Learning Area Coverage Using the Co-Evolution of Model Parameters

Gary B. Parker

Computer Science
Connecticut College
New London, CT 06320
parker@conncoll.edu
860-439-5208

Abstract

The type of search where a robot's track takes it on a path where its sensors can detect all mines located in the area is referred to as area coverage. Planning this track is an issue in robotics and is complicated when the robot is legged due to the reduced precision of its movements. Cyclic genetic algorithms have been used as a method for learning the cycle of turns and straights required for a hexapod robot to solve the area coverage problem. Although successful in a static environment, the learning system needed an anytime component to make it adaptable enough to be used in practice. This paper discusses the creation of a viable learning system by adding the anytime learning technique of co-evolving model parameters. Tests in simulation demonstrate this system's usefulness in generating search patterns despite changes in the robot's performance.

1 INTRODUCTION

Area coverage is a type of path planning that is concerned with the coverage of an area. Some applications are mine sweeping, search and rescue, haul inspection, painting, and vacuuming. Coverage is done by the robot's sensors /manipulators, which are assumed to have a certain width of effectiveness. The area to be covered is described as having defined boundaries and possibly some obstacles. The path planned is supposed to ensure that the area covered by the robot's sensors compared to the total area within the defined boundaries is equal to the desired coverage.

Research in the area of coverage path planning has concentrated primarily on covering a specified area while contending with obstacle avoidance. Zelinsky et al. (1993) used path planning by dividing the area into cells that were marked with the distance to the goal to form a cell to cell path through the area. Choset and Pignon (1997) divided the area into obstacle free sub-areas and found an exhaustive path through the adjacency graph representing these cells. Within each cell the back-and-forth boustrophedic motions (Figure 1) were used to assure coverage. Ollis and Stentz (1997) used vision to control the lines in their boustrophedic motions to do automated harvesting. Hofner and Schmidt (1995) used templates appropriate for the type of robot to determine the best path within varying sized areas. In addition to dead reckoning, landmarks sensed by ultrasonic sensors were used to maintain the desired track. Hert at el. (1996) used an on-line planar algorithm and sensors for an autonomous underwater vehicle to explore areas of the sea floor with arbitrary shape.

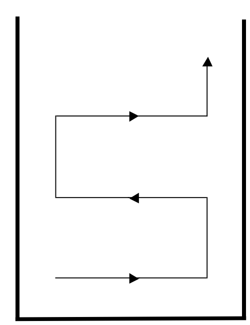

Figure 1: Back-and-forth Boustrophedic motion.

Common to all of these works is either precise control of the robot or some navigational means of finding its position/orientation and making continual corrections to its movement. These assumptions make them ineffective for inexpensive legged robots with minimal sensors and precision of movement since they cannot be positioned perfectly with exact headings. What is often taken for granted in these papers, a capability to perform perfect back-and-forth boustrophedic motions (Figure 1), is difficult for legged robots, since the exact time and rate of turn cannot be specified. In addition, often what is considered a straight gait results in a small drift to one side or the other due to performance differences from one side to the other of the robot. The best straight may actually be what is programmed in as a minimal turn. Efficiency is also a major factor in determining the best track over the ground to cover the area. The best path depends greatly on the capabilities of the robot. If the robot can efficiently rotate or turn sharply, its best strategy may be to do a ladder pattern (boustrophedic with square turns). If tight turns are not efficient for this particular robot, it may be better to make large sweeping turns or buttonhooks with some coverage overlap.

A method for learning turn cycles that will produce the tracks required for area coverage was introduced in previous work (Parker, 2001). The learning was done using a cyclic genetic algorithm (a form of evolutionary computation designed to learn cycles of behavior). Tests of the robot's dead reckoning capabilities with the learned cycles showed that using CGAs is an effective means of learning for area coverage. This provided a means of learning the optimal cycle of turns and straights that could greatly improve the efficiency of area coverage within cells. In addition, this system of learning could compensate for the lack of calibration in robot turning systems. Although successful in a static environment, the learning system needed anytime learning (Grefenstette and Ramsey, 1992) to make it adaptable to changes in the robot or its environment. This paper discusses the use of the co-evolution of model parameters to make the learning system useful in a dynamic environment. Tests in simulation demonstrate this system's success in generating search patterns despite changes in the robot's performance.

