Adaptive Hexapod Gait Control Using Anytime Learning with Fitness Biasing

Gary B. Parker
Department of Computer Science
Connecticut College
New London, CT 06320

Jonathan W. Mills
Department of Computer Science
Indiana University
Bloomington, IN 47405

Abstract

Adaptive learning systems that generate control programs for robots with varying capabilities is of importance in the implementation of autonomous robots. Learning done continuously with the best possible control program running the robot (anytime learning) can achieve the adaptability desired when implemented using some form of evolutionary computation. The difficulty with this method is that autonomous robots often lack the computational power required to run evolutionary computation along with their control program. In addition, anytime learning usually requires input from internal sensors, which are not often available in small autonomous robots, to make adjustments for capability changes. In this paper, we propose an anytime learning system that employs off-line learning, using evolutionary computation, with the control program being downloaded to the on-line controller. The off-line learning does not require internal sensors but uses global observation (external overhead camera) to make the required adjustments to guide the evolutionary computation. The results of periodic tests, done on the actual robot, are used to bias the fitnesses calculated by the evolutionary computation, which uses a model of the robot. Experiments reported in this paper use a simulation of the actual robot (a more accurate model), while construction of the actual learning system is in progress.

1 INTRODUCTION

Autonomous legged robots have distinct control issues related to the coordination of separate legs and their varying capabilities. These issues are often amplified when the robot is small with an on board controller which is purposely simple to accommodate weight and expense restrictions. Autonomy requires that the robot's primary controller be on board. Pre-programming this controller can ensure the desired results, but 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. A system of learning that can be carried out off-line and then downloaded to the on board controller will allow the robot to adapt to changes in real time.

Anytime learning can make use of evolutionary computation in a learning module to make adjustments to the robot's control module. In Grefenstette's work [1992], evolutionary computation was used to continually adjust the controller while the robot operated in an environment where its target to catch had its capabilities periodically change. A natural extension of this for small autonomous 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 could adapt quickly to changes by having the robot's sensors continually update the capabilities of the target model used in the learning element. Our system can adapt to changes in the robot's capabilities without the use of internal sensors. Depending only on global observation, this learning system will use evolutionary computation and anytime learning to train the robot.

Most forms of evolutionary computation require that a population of possible solutions be tested over several iterations. This training can be done on a model of the robot, entirely on the robot, or a combination of the two. If all of the training is done off line and the results transferred to the actual robot when it is complete [Beer 1992, Gallagher 1994, Lee 1997] then significant attention must 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. If most of the training is done off line and then transferred to the actual robot for some remaining generations [Lund 1996, Miglino 1995], then a less accurate model is required, but it can take significant time to do the on line training on the actual robot. If the task can be completed and the fitness can be accurately judged in minimal time, all of the training can be done on line [Husbands 1997, Mondada 1995]. This method precludes the need for any model of the robot, but the training takes *num-generations * num-individuals * time-to-complete-task* to do the training. An increase in any of these results in a multiplicative increase in training time. All of these techniques require that we either put time into the model or into the training on the actual robot. In addition, none refine the solution while the robot is in operation unless it is only doing the task being learned. What we propose is to use continual anytime learning with a way of coupling the simulator to the actual robot.

In this paper, we introduce a new way of relating the actual robot to its model during evolutionary computation. This is done by biasing the solution fitnesses computed on the model by comparing their performance to those produced when using the actual robot. The task being learned is gait generation for hexapod robots. Previous work showed that Cyclic Genetic Algorithms (a form of evolutionary computation) could be used to effectively perform this task in a static environment. This paper reports that tests done in simulation show that a CGA based anytime learning system with fitness biasing can adapt to changes in the robot's capabilities to provide continually viable gaits.

2 ROBOT

2.1 SERVOBOT

The robot used was the ServoBot (developed by David Braun), which 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 gait 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 100ms, then the next instruction in the sequence is activated.

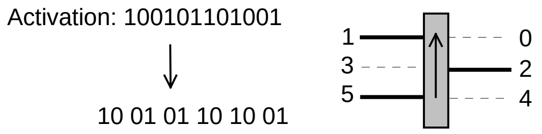

Figure 1: Results of a Robot Activation

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. Figure 1 shows an example of an activation and its result on the robot. The activation can be thought of as 6 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, as shown, 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.

Fields specific for each leg:

current up -- current vertical position of the leg.
max up --posit off the ground when completely up.
current back -- current horizontal position of the leg.
max back -- posit relative to completely forward
    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

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.

