpha individuals while the sample sixty-four has evolved with only one pair of alphas. The sample 4 continues its accelerated growth past the sample sizes of 1 and 2, because these sample sizes lack the ability to represent the true nature of a population with so few comparisons for alpha selection.

The significance of the results can be seen as helping to determine whether it is more important to have a more accurately selected or a more current alpha individual for training. For the box pushing task overall, the best sampling sizes are those which allow for sufficient tests in the early generations, yet enough of a sample to get the nature of the population in later generations.

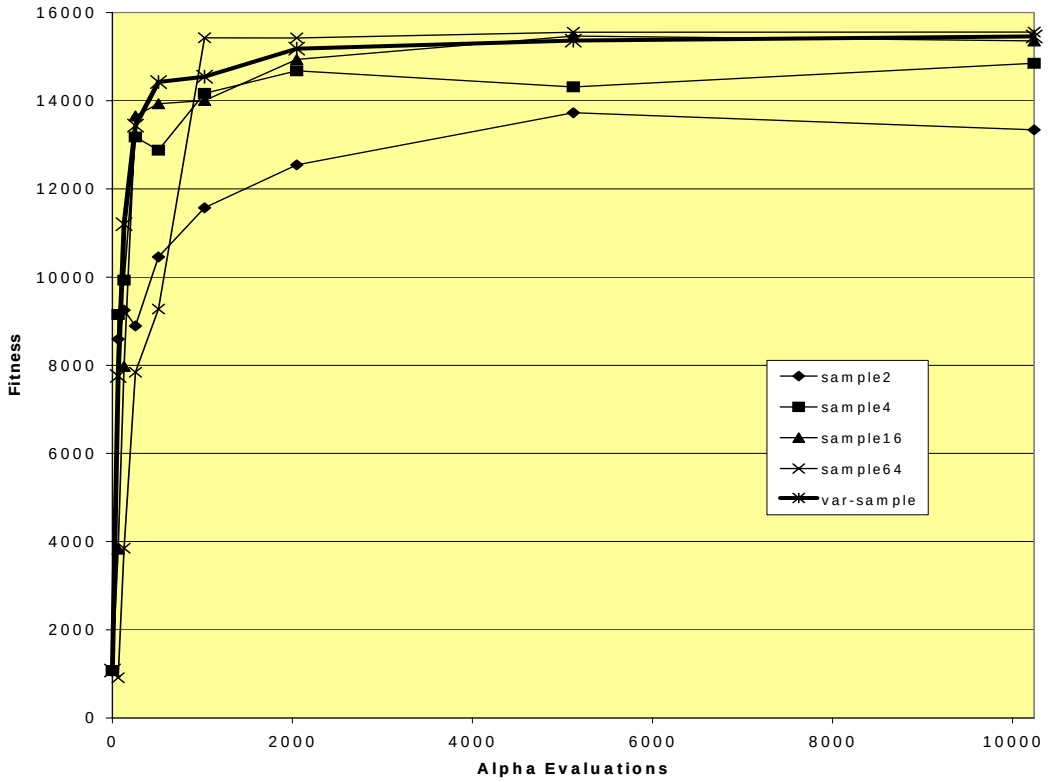


Figure 4. Comparison of the varying sample sizes method with the sample sizes 2, 4, 16, and 64. Each curve is an average of five separate runs of the GA. The variation sampling curve is shown in bold.

*D. Varying Sample Sizes*

Based on these results, we concluded that it was feasible to optimize our method by varying the sample size throughout evolution. This would allow us to capitalize on the accelerated growth of the lower sample sizes in the early generations, while taking advantage of the sustained growth of the higher sample sizes in the later generations. We started the evolution with a sample 2 because of its initial accelerated growth and then increased the sample size based on the fitness achieved by the best combination of partners. By studying the fitness graphs produced through the tests discussed in Section 3.3, we determined that it was appropriate to switch the sample size to 4 when the best team achieved a fitness score of 4,000, 16 when the team achieved a fitness score of 8,000 and then to 64 when the team achieved a score of 12,000. This method of increasing the sample size based on the current fitness can be extended to any finite fitness landscape. We chose to separate the fitness ascensions into four steps because our maximum fitness was around 16,000 which is easily divisible by 4. The graph in Figure 4 compares the performance of an average of five test runs for the varying sample sizes method and each of the sample sizes used as part of the evolutionary method. The data points represent the fitness achieved by the best combination of partners. By looking at the graph in Figure 5 it is clear that the varying sample sizes method produces an accurate solution with fewer alpha evaluations than any single sample size. This method reaches a fitness greater than 14,422 by the 512$^{th}$ alpha evaluation which corresponds to moving the box within approximately 4 units of the target corner in the simulation. A look at the data used to create Figure 4 can be seen in Table 1. When examining the average fitness scores it is evident that the varying sample sizes method not only produces an accurate solution with few alpha comparisons but its final solution after the 10240$^{th}$ alpha evaluation is highly accurate. Though the sample 64 reaches a greater fitness by the 10240$^{th}$ alpha evaluation, the discrepancy between these average fitnesses is less than 100 points. It is important to note that these results are specific to the box pushing task. The design of increasing the sample sizes based on fitnesses was derived from observing the behavior of all the sample sizes.