2 THE PROBLEM

During area coverage the robot is trying to maximize the area covered in minimal time. This may be done by an insect to search for food or check for enemies. A robot could also be searching for enemies or clearing the area of deadly devices such as mines. For the area coverage problem used in this research, the robot was to fully search, starting from a specific point (Figure 2), an area of specific width (180 cm). Since the robot was judged by the area covered in a set amount of time, plus the purpose was to find the most efficient cycles of behavior required to do it, the area to be searched had no bound on one side. The area width was purposely small to accommodate ease in actual testing and to force more turns during training.

The simulated search was for mines that would be fully contained in the area. In order to detect a mine, the robot had to have the entire width of its body (excluding the legs), at its mid point, within the same 60x60 cm square as the mine. For test purposes, 60x60 blocks with mines were placed to completely fill the area. The robot's task was to find as many mines as possible while ensuring that no mines had been missed. The robot's movement was not restrained in any way by the environment. There was no physical constraint requiring it to stay within the mine area.

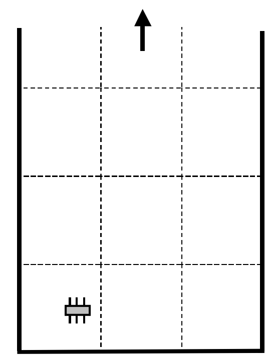

Figure 2: Search area for coverage.

2.1 The Robot

The ServoBot is a small (25x12cm; 25x24cm including legs), inexpensive hexapod robot with a BASIC Stamp II controller. The controller is capable of holding the program that can produce a cycle of activations required for 12 servo motors to move its legs in a normal gait. In addition it can hold a sequence of commands that result in a sequence of turns and straights performed by the robot.

2.2 Gait Cycles

A gait is produced by the controller sending pulses to the robot's actuators (servo motors). The control program includes a sequence of activations that the on-board controller will continually repeat. Each activation controls the instantaneous movement of the 12 servo actuators. 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. 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. For example: an activation of 001000000000 results in the lifting of the left front leg; 000001000000 results in the pulling back of the second right leg. 001001000000 would activate both at the same time. This set of input activations is held active by the controller for one servomotor pulse (approximately 25 msec).

A repeated sequence of these activations can be evolved by a cyclic genetic algorithm (Section 3.3) to produce an optimal gait for a specific SevroBot (Parker, Braun, and Cyliax, 1997). The gait generated for area coverage tests was a tripod gait (Figure 3). The tripod gait is where legs 0, 3, & 4 alternate with legs 1, 2, & 5 in providing the thrust for forward movement. While one set of legs is providing thrust, the other set is repositioning for its next thrust. In the case of the ServoBot used, the entire cycle lasted for 58 activations with each set providing 29 activations of thrust.

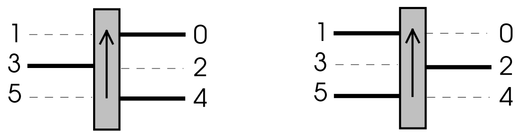

Figure 3: The two leg configurations for the robot during a tripod gait. The solid lines show legs that are on the ground and producing thrust. The dashed lines show legs that are repositioning for the next thrust by initially lifting and then lowering again while the leg is moving forward. Legs 0, 3, & 4 alternate with legs 1, 2, & 5 in providing thrust.

