PUNCTUATED ANYTIME LEARNING FOR EVOLVING A TEAM

GARY B. PARKER and H. JOSEPH BLUMENTHAL
Computer Science, Connecticut College, New London, CT 06320
parker@conncoll.edu and hjblu@conncoll.edu

ABSTRACT

Learning heterogeneous behaviors for robots to cooperate in the performance of a task is a difficult problem. Evolving the separate team members in a single chromosome limits the capacity of the genetic algorithm to learn. Evolving the separate team members in separate populations promotes specialization and gives the genetic algorithm more flexibility to produce a solution, but can be either computationally prohibitive or result in credit assignment complications. In this paper, we apply punctuated anytime learning to assist in the co-evolution of separate team member populations. A box-pushing task is used to show the success of this method.

KEYWORDS: cooperative, co-evolution, robot, hexapod, cycle, genetic

INTRODUCTION

The objective of our work is to develop a robust method of co-evolving two separate populations to produce cooperative behavior in two specialized members of a team. 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 focused research on methods of increasing specialization and cooperation in teams [2]. Specifically, they tested different methods of breeding and communication to optimize solutions. The applied problem in this research was the Predator-Prey problem in which the goal was to have four “lions” trap a randomly moving “gazelle” in a simulated environment. Using the Predator-Prey scenarios, the comparison of breeding and communication methods was based on the success rate of the four evolved agents trapping the gazelle. Two of their conclusions are germane to our research. The first applicable conclusion is that restricted breeding promotes specialization better than free breeding. The difference between free and restricted breeding in the co-evolution of separate populations is that free breeding would allow an individual from population one to breed with population two. A second conclusion was to consider the whole team as a single GP individual.

Though this method proved to be successful, we feel that evolving team members in separate populations would further promote specialization. Learning control of each member of the team separate population play 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 making the best individual to do the task. The difficulty with this approach is that it is hard to determine the fitness of any individual in population one without it being overly influenced by the partner (from population two) that it is teamed with during the trial. Randomly picking partners could result in the loss of a good team one solution due to a poor team two solution. The best way to avoid this would be to have each individual from population one join with every individual from population two to attain its fitness. The difficulty with this is that there would be $2 \times n^2$ trials for each generation of training.

A solution for this problem was developed by Potter and De Jong [7]. This method, called cooperative coevolutionary algorithms (CCAs) allows heterogeneous control systems to evolve by defining a means of computing an individual's fitness without testing it with all of the other population's individuals. It computes each individual's fitness when teamed with the best individual from the other population. Best is determined by the individual's fitness compared to others in its own population. Potter, Meeden, and Shultz [6] used this method of co-evolution to learn the cooperative behavior of several shepherd robots attempting to herd a sheep into a corral. The results showed that this was a successful way of learning cooperative behavior in robots.

Although successful in completing the task and an excellent method for evolving cooperative teams, the CCA method still limits each individual's fitness calculation to being computed with a single partner. Although the individuals from population one are being evaluated with a partner that is currently considered the most fit of population two, this partner's relative fitness is based purely on the "most fit" of population one. Wiegand, Liles, and De Jong [8] show that increasing the number of partners in each fitness calculation improves the results, but can significantly increase the overall computation time. In this paper, we suggest a method that will allow a single partner to better represent the whole of the population. It is based on the concepts of punctuated anytime learning, which were developed in earlier work [3, 5]. This method uses periodic fitness calculations involving the entire population to determine each population's partner. The idea is to find a single individual that can model the nature of the other population. Our results show that this method is very successful at co-evolving separate simulated robots to do a cooperative task.

PROBLEM DESCRIPTION

The task 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 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 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.

Simulation of Robot Performance

The robots simulated in the experiment are ServoBots. These are inexpensive hexapod robots with two servos per leg, one oriented in a vertical capacity and the other oriented in a horizontal capacity, giving two degrees of freedom per leg (Figure 1).

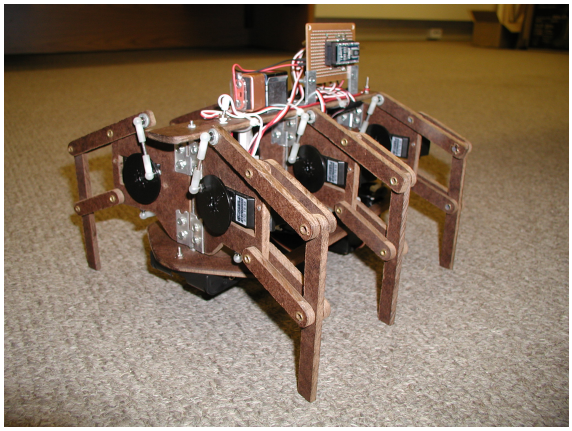

Figure 1: The ServoBot.

