Competing Sample Sizes for the Co-Evolution of Heterogeneous Agents

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

Abstract—**Evolving heterogeneous behavior for cooperative agents is a complex challenge. The co-evolution of separate populations requires a system for evaluation at trial time. If too few combinations of partners are tested, the GA is unable to recognize fit agents, but if too many agents are tested the required computation time becomes unreasonable. To resolve this issue, we created a system based on punctuated anytime learning that periodically tests partner combinations to reduce computation time. In continued research, we discovered that by testing fewer combinations the GA maintains accuracy while further reducing computation time. In this paper we propose a method that concurrently tests varying numbers of partner combinations and the spacing between these combinations at trial time to determine which is optimal for any stage of the co-evolution. We chose a box pushing task to compare these methods.**

Keywords-co-evolution; cooperative agents; team coordination; evolutionary robotics; learning; control

I. INTRODUCTION

The objective of our work is to define a method for evolving a team that simultaneously reduces computation time while maximizing accuracy. Finding a powerful system for team learning is essential because of its potential for application in today's world. Robots that cooperate can often achieve more than the sum of what they could do individually. 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 show the strength of our method because it represents a simplified task related to our method's possible applications. Learning cooperative behavior has been approached in several different ways.

Luke and Spector researched methods for increasing specialization and cooperation in team behavior [2]; a study of a variety of methods for breeding and communication in a Predator-Prey scenario. The simulated scenario was a toroidal (continuous) space where four agents representing "lions" chase a randomly moving "gazelle". Using the success rate of capture for comparison, two of Luke and Spector's conclusions are germane to our work. They concluded that restricted breeding promotes greater specialization than free breeding. In the context of co-evolution, restricted breeding would allow individuals to breed only within their own population. Their second conclusion was to consider the whole team as a single GP individual. Though this method proved successful, 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. 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 solution to this problem that capitalized on evolving team members in separate populations [8]. 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. The chosen members from opposing populations are selected as the best member from the previous generation. The "best" is determined by its relative fitness to other members of its own population. Using this system Potter, Meeden, and Schultz co-evolved artificially intelligent agents to heard sheep into a corral [9]. Their results showed this to be a successful method of producing heterogeneous behavior. Even though CCAs proved to be a powerful system of co-evolution, it still limits an individual's fitness calculation to a single partner.

Wiegand, Liles, and De Jong, examined factors that influenced co-evolution using different implementations of the CCA [10]. In their research they focused on collaborator selection issues and credit assignment at trial time. 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 in fact the collaborator pool size. Wiegand, Liles, and De Jong, also point out that as the pool size increases so does the computation required to perform the evaluations. Taking this to the extreme, the most accurate yet computationally expensive 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 (at punctuated generations) of a single individual from each population (referred to as the *alpha individual*) that represents the overall nature of its population. This selection is made by testing all the individuals in the population with a sampling of individuals from the other populations. The alpha individual is the individual that has the highest average fitness when working with the sample of individuals from the other populations. These alpha individuals are used by the genetic algorithm during the fitness evaluation of a population by testing each individual of the population when partnered with the alpha individual from the other population. We optimize our algorithm in terms of accuracy and reducing computation time by testing two different combinations of the number of individuals in a sample and the spacing (number of normal generations) between punctuated generations.

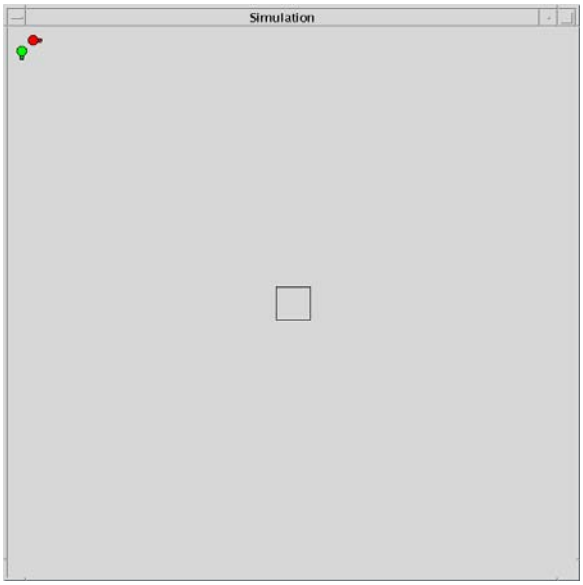

Figure 1. A snapshot of the simulation with everthing in its starting positions.

