Comparison of Sampling Sizes for the Co-Evolution of Cooperative 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 behavior can be a very demanding task. In order to promote the greatest level of specialization team members should be evolved in separate populations. The greatest complication in the evolution of separate populations is finding suitable partners for evaluation at trial time. If too few combinations are tested, the Genetic Algorithm loses its ability to recognize possible solutions and if too many combinations are tested the algorithm becomes too computationally expensive. In previous work a method of punctuated anytime learning was employed to test all combinations of possible partners at periodic generations to reduce the number of evaluations. In further work, it was found that by varying the number of combinations tested, the sample size, the GA could produce an accurate and even less computationally expensive solution. In this paper, we compare different sampling sizes to determine the most effective approach to finding the solution. We use a box pushing task to compare these different sampling sizes.

1 Introduction

The objective of our work is to find a method of producing cooperative agents using co-evolution that maximizes accuracy while minimizing computational cost. Robots that cooperate can often achieve much more than the sum of what they could do individually. Learning cooperative behavior for robots has been approached in several ways.

Luke and Spector developed a method for evolving a team of agents that display heterogeneous behavior [luke96]. The applied problem in their research was the Predator-Prey scenario in which four artificially intelligent agents representing “lions” attempted to entrap the prey represented as a “gazelle”. They used genetic programming and considered each team of four “lions” as one individual instead of evolving team members in separate populations. Even though this method proved successful in producing cooperative emergent behavior, evolution of teams in a single population limits specialization. This approach compromises the ability of the GA to recognize a suitable team member because a partner’s score is overly influenced by the performance of other members of the team. Learning control of each member in a separate population plays to the strength of a genetic algorithm. The members of the population will tend toward a specialization and the evolutionary power of the GA would be concentrated in producing the best and most specialized individual possible.

Potter and De Jong capitalized on evolution of behavior in separate populations by creating cooperative co-evolutionary algorithms (CCAs) [pott94]. This method evaluated an agent’s fitness by testing it with a single member from the other population. This chosen member from the opposing population is the best member from the previous generation. Best is determined by the individual’s fitness compared to others in its own population. Potter, Meeden, and Shultz utilized this method to co-evolve artificially intelligent agents to heard sheep into a corral [pott01]. The strength of the co-evolution was further tested by introducing agents representing wolves, which tried to attack the sheep. This method proved to be effective and the evolved agents successfully moved the sheep into the corral while protecting them from the predators in the simulation.

Although this method proved to be an excellent means of producing heterogeneous behavior, the CCA method still limits each individual’s fitness calculation to being computed with only a single partner. Wiegand, Liles, and De Jong discuss the pertinent issues to optimizing co-evolution of teams [wieg01]. They examine in particular the collaborator selection issues. Their conclusion was that the most influential factor on the overall success of the co-evolution is the collaboration pool size, or the number of combinations tested at trial time. However, they note that as the collaborator pool size increases, so does the computational cost of a generation of training. By this rational, the most accurate method would be to tests all individuals with every possible partner. A task requiring N partners with I individuals in each population, any given generation of training would require I^N evaluations; not an acceptable solution.

Parker developed Punctuated Anytime Learning (PAL), a method to allow for the learning system to be periodically updated throughout simulated evolution [park02]. The computer's model in simulation is updated or the GA fitness evaluation altered by measuring the robot's actual performance at certain consistently spaced number of generations called punctuated generations and entering those values into the GA.

Parker and Blumenthal adapted the concept of PAL to be applicable to evolving cooperative teams [park02]. This method used the periodic nature of PAL to minimize the number of fitness evaluations required to evolve team members in separate populations. The most accurate method of fitness evaluation would be to get an individual's fitness by pairing it with all possible partners at trial time. In order to reduce the number of fitness evaluations, at certain punctuated generations this method selected a single individual from each population as the best representative of the nature of their population. This selected individual, referred to as an alpha individual, was used as a partner at trial times for evaluating the fitness of any individual in the opposing population. Employing this method of PAL, with N generations between each round of alpha selection, the number of fitness evaluations is reduced by a factor of approximately N. Although these results showed that this method produces a highly accurate solution, it is still too computationally intensive to accommodate more than two populations.