3 CYCLIC GENETIC ALGORITHMS

Cyclic Genetic Algorithms were developed [Parker 1996] to allow for the representation of a cycle of actions in the chromosome. They differ from the standard GA in that the chromosome is in the form of a circle with two tails (Figure 3) and the genes can represent tasks that are to be completed in a predetermined segment of time. The tails of the CGA chromosome are provided to allow for pre and post-cycle procedures if required; a means for completing tasks before and after entering the cycle. They can be as simple as primitive tasks (activations) or they can be as complicated as cyclic sub-chromosomes that can be trained separately by a CGA. For our purposes, the genes represent a set of servo activations that are to be sustained for 100 msec each. The trained chromosome will contain the cycle of these primitive instructions that will be continually repeated by our robot's simple controller to produce a gait.

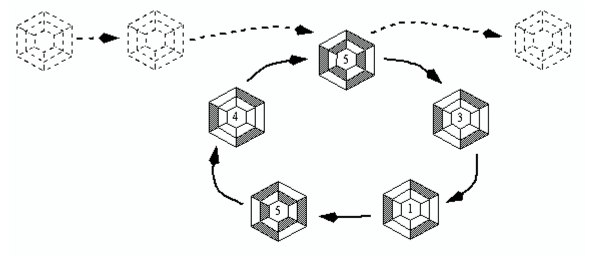

Figure 3: CGA Chromosome

The CGA chromosome can also incorporate parameters for control in its structure. Coordinators were 12 bit numbers that directed the coordination of individual leg movement. These numbers could be looked at as six pairs of bits, one pair for each leg. The first being the back-down coordinator which, if activated, ensured that the leg would be down or moving in that direction if it was moving back. The second bit was the forward-up coordinator, which ensured that the leg would be moving forward if it was moving up. Inhibitors prevented pairs of legs from moving back at the same time. The 2,3 inhibitor prevented both legs 2 and 3 from going back at the same time. It allowed 2 to move back, but inhibited 3. The inhibitors for the set of legs were stored in a single 15 bit number (one bit per possible pair).

The genes of the chromosome had two parts. The activations part was a 12 bit number that contained the encoding required to activate two possible primitives per leg. A bit setting of 1 meant the servo was moving the leg back, a bit setting of 0 meant it was moving the leg forward. When the servo reached the full throw of the leg it would hold its full back position. Similarly, when the servo ran continuously forward the leg would stay at its full forward position. The activation for up and down worked in the same way. The *repetition* part was an 8 bit number that designated the number of times to repeat the activations part. This *repetition* part was what gave the CGA the ability to vary the length of the sequence of primitives being sent to the robot in each cycle.

The effectiveness of the CGA has been tested on a fully functional robot [Parker 1997] and partially disabled robots [Parker 1998]. In both cases, training was done on a model of the robot. Performance tests on simulations were done to determine the fitness of the resulting populations after training on random starting populations. These same gaits were also used to test their usability on the actual robot. The distance traveled by each robot during physical tests was recorded. CGAs were very successful at evolving what appeared to be optimal gaits for the robots tested.

4 ANYTIME LEARNING WITH FITNESSES BIASING

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. A system of anytime learning that can use global performance information to dynamically link the model to the actual robot could alleviate both of these problems. The method introduced here is to do periodic tests on the actual robot of the population of gait solutions evolved using the model. As a result of these tests, biases are assigned to each solution, which are used during further training to effect the solution's fitness.

4.1 ANYTIME LEARNING

There are two main issues that are addressed 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 evolutionary computation. 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 find the disparity between the model and the robot.

Figure 4 shows the three main parts of a system of anytime learning in evolutionary robotics. 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).

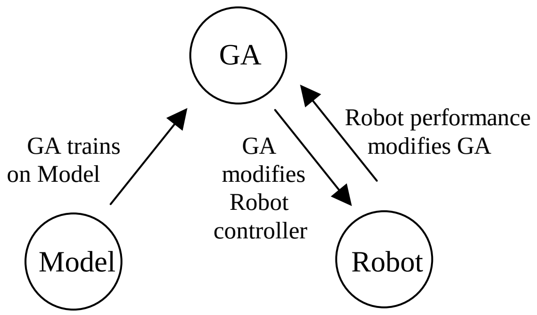

Figure 4: Learning in Evolutionary Robotics

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 physical capabilities of the actual robot cannot be altered by the system, only the program in its controller (which is the purpose of the system). The periodic checks on the actual robot can alter the processing within the GA in an attempt to improve the result of training. This will involve the biasing of evolved solution fitnesses by comparing their performance on the model to their performance on the actual robot.