II. The Box Pushing Problem

The applied problem in our research is to have two hexapod robots starting from one corner of an enclosed square area walk to and push a box that is in the middle of the area to the opposite corner. The simulated area from which the task has been abstracted is an existing colony space in the Connecticut College Robotics Lab. This space measures approximately 8x8 ft. In the colony, two ServoBot robots and a cardboard box can be placed. The problem is 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.

The simulated environment used for evolving the agents measures 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 each trial, the agents and the box were placed in consistent starting positions. All coordinates in the simulation are positive, meaning 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 1 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 zero and increases by one with each consecutive non-aided push of the box. With F representing 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 $F/2^E$. The endurance factor reduces each agents pushing power by a factor of two every gait cycle without its partner. As soon as 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.

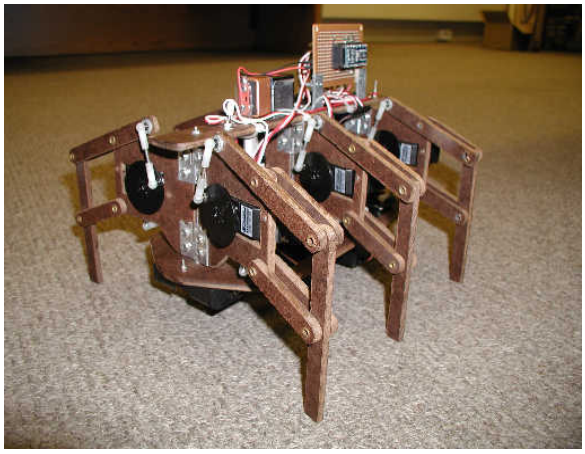

Figure 2. The ServoBot.

The robots modeled in the simulation are ServoBots. A ServoBot is an inexpensive hexapod robot 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 2). In order to control the motion of the robot, a BASIC Stamp II is mounted on the top of the ServoBot. This chip is capable of individually addressing each of the twelve servo actuators to produce and sustain a gait cycle. A gait cycle is defined

as the 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. Each of these activations represents the simultaneous movement of all twelve servos. This information is stored as a twelve-bit number, one bit for each servo. Every bit in the activation dictates the position of a servo with a 1 indicating full back or up and 0 indicating full forward or down depending on the orientation of the servo. Therefore, each pair of bits can represent the motion of a single leg with each bit controlling one servo, which corresponds to one of the two degrees of freedom. The pairs of bits are ordered by their represented leg as 0 to 5 with the legs 0,2,4 on the right side from front to back and 1,3,5 on the left in the same order (Figure 3). Also shown is an example twelve-bit activation number. Using this scheme, the number 001000000000 would lift the front left leg and 000001000000 would pull the second right leg backward. The BASIC Stamp provides a control pulse every 25 ms to drive the legs to their indicated positions.

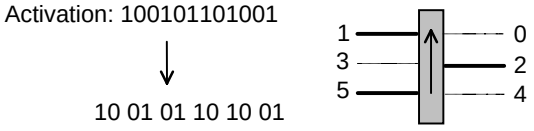

Figure 3. Diagram numbering the legs of the ServoBot and a sample twelve-bit activation. The dark lines represent legs that are down and moving back while the light lines represent legs that are currently or moving up and moving forward.

III. Cyclic Genetic Algorithms

A cyclic genetic algorithm is a variation of a traditional GA, where the genes of the chromosome represent tasks to be completed in a set amount of time [3]. These represented tasks can be anything ranging from a single action to multiple sub-cycles 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 contains 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 performed. The genes can be arranged into repeated sequences that form a chromosome that can be arranged with single or multiple cycles, or the entire chromosome can become a cycle.

The first way the CGA is used in our research is for evolving gait cycles. This can be done by coupling the CGA with the method of representation diagramed in Figure 3 to evolve a gait cycle for a specific ServoBot [3]. The optimal gait cycle for the ServoBot modeled in our simulation was a tripod gait. The tripod gait maintains static stability while maximizing the speed of forward motion for a hexapod. Legs 1, 2, and 5 are down and moving back while legs 0, 3, and 4 are lifting up as they move forward. This optimal gait cycle for the specific ServoBot used requires 58 pulses (29 pulses for thrust and 29 for repositioning) from the BASIC Stamp to each the left and right side servos for completion.