Additional research showed that it is possible to further reduce computations during alpha selection by testing each individual's fitness with less than the entirety of the opposing population [park02]. The chosen group used for alpha selection is referred to as the *sample* and the number of individuals in that sample is called the *sampling size*. This adjustment significantly reduced the number of evaluations required to find an accurate solution. In this paper, we compare different sampling sizes, while ensuring equal fitness evaluations by varying the number of generations between alpha selections, in order to determine which sampling size offers the most accurate solution for the chosen box pushing task.

2 Problem Description

The task is to have two hexapod robots starting from one corner of an enclosed square area to push a box that is situated in the middle of the area to the opposing corner. The scenario from which the task has been abstracted is a colony space in the Connecticut College Robotics Lab measuring approximately 8x8 ft. In this area, the two ServoBot robots and a square 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.

2.1 Simulation of Robot Performance

The robots simulated are modeled after ServoBots, which are inexpensive hexapod robots made of pressed wood with twelve hobby servos (Figure 1). The movements of the servos are coordinated by a central controller. This controller is a BASIC Stamp II capable of individually addressing each of the twelve servo actuators (two on each leg) to produce and sustain forward motion. The BASIC Stamp II is capable of storing a sequence of timed activations to be repeated. These timed activations if sequenced correctly produce a gait cycle 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 motion. Each single activation represents the simultaneous movement of the twelve servos. The list of controls for the twelve servos is represented in the controller as a twelve-bit number. Each bit represents a single servo with a 0 or a 1. For the horizontal servos a 0 indicates full back and a 1 indicates full forward. Likewise, for the vertical servos a 0 corresponds to full down and a 1 corresponds to full lift. Therefore, each pair of bits can represent the motion of one leg, each bit controlling one servo, corresponding to one of the two degrees of freedom. The pairs of bits are ordered to their represented leg as 0 to 5 with legs 0,2,4 being on the right from front to back and 1,3,5 being on the left from front to back (Figure 2). Figure 2 also shows the corresponding twelve-bit activation. By this example, the number 001000000000 would lift the front left leg up, and 000001000000 would pull the second right leg backward.

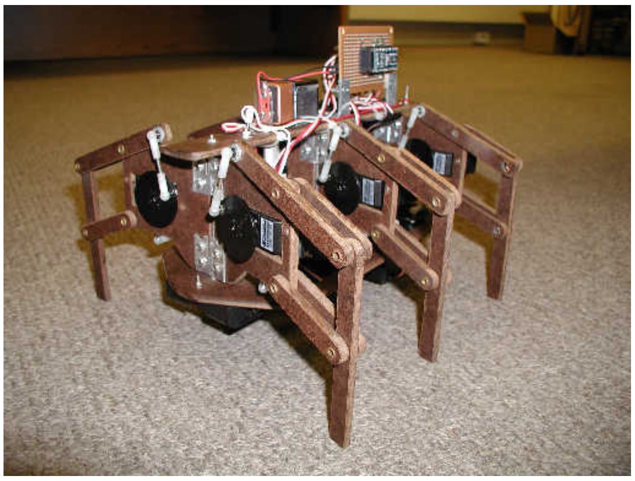
Figure 1: The ServoBot

Each activation is held by the controller for one pulse (approximately 25 msec). With this method of representation, a cyclic genetic algorithm (discussed in a later section) can be used to evolve an optimal gait cycle for a specific ServoBot [park01]. The gait cycle used in our simulation was a tripod gait, in which three legs provide thrust while three legs are repositioning to provide thrust on the next set of activations. The CGA for our specific ServoBot, learned a near optimal gait cycle, which requires a sequence of 29 pulses to move a servo from full back to full front.