2.3 Production of Turns in Gait Cycles

Differing degrees of turn were provided in the gait cycles through the use of *affecters*. These affecters could interrupt activations to the thrust actuators for either the left or right side of the robot. Since the normal gait consisted of a sequence of 29 pulses of thrust to move the leg from the full front to full back position, anything less than 29 would result in some dragging of the legs on that side. For example: a right side affecter of 7 would allow only 14 (2 x 7) thrusts on the right side while keeping 29 on the left. The result would be that the left side would move further than the right resulting in a right turn. Affecters from 0 to 15 (4 bits) were possible. 0 meant that side would get no thrust producing a maximum turn. 15 will not affect the normal gait so the result should be a straight track. A one bit indicator specified if the affecter was for the right or left.

Each gait cycle, made up of 58 activations, was assigned an affecter, which resulted in a turn throughout that cycle. For consistency, each gait cycle started with legs 0, 3, & 4 full forward and legs 1, 2, & 5 full back; all the legs were on the ground. As the gait cycle started legs 0, 3, & 4 would provide the thrust as legs 1, 2, & 5 would start to lift and move forward to reposition for their thrust after 29 activations. A single gait cycle was defined as being complete when the legs returned to their starting positions (in this case, after 58 activations).

2.4 Cycles of Gait Cycles

The controller can be programmed to make the series of turns specified in an input sequence by application of the affecters to produce the corresponding gait cycle. The input sequence includes the turn direction, turn strength (affecter), and the number of times to repeat that gait cycle. Up to nine changes in gait cycles can be used with up to 63 repetitions of that gait cycle. The effective result was to produce cycles of gait cycles that could be used to define a desired path over the ground. A cycle of subcycles results in a single cyclic behavior.

3 STATIC SOLUTION USING A CYCLIC GENETIC ALGORITHM

Training was done to find the best search path for a specific robot. The robot's base gait cycle was learned using a cyclic genetic algorithm and was optimizing for speed; 15 left and 15 right gait cycles were programmed using the method described in Section 2.3. The effect of each of these gait cycles was tested on the actual robot and the results were used to build a list of robot capabilities. This list was used to simulate the robot for the CGA while it generated area coverage search paths.

3.1 Simulated Robot Performance

The robot was simulated by maintaining its current state, which was made up of its *xy* position in the area and its orientation (heading). Its capabilities were stored in a list of 32 gait cycles (Figure 4). Each element of the list could be identified by its gait cycle number (five bits). The high order bit describes whether the turn will be left (1) or right (0) and the remaining four bits indicate the level of turn. The list (F T ΔH) of three numbers after that indicates the result of applying that gait cycle for one cycle.

```
( 0 (3.7 4.0 24.3))  (16 (5.0 -3.7 -26.7))
( 1 (3.7 4.0 22.2))  (17 (5.7 -3.7 -24.7))
( 2 (3.8 4.3 20.2))  (18 (6.0 -5.0 -22.2))
( 3 (4.8 4.3 18.8))  (19 (6.3 -4.8 -20.3))
( 4 (5.3 4.0 16.7))  (20 (7.3 -4.3 -18.8))
( 5 (6.5 4.0 14.7))  (21 (8.4 -4.3 -15.8))
( 6 (7.3 3.8 13.2))  (22 (9.6 -3.8 -13.5))
( 7 (8.1 3.5 12.2))  (23 (10.4 -2.8 -10.3))
( 8 (8.4 3.5 11.0))  (24 (11.1 -2.3 -7.0))
( 9 (9.4 2.8 8.3))   (25 (11.9 -1.5 -5.0))
(10 (10.1 2.3 6.2))  (26 (12.1 -1.5 -3.7))
(11 (11.4 0.8 2.7))  (27 (12.1 -1.0 -2.7))
(12 (12.1 -0.1 0.0)) (28 (12.4 -1.0 -2.3))
(13 (12.1 -0.5 -1.3))(29 (12.1 -1.0 -2.7))
(14 (12.4 -0.5 -1.8))(30 (12.1 -1.3 -2.3))
(15 (11.6 -0.8 -1.3))(31 (11.6 -0.8 -1.3))
```

Figure 4: The robot's capabilities stored in 32 gait cycles. The first number in each element of this list is the gait cycle number. When looked at as a five bit number, the first bit designates whether the turn is left or right and the remaining four bits designate the strength of turn. A strength of 0 is a maximum turn, a strength of 15 is no turn. The left column shows the right turn gait cycles and the right column shows the left turn gait cycles. The three numbers listed after each gait cycle

number represent the robot's capabilities. They are the measured results of running that gait for one cycle.