Different degrees of turns were generated for our ServoBot by decreasing the total number of pulses signaling thrust to the legs on one side of the robot. If the left legs 1,3,5 were given all 29 pulses of thrust but the right legs 0,2,4 were only give 14 pulses the result would be a right turn due to the drag created by the right legs throughout the duration of the gait cycle. Turns were measured by tests on the actual robot giving 15 left and right turns, plus a no turn. These performance values (measured in centimeters moved and degrees turned) were recorded and stored in the table with the addition of a "zero" value, corresponding to the robot not moving at all, giving a total of 32 turns.

The second way in which the CGA was used in our research was for the incremental evolution of a solution for the box pushing problem. The first stage of evolution defined the robots' behavior before they first touched the box, while the second stage determined their actions subsequently. The learning was done in two increments because the first stage required no cooperation while the second stage did. 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 occurs 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 repetitions of that gait cycle. The scheme representation of the chromosome is shown in Figure 4.

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

Figure 4. 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.

In the first increment of evolution, 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. The first increment of learning produced two populations of sixty-four individuals capable of touching 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 starting population had individuals with the first cycle evolved and the second cycle randomized.

In the second increment of evolution, both agents were placed in their respective starting positions for team evaluation. The fitness score of a team of agents a is the product of the positive distances the box 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 the box's starting position, the fitness of any given evaluation would be $((X_{final}- X_{start}) * ((Y_{final} - Y_{start}))$. The team is given a score of zero if the product of the coordinates is negative meaning the agents failed to advance the box towards target corner of the area in either the x or y directions. Since the box's starting position is (125,125) and the target corner is located at (250,250) the maximum attainable fitness is 125^2 or 15625.

IV. Co-Evolution Using Punctuated Anytime Learning

The concept of anytime learning was introduced by Grefenstette and Ramsey [1] to allow for the continuous updating of a robot's internal model. A system of Punctuated Anytime Learning (PAL) was developed by Parker [4] to strengthen offline genetic algorithms by capitalizing on the dynamic nature of the anytime learning approach. Since offline GAs cannot allow for continuous updates of the computer's models, the PAL system updates its model every *G* generations, resulting in periods of accelerated learning. The generations in which the model is updated are referred to as "punctuated" generations. When applied to the evolution of a single population, PAL updates the learning system every G generations by running tests on the actual robot and using these results for fitness biasing in the GA or for the co-evolution of model parameters.

Punctuated anytime learning is a fairly different concept when applied to co-evolving populations [5]. The two systems of evolution bear the same name because they both capitalize on the idea of periodic updates during punctuated generations. 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* generations, all individuals in population A are tested against all individuals in population B. The purpose of this process is to find the individual from each population that pairs best with members of the other population for evaluation at trial time. The chosen individual from each population will be referred to as the *alpha individual*. The most accurate method of evolution would be to select new alpha individuals at each generation. However, the process of alpha selection requires significant computation. Assuming there exists *I* individuals in each population and *N* populations, the computer must perform I^N trials for each generation at trial time. To circumvent this level of computation, new alpha individuals are selected only at certain consistently spaced periods of time, the punctuated generation. During normal, non-punctuated generations, the alpha individuals selected from the last punctuated generations are paired with perspective team members in the evolving 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.

A. 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 teams of three or more agents. We realized that it was possible to select an alpha individual while testing less than the entire population by taking a sample of the population [6]. 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 is much computationally cheaper than using our previous method requiring I^N comparisons to select alphas.

To test the success of the sampling method we tested 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 [7]. In order to accurately measure the sample sizes strengths, it is essential to ensure that each sample size performed an equivalent number of fitness comparisons. The number of total alpha evaluations performed during a period of training depends on the sample size (*S*) and the spacing between punctuated generations (*G*). Since the key to comparing different sample sizes is ensuring an equal number of evaluations, we define *sampling rate* as *S/G*. In testing, the sample rate was set to 1, meaning the sample size used for alpha selection was equal to the spacing of the punctuated generations. Sample 1 performed alpha selection every generation, sample 2 performed alpha selection every second generation, and so on. In addition to these alpha evaluations, each generation also incurs 128 normal evaluations (the number of evaluations during a non-punctuated generation), one for each of the sixty-four members of each population.