The ServoBot is controlled by a BASIC Stamp II that is capable of individually addressing each of the twelve servo actuators (two on each leg) 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 motion. The BASIC Stamp is capable of storing a sequence of timed activations to be repeated. A 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 indicating full back and 1 indicating full forward. 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). The number 001000000000 would lift the front left leg up, and 000001000000 would pull the second right leg backward. Each activation is held by the controller for one pulse (approximately 25 msec).

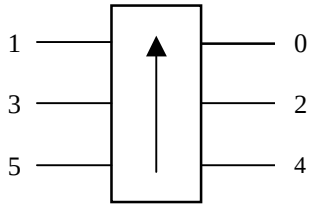

Figure 2: Graphic representation robot with legs 0 to 5.

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 [4]. 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 . The cycle took 29 pulses to complete.

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 [4]. 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 31 performance values (measured in centimeters moved and degrees turned) were recorded and stored in a table.

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. Both robots were represented as circles with a diameter of 6 units but the robots were treated as single points for rules of contact with the box. The box was represented as a square measuring 18 x18 units. 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.

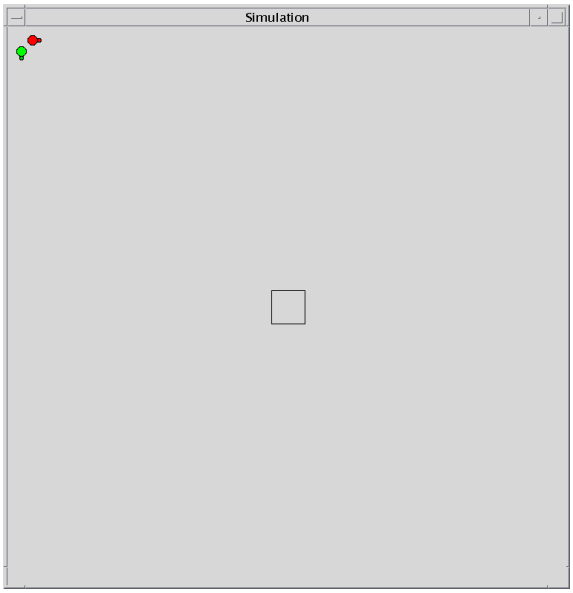

Figure 3: Simulation of the colony area.

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.

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.

Cyclic Genetic Algorithm

A type of evolutionary computation called a cyclic genetic algorithm [4] was used to develop our two heterogeneous cooperative agents. A CGA is much like a regular GA, but in the CGA the chromosome can represent a cycle of tasks. These tasks can be anything from a single action to a sub-cycle of tasks. 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 repeated. The genes can be arranged into a repeated sequence and a chromosome can be arranged with single or multiple cycles. 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. The evolution was done in two different stages. The incremental learning approach was employed because the problem can be easily broken down into two smaller tasks with the first requiring 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 was more complex and involved the use of punctuated anytime learning, which will be discussed later in this section, in addition to the CGA. The fitness score of a team was decided by the product of the box's final (x,y) coordinate position, or zero if the team failed to move the box toward the target corner of the area in the x or y position from the box's starting point (125,125).

The CGA was perfectly fit for evolving our agents 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, as soon as a robot touches the box, the controller would switch from the first to the second cycle, at the completion of the current gait cycle. The CGA chromosome had two cycles contain-

ing 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.

Punctuated Anytime Learning

Punctuated anytime learning (PAL) is a concept developed to allow offline genetic algorithms to capitalize on the strength Greffenstette and Ramsey's dynamic anytime learning approach [1]. Although PAL does not allow for continuous updates of the computer's models, it up dates its model every *n* generations, resulting in a period of accelerated learning. The generations in which the model is updated is referred to as a "punctuated" generation. This concept, when applied to a single GA, is that at every *n* generations the computer runs tests on the actual robot and uses these results for fitness biasing in the GA [5] or in the co-evolution of model parameters [3]. Punctuated anytime learning is a fairly different concept when applied to co-evolving two robots. The updated information that each population in the learning receives is a more accurate representation of the overall nature of the opposing population. For ease of explanation, assume that the experiment has two populations, population A and population B. In this case, every *n* generations, all individuals in population A are tested against all individuals in population B. The purpose of this process is to find the most fit individuals 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 best method of evolution would therefore be to select new alpha individuals for each generation. However the process of selecting the two alphas requires significant computation. Assuming there exists *k* individuals in each population the computer must perform k^2 trials for each population. In order to avoid that level of computation, new alpha individuals are only selected at certain consistently spaced periods of time. 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 individuals for fitness evaluation.

