The Co-Evolution of Model Parameters and Control Programs in Evolutionary Robotics

Gary B. Parker
Department of Computer Science
Connecticut College
New London, CT 06320
parker@conncoll.edu

Abstract

Evolutionary robotics is a research area that makes use of the various forms of evolutionary computation (EC) to provide a means of designing robot control systems. The use of EC can reduce development effort and allow the system to be adaptive to changes. However, it can be computationally expensive enough to preclude on-line learning since most forms of EC require that a population of possible solutions be tested over several iterations. Current research deals with to what extent these tests should be done on a model of the robot. Using the model is faster, but linking the model to the actual robot is a difficult task. In this paper, we introduce a new way of integrating the actual robot and its model during evolutionary computation. This method, which involves the co-evolution of model parameters, is applied to the problem of learning gaits for hexapod robots. The form of evolutionary computation used is the cyclic genetic algorithm (CGA), which was introduced in previous work to deal with the issue of evolving controllers for cyclic behaviors. Tests done in simulation show that the CGA operating on the co-evolving model of the robot can adapt to changes in the robot's capabilities to provide a system of anytime learning.

1 INTRODUCTION

In order to be autonomous, a robot should have its primary controller on board. Pre-programming this controller, although it can ensure the desired results, can be laborious during development and does not provide a means for real-time adjustments necessitated by miscalculation or degradation of the system. Learning control through some form of evolutionary computation can save man-hours of development plus provide the adaptability required for autonomy, but it can be too computationally intense to be carried out on board the robot. This is especially true when the robot is small with an on-board controller that is purposely simple to accommodate weight and expense restrictions. A system of learning to make adjustments to the on-board controller needs to be employed that can be carried out off-line yet allow the robot to adapt to changes in real time.

Evolutionary robotics is a research area that uses evolutionary computation (EC) to design robot control systems. Most forms of evolutionary computation require that a population of possible solutions be tested over several iterations. A significant issue of current discussion is whether or not these tests should be done on simulations (models of the robot). Three general approaches have been published. One is to do all of the training in simulation and transfer the results to the actual robot when it is complete [1,2,6]. This technique requires that significant attention be paid to the model as its accuracy directly effects the results. The time and effort can sometimes exceed the work required to program by hand. A second method is to do most of the training in simulation and transfer training to the robot for some remaining generations [7,9]. This requires a less accurate model (yet one is still required), but it can take significant time to do the on-line training on the actual robot. The third method does all of the training on the robot [4,10]. If the task can be completed and the fitness can be accurately judged in minimal time, this can be a viable option. It precludes the need for any model of the robot. The problem is that the training takes *num-generations* * *num-individuals* * *time-to-complete-task* to do the training. An increase in the *time-to-complete-task* results in a multiplicative increase in training time. The difficulty with all of these techniques is that we either have to put time into the model or into the training on the actual robot. In addition, none of these methods can be refining the solution while the robot is in operation unless it is only doing the task being learned. What is needed is dynamic simulator accuracy; a way of coupling the simulator to the actual robot [4,8,11].

Anytime learning was used by Grefenstette [3] to allow a learning component to continually compute a best solution while the robot operated in the environment using the latest best solution. A natural extension of this for simple robots is to have the learning component off-line while the operations component, which is on the robot, receives periodic downloads of the best solution. Grefenstette's system can adapt quickly to changes in the robot by having the robot's sensors continually update the status of the robot's capabilities in the learning element's simulation. A difficulty develops when we have simple robots, where the learning system must depend on global observation, instead of internal sensors, to determine the robot's capabilities.

In this paper, we introduce a new way of integrating the actual robot and its model during evolutionary computation. This involves the co-evolution of a gait control program and the model parameters of the model used in gait evolution. Tests done in simulation show that the cyclic genetic algorithm (CGA) operating on the co-evolving model of the robot adapts to changes in the robot's capabilities to provide a system of anytime learning.