Fitnessses were recorded at 0, 64, 128, 256, 512, 1024, 2048, 5120, and 10240 alpha evaluations. It was found that the higher sample sizes, which also meant more time between punctuated generations, did poorly in early evolution, but better in later evolution. The lower sampling sizes of 1, 2, and 4, have relatively inferior performance after the 1024^{th} alpha evaluation. They are in general 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. Sample 1 and 2 lack the ability to represent the true nature of a population with so few comparisons for alpha selection; their fitness tend to plateau. 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.

V. COMPETING SAMPLE SIZES

Based on these results, we concluded that it is feasible to optimize our method by using the smaller sample sizes in the earlier generations and the larger sample sizes for evolution in the later generation. This would allow the learning system 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. The best way to optimize the system would be to allow the learning system to choose the appropriate sample size for evolution at any given time; a system of predefined sample size switching can only be realized through extensive testing and observation. To give the learning system liberty to choose a sample size, we created a method that simultaneously evolves two separate populations with different sample sizes. This allows the algorithm to maintain the population of the current best sample size for evolution while testing it against a population with a sample size twice as large.

In order to determine which sample size is best suited for evolution we need to ensure that both competing sample sizes begin their respective generations of training on the same populations with equal number of alpha evaluations incurred during the period of training. To achieve this we set the two competing sample sizes to the same sampling rate. In addition, in order to compare in this new method with those of the past, the sampling rate needed to be set to ½ since there were two evolutions can concurrently taking place. This means that a sample 1 selects alphas every other generation while a competing sample 2 selects alphas every fourth generation, etc. Not only does this provide an equal number of evaluations during a period of training, but it also keeps the total number of evaluations incurred the same as running only a single sample size at the sampling rate of 1. The evolution is started with a sample 1 at generation 0. Each of the two populations is copied and alphas are then selected by both sets of sample sizes. These two sets are evolved until the higher sample size undergoes a second round of alpha selection then the fitnesses of the newly selected alphas are compared. In the case of a competing sample 1 and sample 2, the fitnesses of the alphas are compared after every fourth generation of training. If the higher sample size selects a more fit team of alphas, the populations with which it evolved are then cloned and evolved with a sample size twice as large. On the other hand, if the lower sample size evolves more fit alpha individuals, the two currently competing sample sizes continue until the higher sample size selects new alphas.

A. Results

To test our method, we ran five separate tests of the GA for 5120 generations each with the sampling rate set to 1/2. The results are plotted in Figure 5. The x-axis represents the total number of alpha evaluations performed by both competing sample sizes added together, and the y-axis represents the fitness score achieved by the current best sample size's selected alpha individuals and were recorded at the generations 0, 64, 128, 256, 512, 1024, 2048, and 5120.

When analyzing the above graph it is clear that this method is very successful, with all 5 test runs reaching a fitness of above 14,000 by only the 1024th generation and three of the five tests reaching the same level well before. The optimization of our method is even clearer when compared to the single sample fitness scores shown in Figure 6, where only the sample 64 reaches a fitness greater than 14,000 by the 1024th generation. As can be seen, the competing sample sizes method is compared with