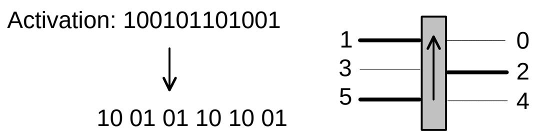

Figure 2: Diagram numbering the legs of the ServoBot and a sample twelve-bit activation.

Different degrees of turns were then generated for our ServoBot by decreasing the total number of pulses sent to one side of the robot. If legs 1,3,5 were given all 29 pulses but legs 0,2,4 were only given 15 pulses the result would be a right turn due to the drag created by the left legs (0,2,4) throughout the duration of the gait cycle [park01]. The effects of each of the 15 left and right turns, plus no turn, were measured as they were performed by the ServoBot being tested. These turns are unique to the particular ServoBot, for example the recorded "no turn" actually veered left due to minor differences in the physical construction. These 31 performance values (measured in centimeters moved and degrees turned) were recorded and stored in a table.

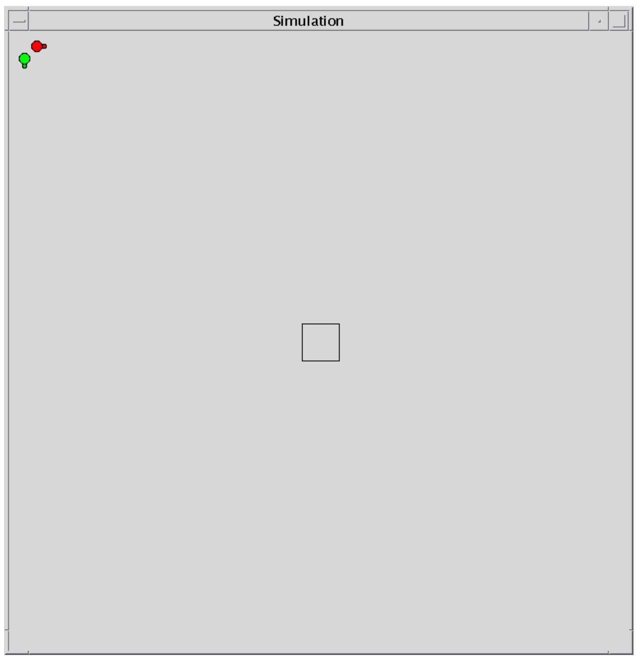

Figure 3: Simulation of the colony area.

2.2 Simulation Environment

The simulated environment used for evolving the agents was an abstraction of the colony space in the lab. The simulated area measured 250x250 units. The environment is peculiar in that the coordinate position of (0,0) is located in the upper left corner of the plane. Both robots were represented as circles but the robots were treated as single points for rules of contact with the box, which was represented as a square. In each trial, both the robots and the box were placed in consistent starting positions. The first robot started on the point (10,5) and faced parallel to the x-axis, while the second robot started in the mirrored position (5,10) but faced parallel to the y-axis. The box started in the middle of the environment at the point (125,125). See Figure 3 for a snapshot of the simulation with everything in its starting positions.

Each robot's ability to push the box on its own (without aid from its partner) was affected by an endurance factor. The endurance factor starts at zero and increases with each consecutive non-aided push. With F representing the would be full force of the robot 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 cuts their pushing power in half after each gait cycle. As soon as both robots push the box simultaneously, both of their endurance factors are reset to zero. In the simulation, both robots move simultaneously, and a trial ends when either each robot has taken 200 steps or one of the three (the two robots or the box) moves out of the simulated area.

3 Method

The behavior of both agents was evolved incrementally in two stages. The first stage defined their actions before they first touched the box, and the second defined their actions afterward. The first stage required no cooperation while the second part did. The second stage was evolved using punctuated anytime learning, which allows for the updating of the computer's models during evolutionary computation.

3.1 Evolutionary Methods

A variation on the standard GA called a cyclic genetic algorithm [park01] was used to develop our two heterogeneous cooperative agents. A CGA is much like a regular GA, but in the CGA the genes of the chromosome can represent tasks to be completed. 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 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 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.