Each gait cycle was tested for rate of turn by running the robot for four cycles while taking three measurements (Figure 5). *F* was the distance in centimeters that it moved forward. The *F* axis was defined as the heading of the robot before movement. *T* was the distance traveled left or right. The *T* axis was defined as a perpendicular to the *F* axis. Left movement resulted in a negative *T*, right in a positive *T*. ΔH was a measurement (in degrees) of the change in heading from the start heading *F* axis to the heading after execution of the gait cycles. Left was negative, right was positive. After making these measurements, each was divided by 4 to attain the average turn rates. The sharpest turns, effecters less than 3, resulted in turns of greater than 90° after four gait cycles, so three cycles were used in these cases. Turn rates, defined using *F*, *T*, and ΔH; were stored for each gait cycle.

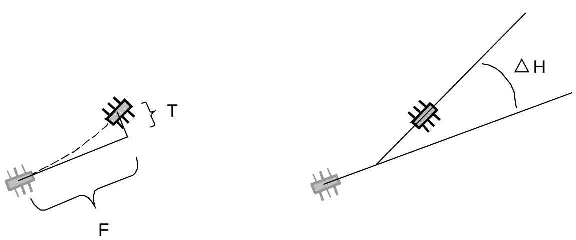

Figure 5: Gait cycle turn measurements. The left diagram shows *F* and *T*. *F* is the distance moved forward (relative to the start position heading). T is the distance moved in the turn direction (perpendicular to the start position heading). The right diagram shows ΔH, the change in heading from before to after turn execution.

3.2 Simulated Environment

The test area (Figure 2) was simulated by an *xy* grid where point (0,0) was the lower left corner. The lower right corner of the area was the point (180,0). The lower boundary was at $y = 0$, the left boundary was at $x = 0$, the right boundary was at $x = 180$, and there was no upper boundary. Mines were considered to be in 60×60 square blocks. The first row had centers at (30,30), (90,30), and (150,30). The second row started at (90,30), etc. The robot's start position was placed at (45,30) with an initial heading of 090 (Figure 2). This location assured acquisition of the first mine and put it in a good starting place to acquire the first row of mines. Motion was determined by applying each gait cycle from the chromosome one at a time. Using the current *xy* position and heading of the robot, a new position was calculated by applying the forward (*F*) and left/right (*T*) movements stored for that gait cycle as described in the previous section. The new heading was an addition of the current heading and the gait cycle heading change (ΔH). The path was not restricted from going outside of the area and the calculations remained the same if it did. This allowed, if appropriate, for the robot to do its turns out of the area so that it could attempt straight tracks within the area.

3.3 Cyclic Genetic Algorithms

Cyclic genetic algorithms (CGAs) were developed to allow for the representation of a cycle of actions in the chromosome (Parker and Rawlins, 1996). They differ from the standard GA in that the chromosome can be thought of as a circle with up to two tails (Figure 6) and the genes can represent subtasks that are to be completed in a predetermined segment of time. The tails of the CGA chromosome allow for transitional procedures before and/or after the cycle, if required. In our area coverage experiments, we used only the cyclic portion since the start position was known and the search tactic was to be applicable for any duration. The CGA genes can be one of several possibilities. They can be as simple as primitive subtasks (activations) or they can be as complicated as cyclic sub-chromosomes that can be trained separately by a CGA. For the area coverage problem the genes represented a set of gait cycles that were to be sustained for one cycle each. The trained chromosome contained the cycle of these gait cycles that was continually repeated by our robot's controller to produce a path that was to efficiently cover the designated area.