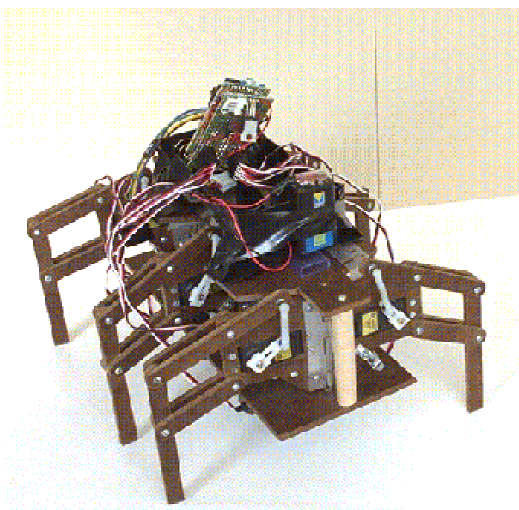

Figure 1: ServoBot.

2 ROBOT

2.1 ServoBot

Although the learning system with overhead observation and communication capabilities was not yet in operation and had to be simulated, the robot simulation was modeled after a robot that had been tested in previous gait experiments [5,13,14]. The ServoBot (Figure 1) is an inexpensive hexapod robot that has two degrees of freedom per leg. Twelve servos, two per leg, provide thrust and vertical movement. A control sequence is transmitted to a field programmable gate array (FPGA) through lines that connect to a Sparc workstation. Once the control sequence is transmitted, the line can be disconnected. The FPGA can store and execute (repeating the designated section) the sequence of primitive instructions downloaded. Each instruction corresponds directly to an activation that is activated for 50 ms, then the next instruction in the sequence is activated.

An input to the ServoBot is a 12 bit number where each bit represents a servo. A signal of 1 moves the leg back if it is a horizontal servo and up if it is a vertical servo. A signal of 0 moves it in the opposite direction. The activation can be thought of as six pairs of actuations. Each pair is for a single leg with the first bit of the pair being that leg's vertical activation and the second being that leg's horizontal activation. The legs are numbered 0 to 5 with 0, 2, & 4 being on the right from front to back and 1, 3, & 5 being the left legs from front to back. The activation 100101101001 results in one phase of the classic tripod gait, which is considered to be the optimal gait for speed in this simple rigid robot when all its actuators are fully functioning. Legs 1, 2, & 5 are providing thrust while legs 0, 3, & 4 are being lifted. Some sequence of these activations will result in a pattern of leg movement that will produce a viable gait.

2.2 Model

The model was a simple data structure that held each leg's capabilities and current state (Figure 2). Input activations were used with these capabilities to determine how much to change the current state of the model during training.

Measurements to fill the fields for each leg's capabilities were taken before training. Each leg's vertical position was measured relative to this initial zero position, which was on the ground. Each leg's horizontal position was measured relative to its at rest full forward position. Measurements to fill the maximum position fields of the model were taken by activating each control on the actual robot while recording the leg's maximum throw. An average rate per activation was calculated for horizontal and vertical movement by dividing the maximum throw by the minimum number of activations required to attain it.

```
Fields specific for each leg:
   current up -- current vertical position of the leg.
   max up -- position off the ground when completely up.
   current back -- current horizontal position of the leg.
   max back -- posit relative to completely forward when
               completely back.
Fields applicable to all legs:
   rate up/down -- rate of vertical movement when  servo
               activated.
   rate back/forward -- rate of horizontal movement when
               servo activated.
```

Figure 2: ServoBot model data structure.

3 CYCLIC GENETIC ALGORITHMS

Cyclic genetic algorithms (CGAs) were developed [12] to allow for the representation of a cycle of actions in the chromosome. The contents of this chromosome can be directly downloaded into the ServoBot's controller to produce a gait. Previous work [12,13,14] discusses the details of this type of evolutionary computation and its success in evolving gaits in simulation that were transferable to the actual robot. Further work needed to be done to make adjustments when necessary to accommodate for dissimilarities in the simulation and actual robot, but in general the CGA was very successful at evolving near optimal gaits for both fully functional and partially disabled robots.