4.2 FITNESS BIASING

The GA used in our learning system was the CGA since dynamic gait production was the goal. The means of using tests on the actual robot to modify the GA was called “fitness biasing,” which effected selection in the CGA.

Probability for selection was determined by computing each individual's fitness on the robot model. Fitness was computed one activation at a time by summing the fitnesses of individual activations as each was applied to the current state of the model. The fitness would then be reduced if the robot was out of balance or had asymmetrical movement (one side producing more thrust than the other). This was repeated from one activation to the next for each activation in the start section and continued in the iterative section until a total of 100 activations (during training) was reached. This fitness was computed for each individual in the population.

After each *n* generations all solutions of the CGA population would be tested on the actual robot. These measurements were used to bias the fitnesses found on the model to make them equal to the actual robot fitnesses. These biases were used as the CGA continued training. In this way, solutions that were good on the actual robot but poor on the model had boosted fitnesses during training, which resulted in their production of more offspring. This solution required *population-size* actual tests every *n* generations. In our experiments, the *population-size* was 64 individuals and *n* was set to 50 so there was a little more than one actual test per CGA generation.

A bias for each solution of the population's 64 was computed using the following algorithm:

```
Model-Fitness = Compute-Model-Fitness(Solution)
if absolute-value(Model-Fitness) < 1
  Bias = 1
else
  Actual-Fitness = Test-on-Robot(Solution)
  Bias = Actual-Fitness / Model-Fitness
```

This bias was stored with its corresponding solution. It was used in subsequent generations of the CGA to alter the fitness of the solution computed on the model of the robot. This was done by multiplying the fitness computed on the model by the bias:

Corrected-Fitness = Model-Fitness * Bias

These *Corrected-Fitness*es were used for selection during the training being done by the CGA.

4.3 GENETIC OPERATORS

Pairs of individuals were stochastically selected for reproduction using the *Corrected-Fitness*es. These two individuals produced a single individual for the next generation; combining their attributes by crossover with possible random variations caused by mutation. This new individual’s bias was computed by averaging the biases of its parents.

Crossover was accomplished by randomly picking corresponding spots in the two selected parents. In the precycle tail, a single point in both chromosomes was picked. In the cyclic section, since it could be considered a circle, crossover was performed at two points. The effect was to swap sections within the circle. An alternate type of crossover was a gene-by-gene crossover that performed crossover in each of the corresponding genes of the two chromosomes. Crosses could happen between the individual members of the list or within the bits of the specific numbers in the list.

There were two types of mutation used and selected randomly after each recombination; one in which each gene had a random chance of being replaced by a new completely random gene. The other was one were each part of the gene had a random chance of having one of its bits flipped.

Gene-by-Gene Evaluation, a genetic operator peculiar to CGAs was used to clean up the chromosome by randomly picking one or two individuals from the population on each set of trails and examining each gene one at a time. Genes were evaluated move-by-move by comparing the previous move fitness to the present. Genes that performed poorly in their current position were eliminated. Genes that were good in the execution of their early repetitions and subsequently dropped in the later repetitions were modified by reducing their repetitions. Genes that had zero repetitions were moved out so that only active genes were at the start of the cyclic section.

5 TESTS

Two tests were used to verify that fitness biasing was an effective means of improving robot performance during training. Each test involved training on the model, with tests on a simulation (more accurate model) of the robot. The calculated distance traveled by the simulated robot after 10 sec of activation of the current gait was recorded. 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 robots 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 second test was a series of varying changes to the robot's leg 1 and 2 capabilities to see if the system could continually compensate as capabilities came and went. Observance of the robot's (simulated) performance showed how well the learning system reacted in these four tests.

6 RESULTS

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 153.8 after 200 generations. The 3x experiment resulted in an improvement from 189.0 to 230.3 after 200 generations. In both cases the gait produced looked optimal and the system adjusted appropriately for the differing rates.

Figure 5 shows a graph of the adaptability test. The dotted line shows what the results would be without anytime learning. The solid line shows the average performance of the five stating populations. The error bars show the standard error at each sampling point. The robot was initially trained with a slightly inaccurate model. This results in a robot fitness of 75.4 as can be seen at generation 0. At this point the anytime learning with fitness biasing begins. By 500 CGA generations the average fitness has improved to 76.4 (dynamically correcting for 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.