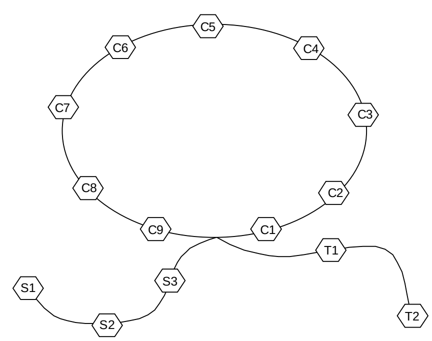

Figure 6: CGA chromosome with three genes in the start section (before cycle), nine genes in the cyclic section, and two genes in the tail section (after cycle).

3.3.1 Area Coverage Chromosome

The controller program has a provision for nine changing gait cycles in the search cycle. Each gait cycle (there are 32 possible) takes 5 bits to identify and the repetitions of each gait cycle can be from 0 to 63. The CGA chromosome used directly resembles the required input to the controller. Each chromosome is made up of 9 genes (9 genes fit within the storage capacity of the BASIC Stamp and were judged to be sufficient to perform the task) and each gene of the chromosome is made up of 2 parts (a 5 bit number and a 6 bit number). The scheme representation of the chromosome is shown in Figure 7. The first number in each pair represents the gait cycle while the second represents the number of times to repeat that gait cycle.

((GS_1 R_1) (GS_2 R_2) (GS_3 R_3) (GS_4 R_4) . . . (GS_9 R_9))

Figure 7: Area coverage chromosome.

3.3.2 Implementation of the CGA

An initial population of 64 individuals, made up of chromosomes described in the previous section, was randomly generated. Each individual, representing a cycle of gait cycles that would form a path, was tested to determine its fitness after 100 of these gait cycles were executed (this would be a total of 5800 activations); 100 gait cycles were enough to ensure that some turning was required to cover the optimal number of mines. It was not, however, enough to force the formation of a cycle that could provide continuing full coverage. Due to this limitation, the required number of gait cycles was randomly (with a 1 out of 2 probability) increased to 200. This allowed for faster fitness computations while using 100 gait cycles, yet put selection pressure on the population to evolve individuals capable of performing well at 200 gait cycles. The CGA was run for 5000 generations with the best solution (individual chromosome) saved whenever there was an increase in fitness (more mines covered in the allotted time).

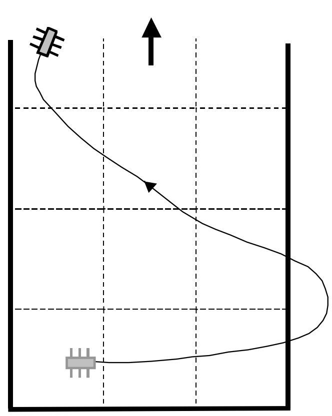

Figure 8: Fitness calculation. This track would result in a fitness of 4. The first row of mines, plus the second row's right mine are covered. The second row's middle mine is not considered covered because the robot's body at the midsection was not entirely within the block. The third row middle mine would be detected by the robot, but will not count since there were mines in the second row that were not covered. However, it would be counted if the robot circled back around and covered the two mines missed in the second row. Also note that the robot is free to depart the area with no direct fitness penalty.

3.3.3 Solution Fitness

Selection probability was determined by the individual's fitness. This fitness was calculated by counting the number of mines detected (mine blocks covered by robot's path) after it had completed a specified number of gait cycles (Figure 8). Counting, which was not done until the search path was completed, began by rows from the bottom of the area. As soon as a mine block was missed no more rows were counted, although the mines from the partial row were counted. The idea was for the robot to ensure that the area covered was completely free of mines. For the fitnesses calculated during training, mines visited more than once were not counted, although they did count for row completions. This was to discourage paths that were wasting time re-searching covered area. Once a fitness was calculated for each individual in the population, pairs were stochastically selected for reproduction. The genetic operators used were the same as described in previous work (Parker, 2001).