4 CO-EVOLVING MODEL PARAMETERS

Previous work with CGAs, although promising for producing gaits in a static environment, would require the addition of on-board capability sensors to be used dynamically. In addition, preparation was time consuming as each of the robot's capabilities had to be accurately measured to be recorded in the model. Some means of dynamically linking the model to the actual robot could alleviate both of these problems. The method introduced here is to do periodic tests of evolved gaits on the actual robot to co-evolved the accuracy of the robot's model with the CGA produced gait. It is an alteration of anytime learning (Grefenstette [3]).

4.1 Anytime Learning

There are two main issues we address concerning the development of a system for anytime learning in evolutionary robotics; anytime learning without sensors and interactively connecting simulations to actuality to provide efficient ER training. A solution for both of these issues involves the use of anytime learning with modifications to compensate for the lack of internal sensors in our robots. Training with a GA will take place off-line on a simple model. Periodic checks on the actual robot will help to verify the model's accuracy.

In order to understand the use of anytime learning in evolutionary robotics let us look at what is taking place during training. Figure 3 shows the three main parts of the system. There is the actual robot, the model, and the genetic algorithm. The model is designed before training to be as accurate as possible (depending on the simplicity desired). The GA is set up for training and initiated with a random starting population. The program in the controller of the actual robot is determined by the other two components of the system. The learning system cannot alter the physical capabilities of the actual robot, only the program in its controller (which is the purpose of the system). The periodic checks on the actual robot can alter the model in an attempt to improve the result of training.

This will involve the co-evolution of the model parameters and the controller program. After each n generations the best, worst, and a random solution, found on the model, can be tested on the actual robot. Using only outside observation of the results, measurements of the distance moved will be taken. These measurements will be used to judge the accuracy of the population of model parameters. This population can start out either as randomly generated individuals or as a combination of perturbations (to varying degrees) of the original model parameters. The most accurate model parameter will be used for the controller evolution, the population of model parameters will continue to evolve until interrupted by updated actual test information. This solution requires three actual tests every n generations.

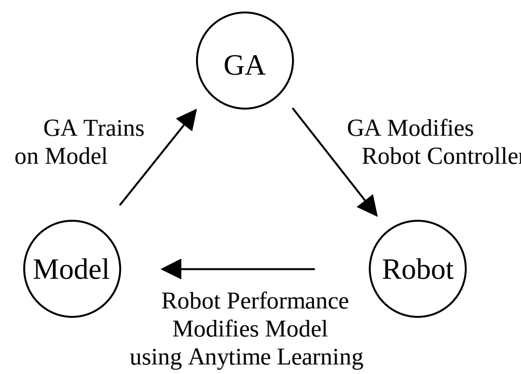

Figure 3: Anytime learning in evolutionary robotics.

The form of evolutionary computation used to co-evolve the model parameters is a basic genetic algorithm. A population of 64 individuals is randomly generated before training begins. Each individual is made up of 12 genes, which have a size of 6 bits. A sample chromosome is shown in Figure 4. Each gene represents a corresponding field in the robot's model. The first 6 genes represent the vertical maximum movement for each of the six legs. The last six represent the maximum horizontal throw of each of the six legs. These genes evolve to produce models that correspond in performance to the actual robot. Although the model's leg rates are held constant, the fitness computation is done in such a way that the adjustment of the max throw fields can make the required adjustments to compensate for inaccurate rate measurements.

(011111 110011 101111 001111 111000 111011
101010 110011 100111 101101 011111 101110)

Figure 4: Model chromosome.

The fitness is computed by selecting 3 representative gaits from the CGA's gait population. The best (using the current best robot model), a random, and the worst performing gaits are selected. Each of these 3 is run on the actual robot, which is simulated by a model that more accurately represents the robot at this point in our research, and the total distance traveled is recorded. This is designed to require only an external observation by an overhead camera of distance traveled to work in the actual robot. These 3 distances are used to evolve a more accurate model. Each chromosome of the model population is used to compute its fitness using the following algorithm:

```
R1 = Test best gait on the robot
R2 = Test random gait on the robot
R3 = Test worst gait on the robot
For each individual in the model population
    M1 = Get fitness using current individual and best gait
    M2 = Get fitness using current ind and random gait
    M3 = Get fitness using current ind and worst gait
    Order-Same = True if the fitness order of R1, R2, & R3
                 is the same as M1, M2, & M3
    Signs-Same = True if Rx & Mx have the same sign for
                 x = 1 to 3
    Fit-Comp = Fitness of the robot to model comparison
    If neither Order-Same or Sign-Same are True
      Return  .001
    If one only of Order-Same or Sign-Same is True
      Return  .001 x Fit_Comp
    If both are True
      Return Fit_Comp
```