Individuals were selected stochastically for breeding based on their fitness score and standard operators were used for the CGAs. Incremental learning was employed because the problem can be easily broken down into two smaller tasks allowing for greater specialization in etc cooperation. The first increment required no cooperative behavior. For the first increment, two completely separate populations were evolved; one for each robot. They were evolved using an identical method except for the robot's starting positions. 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 after the trial finished or zero if the individual failed to move the box positively in the y direction. For population B starting at (5,10) facing down the y-axis, the individual's score was computed the same as for an individual in population A, except the robot was charged with moving the box positively in the x direction to receive a non-zero score. The second increment of the learning process, which will be discussed later in section 3.1 was more complex and involved the use of punctuated anytime learning in addition to the CGA.

The fitness score of a team 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.

The CGA was perfectly fit for our evolution because it is designed for learning cyclic behavior and it allowed for our incremental learning approach. The set of actions to get each agent to the box was one cycle and each agent's behavior after touching the box was defined in the second cycle of their CGA chromosome. During a trial, when the robot touches the box, after the completion of the current gait cycle the controller would switch from the first to the second cycle. The two cycles of the CGA chromosome had nine genes each. Every gene contained two 5-bit numbers, one representing a gait cycle with 31 possible turns or a 0 which indicated that it was to stand still and the other representing the repetitions of that gait cycle. The scheme representation of the chromosome is shown in Figure 4.

$$(((T_1\ R_1)\ (T_2\ R_2)\ \ldots\ (T_8\ R_8))\quad \mathbf{((T_1\ R_1)\ (T_2\ R_2)\ \ldots\ (T_8\ R_8)))}$$

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, which appear in bold, represent the second cycle.

Only the first cycle of the CGA chromosome was evolved during the first increment of learning. A population of these chromosomes learned for each team member during this first increment was used to evolve each team in the second increment. The first cycles remained unchanged while the second cycles for each chromosome were randomly generated to create start populations for the second increment of learning.

3.2 PAL For Evolving Team Coordination

Punctuated anytime learning (PAL) was developed to strengthen offline genetic algorithms by capitalizing on Greffenstette and Ramsey's dynamic anytime learning approach [gref92] . Although PAL cannot allow for 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 [park02] or in the co-evolution of model parameters [park99].

Punctuated anytime learning is a fairly different concept when applied to co-evolving separate populations to form members of a team. The updated information that each population in the learning receives is a more accurate representation of the overall nature of the other population. For ease of explanation, assume that the experiment has two populations, population A and population B. In this case, every *G* generations, selected 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 the other population. The chosen most fit individual from each population will be referred to as the "alpha individual". The generations in which the computer finds new alphas are called "punctuated" 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, it also decreases the total number of evaluations required to find an accurate solution.

3.3 Sampling Populations

The original adaptation of PAL was to perform alpha selection at punctuated generations by testing all members of a population A against all members of a population B. This method proved to be a powerful system for evolving teams. Although it was effective, this method remains too computationally expensive. In order to further reduce computation time, we tested the possibility of selecting alphas using less than the entire population. Assuming that the experiment has two populations, population A and population B, every *G* generations, some chosen number of individuals in population A are randomly selected and tested against all individuals in population B. The selected individuals from population A are referred to as the *sample*, and the number of chosen individuals is called the *sampling size*. Our tests [park02] involved using a sample size of 8, which was thought to be a good starting point as it is the square root of our original sample size of 64. We found the sample size of 8 to be both accurate and swift in alpha selection.

For the research reported in this paper, we tested a variety of sampling sizes to investigate their merits. We realized the strength of combining the periodic nature of PAL with a smaller sampling size to reduce the intensive computation time in evolving team behavior.