In order for a genetic algorithm to produce optimal results for a cooperative task, the fitness evaluation must be capable of recognizing the most specialized individuals in all populations. This process is impeded by the method of placing all members of a team into a single chromosome and evaluating the team's fitness in one trial. This method hinders the identification of specialized individuals because even the most fit agent can not ac-

complish part of or the whole task by themselves. Imagine that this method of evolving a team was applied to our problem, thereby a single chromosome would contain two individuals one from each population. Let's assume that an individual B exhibited highly specialized behavior, but individual A is an inadequate partner and immediately runs out of bounds in the beginning of the trial. The team would be given a score of zero because individual A stopped the simulation before the highly specialized individual B could even reach the box. Therefore, the extremely fit gene sequence for individual B has very little to no chance of being selected for crossover. If populations represented each member of the team and co-evolution was performed using the punctuated anytime learning method, the individual B's specialization would be recognized because it would be partnered with the most highly specialized partner possible. The punctuated anytime learning approach allows for the GA to evaluate each individual's fitness with the most specialized partner possible.

RESULTS

In order to test the method, five different tests were run for five thousand generations each. The fitness scores plotted on the graph (Figure 5) represent the team of two from that generation which received the highest fitness score. The fitness scores for the elite team were output at generations 0,50,200,500,1000,2000,5000. Those points are plotted in Figure 5 with the average curve represented in bold. The fitness score for a team is the product of the box's final coordinate position (x,y) minus the product of the box's starting coordinate position. Therefore, the largest score possible is 125^2 which is 15,625. The graph shows that the method was effective in learning the task and all but one of the five tests reached a fitness of 14,000 by generation 5000.

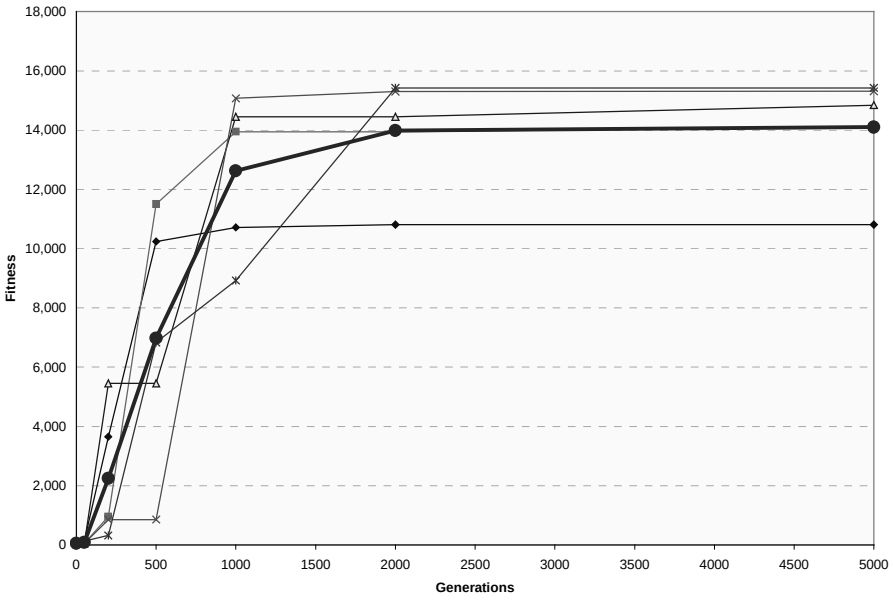

Figure 5: Results of box pushing task. The fitnesses of the best team for each of the five trials are shown. The average is in bold.

We observed the solutions at generation five thousand for each of the five tests. All five teams exhibited cooperative behavior at many different times throughout the simulation, but no test evolved a team in which both robots simultaneously pushed the box for the entire duration. We believe that this is due to a difficulty in timing. Some gait cycles are faster than others so the robots cannot coordinate their speeds. Inevitably a robot slips off the face of the box and it is hard for its partner to adjust. The solution produced by the evolution is for one to continue to push the box, while the other makes a circle and regains timing. In future work, we could rectify this problem by creating a speed variation for each gait cycle by having the controller send all the legs an equal amount of fewer pulses than each would normally receive.

CONCLUSIONS

Using punctuated anytime learning is an effective tool for co-evolving heterogeneous agents to do a cooperative task. It worked well in a simulation that modeled the behavior of actual robots. Other tests in two agent cooperative tasks should continue to show its effectiveness. Although not computationally excessive with two agents, this method's performance will decrease significantly with the addition of more members in the team. One solution for this is to not use the entire population at each punctuated step, but use a sampling of 5 or 10 individuals randomly selected from the other populations. With the correct sample size, the alpha individual will still accurately represent the population yet require less computation time to determine. Future work will include tests on the actual robot and comparisons of this method with single chromosome team learning and cooperative coevolutionary algorithms.

REFERENCES

1. Grefenstette, J. J. and Ramsey, C. L. "An Approach 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. "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).
4. 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).
5. Parker, G. B. and Larochelle, K. J. "Punctuated Anytime Learning For Evolutionary Robotics." *Proceedings of the World Automation Congress (WAC2000), Volume 10, Robotic and Manufacturing Systems*. (2000), 268-273.
6. 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).
7. 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.
8. 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.