*M*1 is the model's fitness when run using the best gait from the CGA, *M*2 is the model's fitness when run using the random gait, *M*3 is the worst gait fitness. The *R*1, *R*2, and *R*3 are the robots fitnesses (or distance) traveled using the corresponding gaits. The value for *Fit-Comp* is computed using the following equation:

$$\frac{1}{\left\|\left|\frac{M1-R1}{R1}\right|-\left|\frac{M2-R2}{R2}\right|\right\|}+\frac{1}{\left\|\left|\frac{M1-R1}{R1}\right|-\left|\frac{M3-R3}{R3}\right|\right\|}$$

*R*1, *R*2, and *R*3 are changed to equal 1 if their absolute value is less than 1.

The purpose of this equation is to compute how closely the robot and model match while disregarding the differences in rates. In other words, if the robot's rate is 3 cm of movement per 50 ms and the max throw is 45, the corresponding model can have a rate fixed at 2 cm per 50 ms and adjust its max throw to equal 30. In each case, 15 activations will move the leg full throw. Three gaits to compare were chosen to increase accuracy over two yet not require too many tests on the actual robot.

After a fitness is computed for each individual, selection is done stochastically, giving all individuals a chance for reproduction, but favoring the most fit. Once a couple is selected, they produce a single offspring through crossover and mutation. This is done 64 times to produce an entirely new population.

Parents

(000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000)

(111111 111111 111111 111111 111111 111111 111111 111111 111111 111111 111111 111111)

Offspring

(000000 000000 000000 000000 111111 111111 111111 111111 111111 111111 111111 111111)

Figure 5: Between gene crossover.

Crossover is accomplished in two ways. One is through single point crossover between genes. An example is shown in Figure 5. This allows for the mixing of the chromosome's genes while allowing established gene schema to stay intact. The second method of crossover, crosses bits within each gene. This method is shown in Figure 6. It helps alter individual genes to gain fitness plus allows crossover that is not dependent on gene ordering. Crossover is always used in forming each new individual; either type of crossover can be used (equal probability).

Parents

(000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000 000000)

(111111 111111 111111 111111 111111 111111 111111 111111 111111 111111 111111 111111)

Offspring

(001111 000001 000111 001111 000000 011111 000001 000111 000011 001111 111111 000111)

Figure 6: Within gene crossover.

Mutation also has 2 variations. One that causes an entire gene to be replaced by a new gene. The other changes only an individual bit within a gene. Each had a 1/200 chance of changing each gene of the chromosome.

The co-evolution of model parameters is designed to be used in either of two ways. One is to have both the CGA and the model parameter evolution to be operating concurrently on separate processors. The second is to schedule alternating evolution of both GAs on the same processor. They should both yield similar results although the former would be faster. In our experiments we used the later as we were working with a single processor.

The system started out by producing a population of gaits using the CGA applied to the initial model. This model was considered to be a rough estimation of the robot. After 500 generations of CGA only training, co-evolution of model parameters began. A random start population of models was produced and checked against the robot as described previously. They were allowed to evolve for three generations. If, after this time, one was superior to the initial model, it replaced it as the current model. The CGA was then allowed to run for 10 generations using the current model to produce a new population of gaits. The model and gait are constantly changing to keep up with dynamic changes in the robot's capabilities. This method of alternating evolution was used to allow experimentation without worrying about parallel processing. When the actual learning environment is constructed, the processes can be running concurrently.