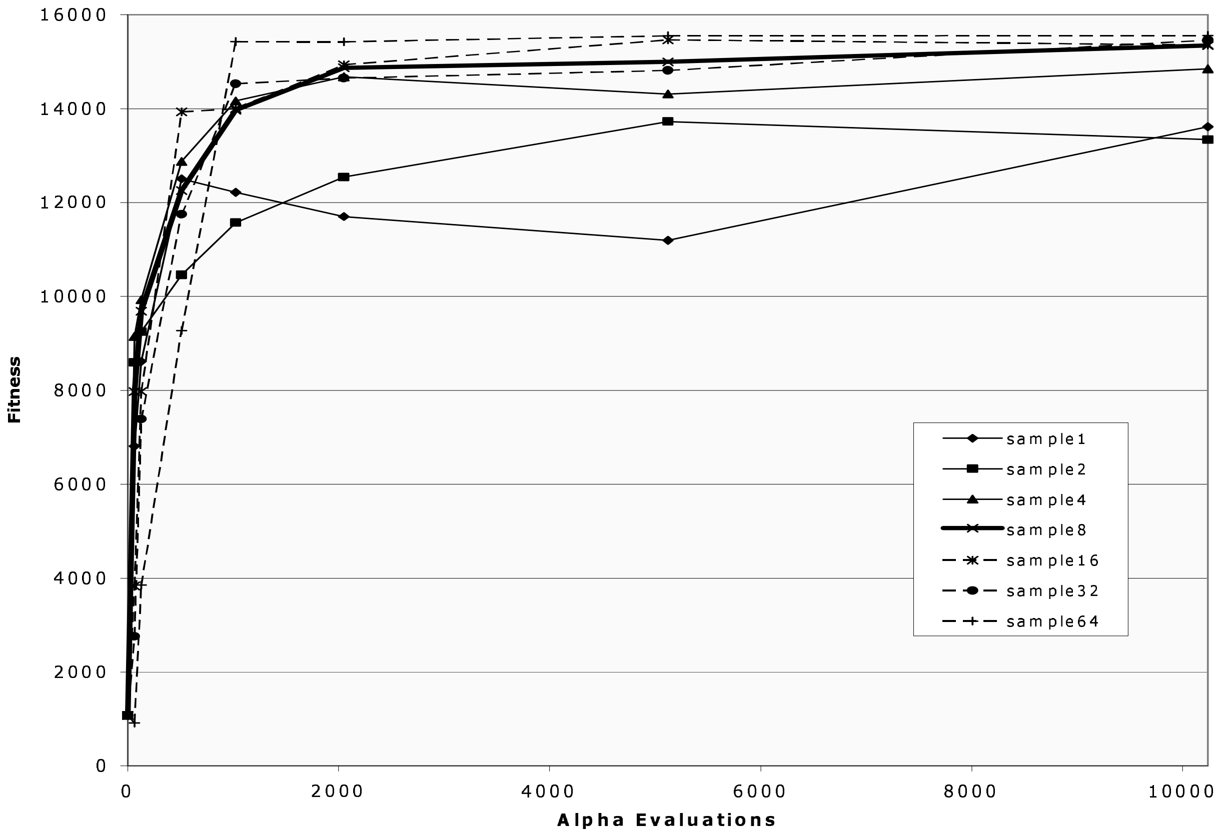

Figure 5: Results of the box pushing, shown through 10,240 alpha evaluations for sampling sizes 1, 2, 4, 8, 16, 32, and 64. Each curve is an average of five separate runs of the GA. Sample 8 is shown in hold and all higher sampling sizes are shown as a dashed line.

To express mathematically the computational reduction achieved by this union of methods, we let *G* represent the number of generations between alpha selections, *I* represent the number of individuals in a single population, and *N* represent the number of populations. The most accurate method of testing would be to compare all individuals in a population against all others in the opposing population for alpha selection every generation. A single generation of training would therefore require I^N evaluations. To reduce this level of computation, alphas are only selected at punctuated generations. If alpha selections occur every *G* generations, this reduces the evaluations by that factor of *G*. This solution cuts computation time to I^N/G, but further reductions are necessary to accommodate the co-evolution of more than two populations. In order to further reduce computations, sampling is used. Using the previous parameters and adding the term *S* representing the sampling size, any given alpha selection requires only $2 * (I * S^{N-1})$ trials. If done every *G* generations then the number of trials is $(N * (I * S^{N-1}))/G$.