3.3.4 Tests

Five initially random populations were each trained using the CGA. Tests were done on the individuals saved during training to record the progress of the best individual in solving the area coverage problem. The average of the best fitnesses for each recorded generation from the five populations was then calculated. The initial growth was relatively fast. The rate of improvement slowed as each population gained a near optimal back-and-forth boustrophedic pattern and only make slight improvements after that. All the solutions had a fitness of at least 30 by generation 500. This means that 30 blocks were covered as the robot completed 200 gait cycles. The average fitness of the five final solutions was 34 and the best solution resulted in a path where the robot attained 37 blocks.

4. DYNAMIC SOLUTION USING THE CO-EVOLUTION OF MODEL PARAMETERS

The CGA worked well in learning a good solution for area coverage, but its answer was only as good as the model used for training. The model was produced through accurate measurements of the actual robot as it performed each gait cycle. Due to natural changes in the robot's capabilities, this model quickly becomes inaccurate. Only through continual adjustments to the model's parameters can the CGA continue to find an optimal solution.

4.1 The Co-Evolution of Model Parameters

Previously introduced and used as a learning system designed to generate gaits for hexapod robots (Parker, 2000), co-evolving model parameters dynamically links the model to the actual robot. It involves doing periodic tests of evolved solutions on the actual robot to co-evolve the accuracy of the robot's model with the CGA produced control solution. This extension of anytime learning allows for an adaptive real-time learning system that needs only global observation to make corrections in the robot model.

Figure 9 shows how it affects the learning. The model's parameters are constantly under review while anytime learning is in process. Training with a GA takes place off-line on a simple model. Periodic checks on the actual robot help to verify the model's accuracy.

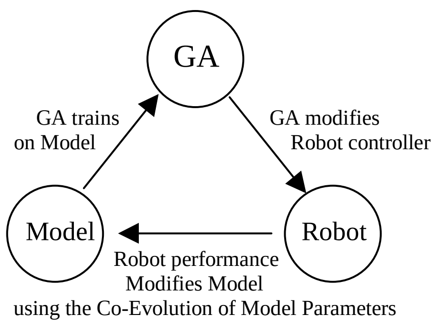

Figure 9: Co-evolving model parameters.

The form of evolutionary computation used to co-evolve the model parameters in past experimentation (Parker, 2000) has been a basic genetic algorithm. A population of individuals is randomly generated before training begins. This population can start out either as randomly generated individuals or as a combination of perturbations (to varying degrees) of the original model parameters. Each individual is made up of a set number of genes. Each gene represents a corresponding field in the robot's model. These genes evolve to produce models that correspond in performance to the actual robot. After each *n* generations the best and two other area coverage solutions are tested on the actual robot. These measurements are used to judge the accuracy of a population of model parameters by comparing the performance of the area coverage solutions on the actual robot with their performance on each model. The most accurate individual in the population of model parameters is used for continued controller evolution. Fitnesses for each individual in the population of model parameters are used as they co-evolve with the controller solutions. 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.

4.2 Partial Recombination Used in the Evolution of Model Parameters

The model parameters that are needed to be co-evolved for area coverage are the components of the gait cycle table shown in Figure 4. The *F*, *T*, ΔH for each affector needs to be learned. The chromosome for co-evolution is shown in Figure 10.

((0 (F T ΔH)) (1 (F T ΔH)) (2 (F T ΔH)) . . . (31 (F T ΔH)))

Figure 10: Model parameter chromosome.

Each gene was made up of affector number (including the one bit indicating the turn direction) and a set of three numbers representing the robot's distance moved and change in orientation.