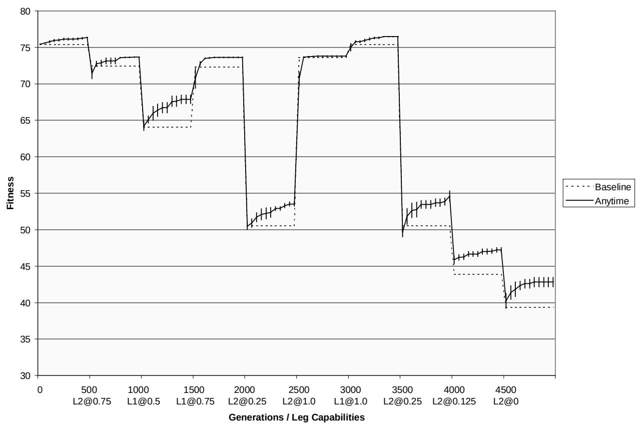

Figure 5: Adaptability of Anytime Learning Using Fitness Biasing. Average fitness of the five starting populations is shown. Standard errors are shown as error bars.

The graph shows the change by indicating 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. Figure 6 shows the fitness changes for all five starting populations.

Some interesting observations can be made from these graphs. The anytime learning always improves the simulated robot's performance over time. The largest improvements are in the mid to lower capability ranges (.25 to .5). In the upper ranges, the static model is still close to being accurate, so there is minimal improvement possible. In the lowest ranges, there is much more room for improvement, but the cycles produced start to resemble the baseline gait cycle.

The general solution to correcting horizontal movement limitations is for the learning system to decrease the duration of the strides. This results in somewhat less fitness since the initiation of the stride produces less thrust than when the stride is in full motion. At some point, stride length reduction starts to produce diminishing returns and it becomes advantageous to just drag a leg for a few activations. The CGA typically starts to increase the stride again. Although it loses total thrust by dragging a leg, it gains by decreasing the initiation time during the stride. The end result is that gaits produced with very low leg capabilities are more similar than mid leg capability gaits to the fully capable gaits. This results in less deviation from the baseline when leg capabilities are very low.

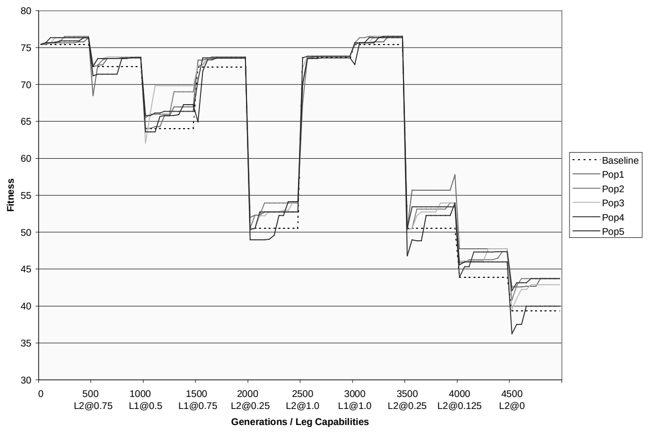

Figure 6: Adaptability of Anytime Learning Using Fitness Biasing. All five starting populations are shown.

It can also be observed that the system with anytime learning almost always outperformed the system without. The notable exceptions were when reduced capability changes initially occur. The anytime learning system, since it had optimized for the previous capabilities, often caused a drop below the baseline until it could readjust, which seldom took more than 50 generations.

7 CONCLUSIONS

Tests done in simulation show that anytime learning with fitness biasing can greatly improve a CGA's ability to dynamically adapt to changes in the robot's capabilities. This system of modified anytime learning, which requires only external observation of the robot's performance, can be an effective means of coupling the learning system 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 [1992] work to help improve performance when there are large levels of capability degradation. Also planned, after construction of a learning environment with an overhead camera for observation, is to do tests on the ServoBot.

Acknowledgments

This research was supported in part by NSF Graduate Research Traineeship Grant GER93-54898.

References

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

Gallagher, J. and Beer, R. (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.

Grefenstette, J. and Ramsey, C. (1992). "An Approach to Anytime Learning." *Proceedings of the Ninth International Conference on Machine Learning*, (pp. 189-195).

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).

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*.

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

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.

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

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).

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).

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).