4 Results

In order to establish the relative strength of the sampling rates, we staggered the punctuated generations such that each sample rate performed the same number of fitness evaluations at the generations when results were recorded. This staggering also allows us to ensure an equivalent number of alpha evaluations, those trials dedicated to alpha selection. The sampling sizes tested where one, two, four, eight, sixteen, thirty-two, and sixty-four. The arrangement of the sampling sizes was set such that the sample one performed alpha selection every generation, the sample two performed selection every other generation, sample four performed selection every fourth generation. This system of testing forces an equivalent number of alpha evaluations and generations of growth. In addition to these alpha evaluations, each generation also incurs 128 normal evaluations, one extra for each of the sixty-four members of both populations. The performance of the most fit combination of partners for each of the seven different sampling rates plotted on the graph (Figure 5) are each the average of five separate test runs of the GA. The x-axis represents the total number of alpha evaluations during training and the y-axis represents the fitness achieved from the combination of the best from each population to form partners. Fitnesses were recorded at 0, 64, 128, 256, 512, 1024, 2048, 5120, and 10240.

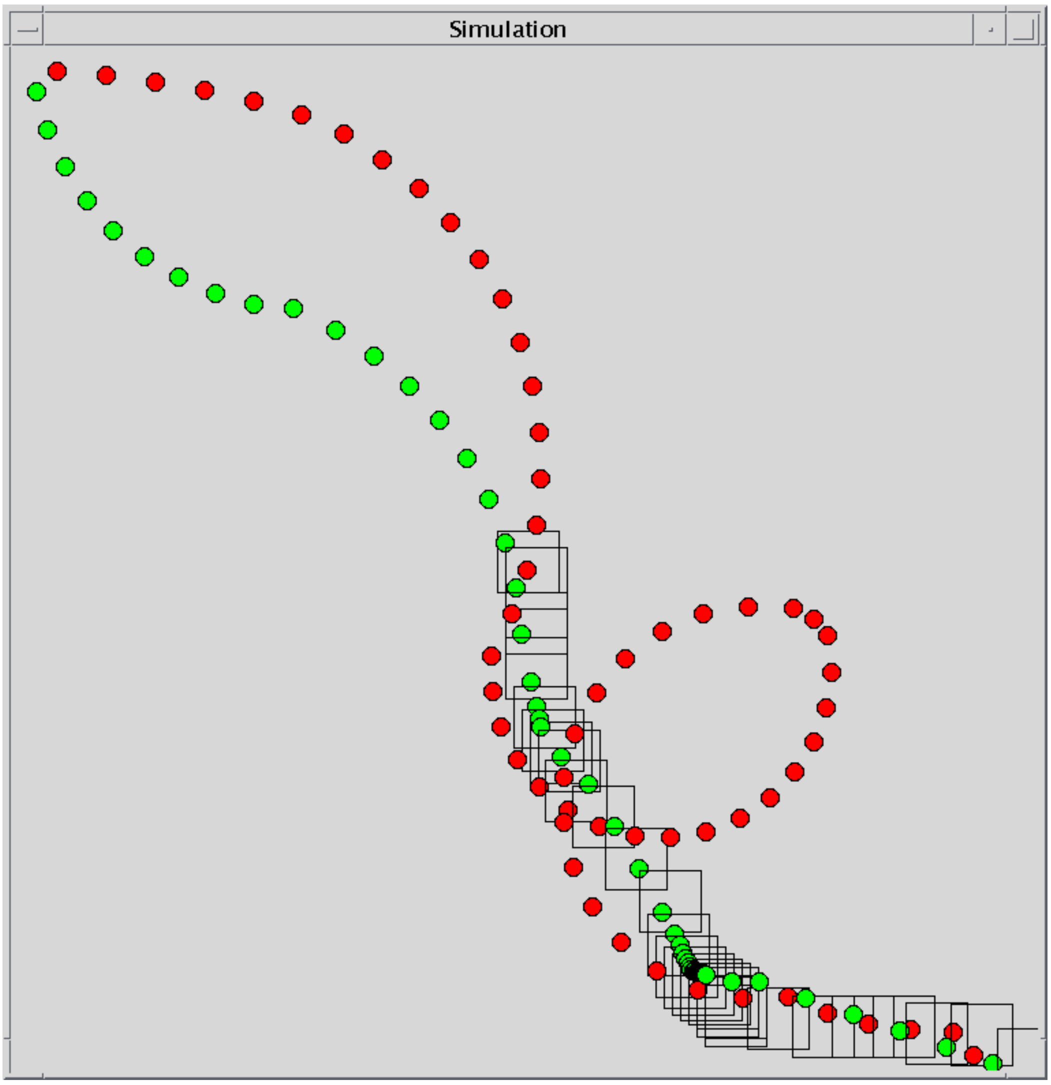

Figure 6: Graphical simulation of a sample 64 trial that achieved maximum fitness.

Figure 5 shows that all seven of the sampling sizes reach reasonably accurate solutions, though sample 64 reaches the highest average fitness of the best combination of partners of any sampling size. You can see in Figure 5, the curves representing the lower sampling sizes 1, 2, and 4, do not reach above the sample 8 after the 1024th alpha evaluation. The lines representing the higher sampling sizes 16-64 outperform the sample 8 at later generations, with the exception of sample 16 at 5120 alpha evaluations.

We examined the graphical simulations of the tests that were averaged together to create the data series in Figure 5. When observing the simulations of the higher sampling sizes (8, 16, 32, & 64) we noticed that these sizes push the box within three units of the corner area (250,250), yielding a fitness score over 15,000. All five of the sample 64 tests moved the box within 1.5 units of the target corner. This is consistent with our assumption that the sample 64 is the most accurate method. Refer to Figure 6 for a snapshot of a graphical simulation of a sample 64 trial at maximum fitness.