Although the basic GA worked well in co-evolving the model parameters for hexapod gait generation (Parker, 2000), it was deemed inappropriate for evolving the model parameters for area coverage. Since a turn cycle used up to nine gait cycles, 23 of the 32 gait cycles would have no bearing on the fitness. They would be altered as a side effect while the nine or less being evolved tended toward an optimal. This could result in the loss of vital building blocks required to evolve the gait cycles destined for future use. In addition, since some means of keeping related building blocks close to each other should be advantageous (Holland, 1975), related gait cycles needed to be together during evolution. Although several gait cycles may be related by their probable use together in the execution sequence, predicting these relations in advance is difficult. Both of these problems are addressed by using partial recombination.

In a standard GA, the chromosome is fixed and related building blocks can be separated by significant distances. One means of lessening this gap is to group related building blocks together, but this requires that one knows which are related. Another solution, messy GA's (Goldberg, Deb, and Korb, 1991) can be used to learn this ordering. In the model parameter learning problem, the related building blocks cannot be predetermined, but they become better known during runtime. There is no need to learn the best building block positions; they can be assigned during learning. They do change, however, so reassignments must be possible. No previous research dealing with the problem of evolving only part of the parameters could be located. Partial recombination solves this problem by extracting the needed gait cycles for application of the genetic operators. These gait cycles (each of which could be considered a building block) were used to build a new chromosome by placing them in the order they were needed in the CGA generated turn cycle, which was probably the best guess for related order. The genetic operators were applied to this partial list of all the gait cycles for the designated number of generations. Upon completion of training, they were re-inserted into the main model parameter population. This allowed for training on the appropriate gait cycles (building blocks) arranged in the proper order without disturbing the rest of the building blocks.

Co-evolution starts when a best solution is sent to the Model parameter GA by the turn cycle generating CGA. Two more turn cycles are generated using this best solution. One is a perturbation of up to ± 1 on each of the non-zero repetitions in the turn cycle. The other uses a .25 probability starting from the first gene to find a gene that it sets the repetitions to 50. The partial recombination GA chromosome is built by extracting the needed gait cycles from each individual of the model parameter population. The three turn cycle solutions (the best found by the CGA, plus two perturbations) are each run on the actual robot and on the 64 partial robot models. The fitness of each is judged by comparing (finding absolute difference) its performance to the actual robot's performance. Two figures are compared for each gait cycle solution—the number of blocks covered and the number covered only once. This done on the three turn cycle solutions results in six total differences which are added together to get the fitness. This fitness is used to perform the standard GA operators of selection, crossover, and mutation. After 50 generations, the partial chromosomes are re-inserted into the main chromosome with the best designated as the current model for further CGA training.

4.3 Tests

To test anytime learning in simulation, a "correct" gait cycle list that differed from the one created through capability measurements was generated. It had each turn rate shifted to be off by one. Each turn strength was to be less than expected. Gait cycle n of the new list would be equivalent to gait cycle $n + 1$ of the old list. The max turns (strength 0 for both left and right) were thrown out and the turns with strength 14 were equal to strength 13 turns. This new gait cycle list was used to simulate the actual robot. This simulated actual robot did not turn as sharply as the training model indicated so all the path planning solutions would be slightly off. This could match an actual situation where the robot lost some of its turn capability in all turns.

To perform anytime learning, a population of 64 gait cycle lists was generated by perturbing the original gait cycle list used for CGA training. The final (5000 generation) population of path plans generated by the CGA was used for continued training. The anytime learning system tested all these solutions using the original gait cycle list (the best known model at the time). This test was done for 50 gait cycles as opposed to 100 or 200 since this is what would be done on the actual robot. The best solution plus two perturbations of it, as described in Section 4.2, were used to find a fitness for each gait cycle list (model) in the population of models. A genetic algorithm with partial recombination was run for 40 generations. At the completion of this training, the best model was used to replace the original model. The CGA was run again for 40 generations using this new model to evolve a new best path planning solution. This process was continually repeated for a total of 1000 CGA generations. The best path planning solution (judged by its performance on the best know model at each 40 generation mark) replaced the simulated robot's operational solution if its performance was better on the actual robot.