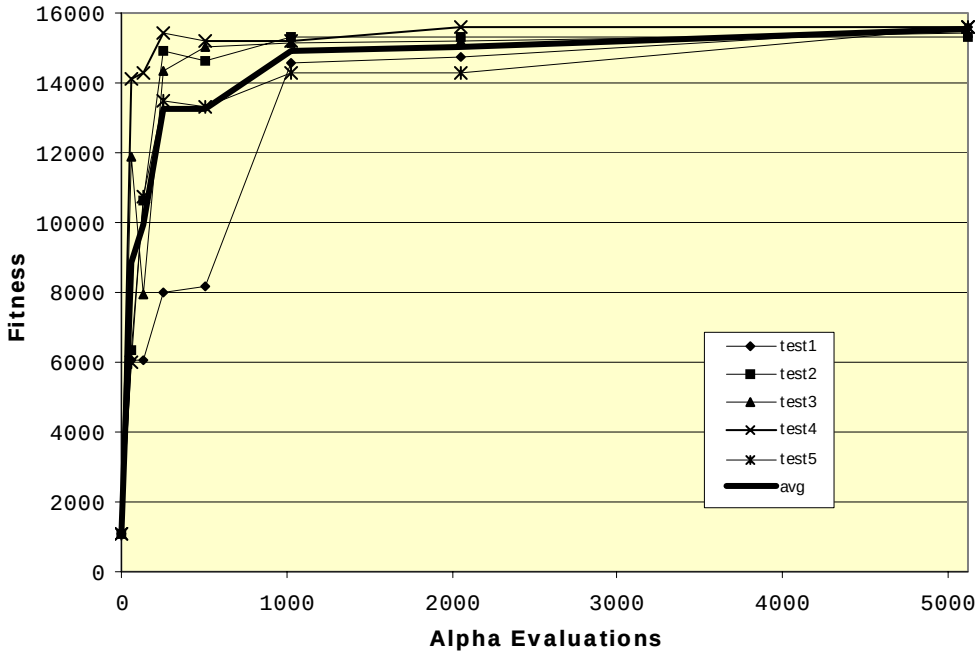

Figure 5. Plotted above are five separate test runs of the competing sample GA with the sampling rate set to 1/2. The average is shown in bold.

seven other methods, each using a set sample size. The new method (shown in bold) is comparable to the lower sample sizes during early evolution and higher sample sizes during late evolution. It makes the changes to the sample sizes when needed to be most effective for the evolution of a solution to the problem.

The greatest advantage of using the system of competing sample sizes is that the learning system decides which sample size is best suited for a period of evolution. If a person is forced to choose appropriate sample sizes, this requires extensive testing of all sample sizes in separate GA runs to observe their behavior specific to a particular problem. With this new method of competing sample sizes it is possible to find an extremely accurate yet computationally inexpensive solution to a problem with only a single run of the GA.

VI. Conclusions

There are many challenges involved when attempting to optimize a method of co-evolution. It is essential to have the learning system decide when to adjust sample sizes because the optimal switching points can vary greatly between different evolutions on the same problem. A plausible way to punctually, accurately, and objectively adjust the sample size is to test different sample sizes during the evolution itself. To avoid doubling the level of computation from our previous method, we set the sampling rate to half of its original value. The results show that not only does this method produce a highly accurate solution but the timely switching of sample sizes reduces the overall computation time. In the future, we hope to extend this method to problems involving more than two agents to demonstrate the scalability of the method.

References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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).

[10] 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.

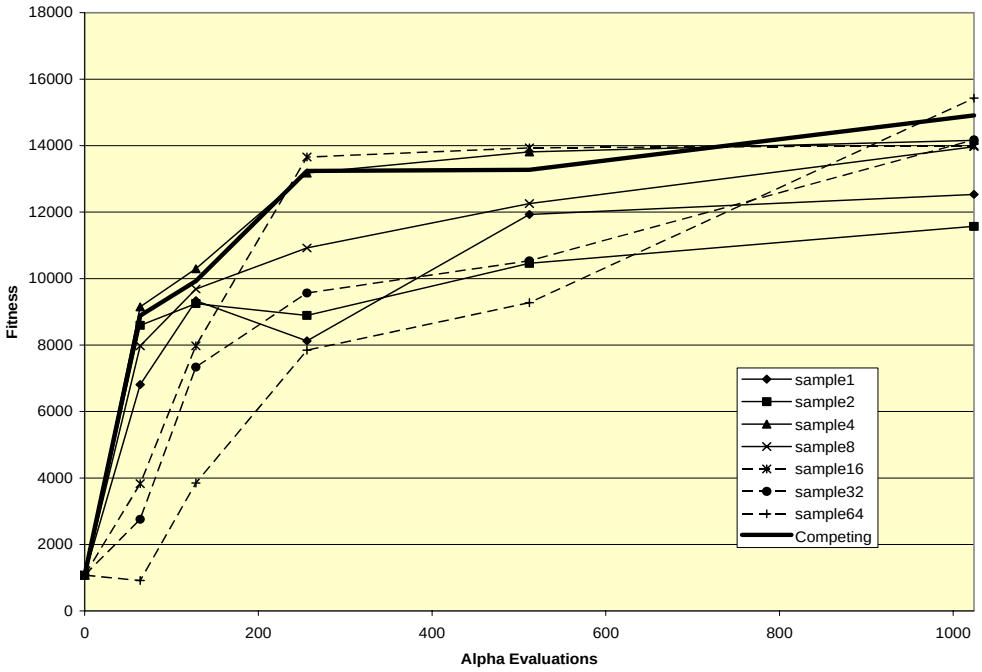

Figure 6. This graph shows a comparison of the competing sampling sizes method (shown in bold) compared to using set sampling sizes. In each case, the line shown is the average of 5 runs.