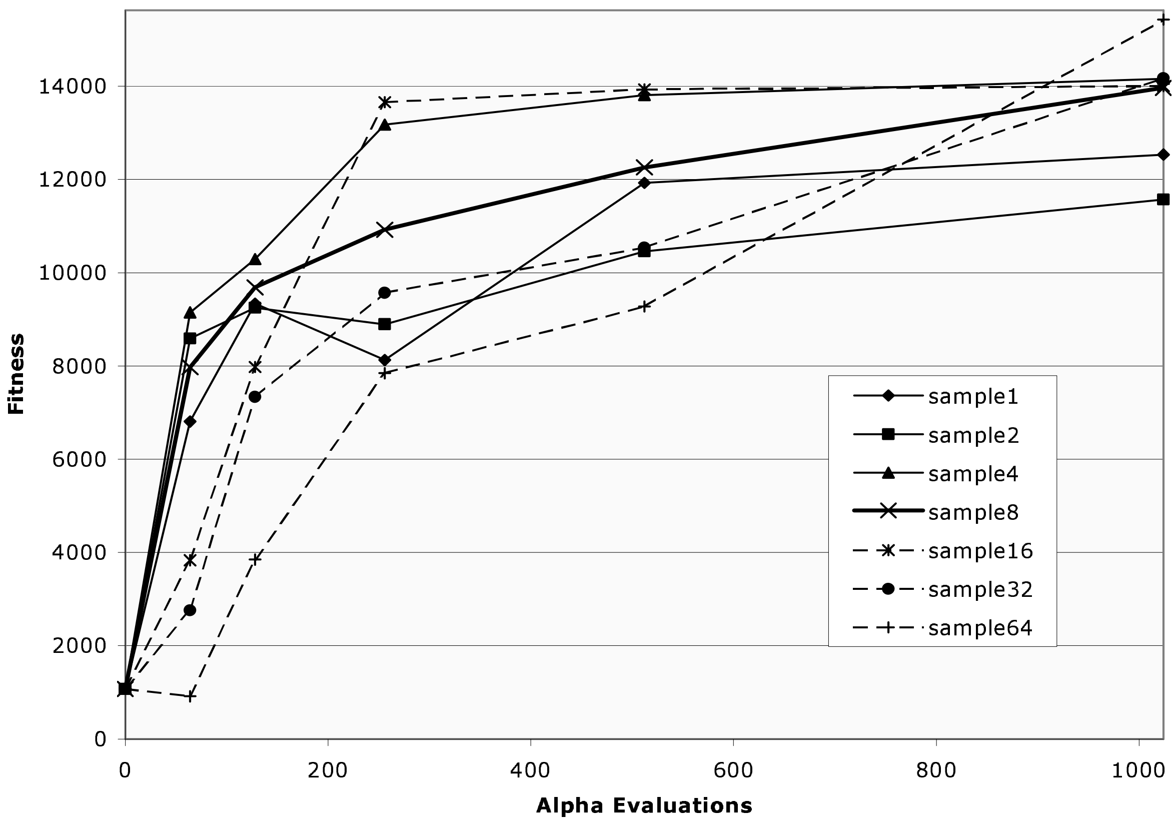

Figure 7: Results of the box pushing for sampling sizes 1, 2, 4, 8, 16, 32, and 64. Each curve is an average of five separate runs of the GA. The lower sample sizes are shown as dashed lines to demonstrate accelerated initial growth. Sample 8 is shown in bold.

A closer look at the lower alpha evaluations portion of the results (Figure 7) shows that the lower sample sizes 1, 2, and 4 are, in general, better in the initial stages of learning. This quality of the sample sizes shows their application for optimizing the method. These sampling rates exhibit their accelerated growth up to the 128th 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 test continues its accelerated growth past the sample rates of 1 and 2. We believe that the sample 1 and sample 2 lack the ability to represent the true nature of a population with so few comparisons for alpha selection. This is further confirmed by noting their overall behavior in Figures 5 and 7. In both the lower and upper ranges of learning these two sampling sizes lack consistency.

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. Therefore the sampling rates of 4, 8, or 16, represent strong candidates for consistent and sustained growth. In both Figures 5 and 7 sample 8 is shown in bold because it is the median of the series.

5 Conclusions

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. While these results can be considered indicative of the method it is important to note that the results are specific to the box-pushing task.