Changes to the actual robot controller took place at every 50 generations. If the current model's performance on the robot was superior to that of the one selected as the robot's controller (this value is stored in the learning system), they would be switched. If it was inferior, the robot would be re-tested with its selected controller to determine which was the best. In this way, robot performance constantly improves or stays the same, unless there is a degradation of capabilities. This could, in the worst case, add an one more actual robot test every 50 generations. In the average case, this should add very few extra tests since the only time an extra test will be required is when there is a reduction in the robot's capabilities.

5 TESTS

Three different tests were used to verify that co-evolving model parameters for anytime learning was an effective way to improve robot performance during EC. Each test involved training on the model, with tests on a simulation (model of specified parameters) of the robot. The distance traveled by the simulated robot after 10 s of activation of the current gait was recorded in cm. Each test was done using 5 random start populations. The results reported are the average performance of these five for each test.

The first test used a generic initial model. This model had all the settings the same; a general guess at the robot's performance. This test was to verify that the initial measurements discussed in section 3 would no longer be required with the addition of anytime learning. The second test used a robot with rates that were 2 and 3 times faster than those used in the model. This was to verify that the system could automatically adjust to differences in the model's and robot's rates of leg movement. The third test was a series of changes to the robot's capabilities that took place while the system was running. Observance of the robot's (simulated) performance showed how well the learning system reacted to these changes.

6 RESULTS

The first test used a starting generic model of all 32s (100000). The CGA was allowed to run for 500 generations and, at that time, the co-evolution of model parameters began. The robot's simulated distance traveled in 10 seconds was 59.7 cm at this time. After 200 generations, the system had sufficiently adjusted the model's settings to gain what looked like an optimal gait at 69.6 cm traveled.

The second experiment was equally successful. Experiments were done with robots having rates of two and three times that which was used in the model. The 2x experiment resulted in an improvement from 126.0 to 153.6 after 200 generations. The 3x experiment resulted in an improvement from 189.0 to 229.8 after 200 generations. In both cases the gait produced looked optimal and the system adjusted appropriately for the differing rates.

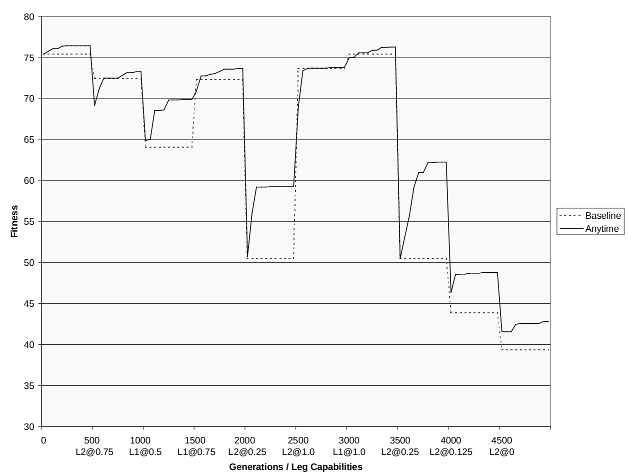

Figure 7: Adaptability of co-evolving model parameters.

Figure 7 shows a graph of the adaptability test. The robot is initially trained with a slightly inaccurate model. This results in a robot fitness of 75.3 as can be seen at generation 0. At this point the co-evolution begins. By 500 CGA generations the fitness has improved to 76.3 (dynamically correcting to the initial inaccuracies). After each subsequent 500 generations, there is a change in the robot's capabilities. The horizontal movement maximum throw of either leg 1 or 2 is changed. The graph shows the change by indication the fraction of original capability for the specified leg. At 500 generations leg 2's max horizontal throw is reduced to .75 of its original. At 1000 generations leg 1's max horizontal throw is reduced to .5 of its original, leg 2 stays at .75. This continues throughout the test. The dotted line shows what the results would be without a dynamically evolving model.

Some interesting observations can be made from this graph. The anytime learning always improves the simulated robot's performance over time. The largest improvements are in the mid capability ranges (around .5). In the upper ranges, the static model is still close, and in the lower ranges, there just isn't a lot that can be done to improve performance. It can also be observed that the system with anytime learning almost always outperforms the system without. The one notable exception is at 500 generations where it was below the baseline for 100 generations after leg 2 went to 75%. The final observation is that although the anytime system sometimes drops significantly after a degradation in robot capabilities, it quickly recovers to improve performance.