TABLE 1: TABLE OF AVERAGE FITNESS SCORES WHICH ARE PLOTTED IN FIGURE 5.

| | 0 | 64 | 128 | 256 | 512 | 1024 | 2048 | 5120 | 10240 |
|---|---|---|---|---|---|---|---|---|---|
| sample2 | 1075.87 | 8592.07 | 9248.61 | 8893.13 | 10457.67 | 11572.11 | 12544.2 | 13726.38 | 13339.81 |
| sample4 | 1075.87 | 9149.9 | 9931.98 | 13175.6 | 12876.34 | 14164.44 | 14676.55 | 14314.95 | 14847.15 |
| sample16 | 1075.87 | 3830.9 | 7975.32 | 13656.04 | 13931.37 | 14003.6 | 14936.59 | 15464.85 | 15356.91 |
| sample64 | 1075.87 | 914.58 | 3849.58 | 7843.88 | 9274.53 | 15428.84 | 15423.68 | 15554.57 | 15556.31 |
| var-sample | 1075.87 | 7754.68 | 11200.67 | 13436.36 | 14422.97 | 14546.82 | 15182.41 | 15364.77 | 15457.45 |

## IV. CONCLUSIONS

Our results show that it is possible to increase the sample size for alpha selection throughout the learning based on the fitness achieved by the best combination of partners. An advantage of the varying sample sizes method is that it gives a highly accurate solution at any point in the learning. The limitation of using a single sample size is that it does not always maintain consistent growth. If it a GA is evolving with a sample 64 the solutions in the early generations of training would be inferior even though it ultimately reaches the highest fitness. Conversely, if the GA is evolving with a sample 2, yet it may provide good solutions in the early generations, finding an accurate solution in the final generations of training is unlikely. No single sample size is capable of producing consistently accurate solutions throughout training.

This method also reduces computation time by producing an accurate solution with fewer generations of training. While our method of switching based on fitness has proved effective, it is only applicable to problems with a similar maximum fitness. In future research we will develop a more general method to determine the best time to change sample sizes during the evolution without any prior knowledge of the problem.

## REFERENCES


[1] Luke, S. and Spector, L., "Evolving Teamwork and Coordination with Genetic Programming," *Proceedings of First Genetic Programming Conference*. (1996), 150-156.

[2] Potter M. A. and De Jong K. A., "A Cooperative Coevolutionary Approach to Function Optimization," *Proceedings of the Third Conference on Parallel Problem Solving from Nature*. (1994), 249-257.

[3] Potter, M. A., Meeden L. A., and Schultz A. C., "Heterogeneity in the Coevolved Behaviors of Mobile Robots: The Emergence of Specialists," *Proceedings of the Seventeenth International Conference on Artificial Intelligence*. (2001).

[4] Wiegand R. P., Liles W. C., and De Jong K. A., "An Empirical Analysis of Collaboration Methods in Cooperative Coevolutionary Algorithms," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. (2001), 1235-1245.

[5] Parker, Gary B., "Evolving Cyclic Control for a Hexapod Robot Performing Area Coverage," *Proceedings of 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2001).* (2001), 561-566.

[6] Parker, Gary B. and Rawlins, Gregory J.E., "Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots" *Proceedings of the World Automation Congress (WAC '96), Volume 3, Robotic and Manufacturing Systems.* (1996), 617-622.

[7] Grefenstette, J. J. and Ramsey, C. L., "An Approach to Anytime Learning. *Proceeding of the Ninth International Conference on Machine Learning*, (1992), 189-195.

[8] Parker, Gary B., "Punctuated Anytime Learning for Hexapod Gait Generation," *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002).* (2002), 2664-2671.

[9] Parker, Gary B. and Mills, Jonathan W., "Adaptive Hexapod Gait Control Using Anytime Learning with Fitness Biasing," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99).* (1999), 519-524.

[10] Parker, Gary B., "Co-Evolving Model Parameters for Anytime Learning in Evolutionary Robotics," *Robotics and Autonomous Systems*, Vol. 33, Issue 1, (2000) 13-30.

[11] Parker, Gary B. and Blumenthal, J., "Punctuated Anytime Learning for Evolving a Team," *Proceedings of the World Automation Congress (WAC2002), Vol. 14, Robotics, Manufacturing, Automation and Control*. (2002), 559-566.

[12] Parker, Gary B. and Blumenthal, J., "Sampling the Nature of A Population: Punctuated Anytime Learning For Co-Evolving A Team," *Intelligent Engineering Systems Through Artificial Neural Networks (ANNIE2002, Vol. 12)* (2002), 207-212.

[13] Parker, Gary B. and Blumenthal J., "Comparison of Sampling Sizes for the Co-Evolution of Cooperative Agents," *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003).* (2003), 536-543.