Reducing the number of generations between alpha selections is a key factor in early generations and having a high sampling rate is needed in late generations. Therefore, a possible approach to optimizing our GA would be to utilize ascending sampling sizes based on the current fitness levels of the evolution. This would play to the strength of the nature of our sampling method as the lower rates would promote initial acceleration in growth and the higher rates would provide consistent growth, which endures until an optimal solution can be reached.

References

[gref92] Grefenstette, J. J. and Ramsey, C. L. “An Aproach to Anytime Learning.” *Proceeding of the Ninth International Conference on Machine Learning*, (1992), 189-195.

[luke96] Luke, S. and Spector, L. “Evolving Teamwork and Coordination with Genetic Programming.” *Proceedings of First Genetic Programming Conference*. (1996), 150-156.

[park99] Parker, Gary B. “The Co-Evolution of Model Parameters and Control Programs in Evolutionary Robotics.” *Proceeding of the 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation*. November 1999 (162-167).

[park01] Parker, Gary B. “Learning Control Cycles for Area coverage with Cyclic Genetic Algorithms.” *Proceeding of the 2nd WSES International Conference on Evolutionary Computation (EC ’01)*. February 2001 (283-289).

[park02] 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)*. October 2002 (2664-2671).

[park02] Parker, Gary B. and Blumenthal, J., 2002, "Punctuated Anytime Learning For Evolving A Team." *World Automation 2002 Congress Proceedings***.**

[park02] Parker, Gary B. and Blumenthal, J, 2002, “Sampling the Nature of A Population: Punctuated Anytime Learning For Co-Evolving A Team.” *Intelligent Engineering Systems Through Artificial Neural Networks (ANNIE2002, Volume 12)* (2002), 207-212.

[pott01] 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).

[pott94] 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.

[wieg01]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.