7 CONCLUSIONS

Tests done in simulation show that the CGA operating on the co-evolving model parameters of the robot can adapt to changes in the robot's capabilities to provide a system of anytime learning. This system of modified anytime learning, which requires only external observation of the robot's performance, can be an effective means of coupling the performance of the model to the robot during evolutionary computation. Tests in simulation do not always prove a learning system's viability on the actual robot, but we are confident, due to extensive previous research using this model and robot, that this learning method will be successful on the ServoBot.

Future research will be to introduce a population resetting scheme such as the one described in Grefenstette's [3] work to help improve performance when there are large levels of capability degradation. Also planned, after construction of a learning environment with and overhead camera for observation, is to do actual tests on the ServoBot.

Acknowledgments

This research, performed at Indiana University's Adaptive Systems Laboratory, was supported in part by NSF Graduate Research Traineeship Grant GER93-54898.

References

1. Beer, R. D. and Gallagher, J. C. (1992). "Evolving Dynamical Neural Networks for Adaptive Behavior." *Adaptive Behavior*, 1 (pp. 91-122). Cambridge: MIT Press.

2. Gallagher, J. C. and Beer, R. D. (1994). "Application of Evolved Locomotion Controllers to a Hexapod Robot." Technical Report CES-94-7, Department of Computer Engineering and Science, Case Western Reserve University.

3. Grefenstette, J. J. and Ramsey, C. L. (1992). "An Approach to Anytime Learning." *Proceedings of the Ninth International Conference on Machine Learning*.

4. Husbands, P., Harvey, I., Cliff, D., and Miller G. (1997). "Artificial Evolution: A New Path for Artificial Intelligence?" *Brain and Cognition* v. 34, (pp. 130-159).

5. Johnson, S., Parker, G., Cyliax I., and Braun, D. (1997). "Using Cyclic Genetic Algorithms to Reconfigure Hardware Controllers for Robots." Indiana University Computer Science Department Technical Report No. 494.

6. Lee, W.-P., Hallam, J., and Lund, H. (1997). "Applying Genetic Programming to Evolve Behavior Primitives and Arbitrators for Mobile Robots." *Proceedings of IEEE 4th International Conference on Evolutionary Computation*.

7. Lund, H., and Miglino, O. (1996). "From Simulated to Real Robots." *Proceedings of IEEE 3rd International Conference on Evolutionary Computation*.

8. Mataric, M. and Cliff, D. (1996). "Challenges in Evolving Controllers for Physical Robots." *Evolutional Robotics, special issue of Robotics and Autonomous Systems*, Vol. 19, No. 1, October 1996, (pp 67-83).

9. Miglino, O., Lund, H., and Nolfi S. (1995). "Evolving Mobile Robots in Simulated and Real Environments." Technical Report, Institute of Psychology, C.N.R., Rome.

10. Mondada, F. and Floreano, D. (1995). "Evolution of Neural Control Structures: Some Experiments on Mobile Robots." *Robotics and Autonomous Systems*, 16, (pp. 183-195).

11. Nolfi, S., Florano, D., Miglino, O., Mondada, F. (1994). "How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics." *Proceedings of the International Workshop on the Synthesis and Simulation of Living Systems*, (pp. 190-197).

12. Parker, G. and Rawlins, G. (1996). "Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots." *Proceedings of the World Automation Congress (WAC'96), Volume 3, Robotic and Manufacturing Systems*. (pp. 617-622).

13. Parker, G., Braun, D., and Cyliax I. (1997). "Evolving Hexapod Gaits Using a Cyclic Genetic Algorithm." *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing (ASC'97).* (pp. 141-144).

14. Parker, G. and Cyliax I. (1998). "Locomotion Control Cycles Adapted for Disabilities in Hexapod Robots." *Proceedings of the World Automation Congress (WAC '98), Volume 7, Robotic and Manufacturing Systems*. (pp. 359-364).