The results of applying anytime learning to these resultant populations are shown in Figure 11. The simulated robot, which was the "corrected" model, was used to test the best solution (plus two perturbations) for fitness calculations used to co-evolve the model parameters. Although the same simulated robot was used for all five tests, the starting populations differ since they came from the previous test. In addition, the model parameter populations differ since they were randomly generated.

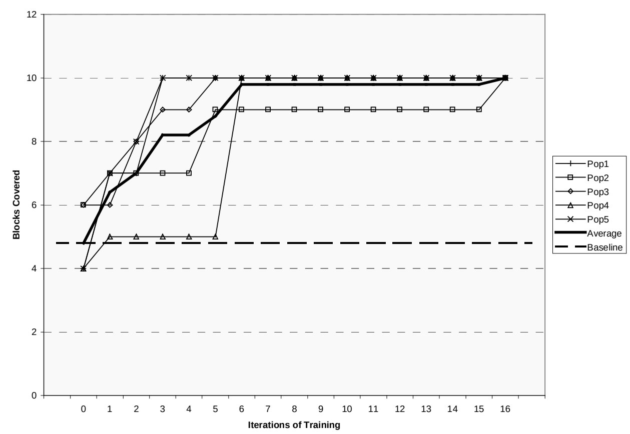

Figure 11: With the co-evolution of model parameters. The vertical axis shows the number of blocks covered after 50 gait cycles (this is also referred to as fitness). All lines represent the results of running the operational control path (the best generated up until that point). The baseline is the average fitness of the five without anytime learning. The horizontal axis shows the number of training iterations. Each iteration is equal to 3 trials on the robot, 40 generations of model training, and 40 generations of path planning training.

Each Pop$x$ listed represents what one would see as a robot was training using anytime learning with the CGA learning the turn cycle control sequence and the GA with partial recombination learning the model parameters. As the best model was updated it was stored in the operational side of the control system. The best model remained in effect until a new model evolved that was found to be superior. The graphs trace these best models as the robot's performance continually improved. As can be seen, four of the five populations reached a fitness of 10 by the sixth iteration of the learning system. Each iteration consisted of three tests on the robot, 40 generations of model evolution, and 40 generations of control evolution. The fifth population got stuck at a fitness (blocks covered) of 9, but finally reached 10 after 16 iterations.

The heavy dashed and solid lines show average performance of the best models. The dashed line indicates what the average fitness of the five populations would be if there was no anytime learning. There is no improvement over time since the system's best model never changes. The solid line shows the average when anytime learning is employed. Now the best model is continually improved, which results in a more accurate training environment for the CGA.

5 CONCLUSION

Area coverage path planning provided an interesting problem for the CGA to solve. The setup was such that only a cycle of actions (turns and straights) could provide a useable solution. The learned cycle of actions provided the basis for a robot controller to repeat its pattern of movement as required to cover an area of any specified length. Although testing was done using only dead reckoning to strive for the best solution, the assumption is that some rough navigational device that makes minor position adjustments after each single cycle would be required for actual implementation.

The addition of anytime learning improved the practical usefulness of the system and confirmed the ability of the co-evolution of model parameters to provide real-time corrections. Tests in simulation showed that this type of anytime learning only needs global observation to improve the outcome. The average increase in blocks covered after training with the co-evolution of model parameters was nearly two times its starting average. This system moves the work in learning away from the robot allowing it to go about its operations while the path planning takes place off line.

These tests demonstrate that an anytime learning system based on the co-evolution of a path plan using cyclic genetic algorithms and of a simulation model using a genetic algorithm with partial recombination is an effective means of learning adaptive control strategies for hexapod robots performing area coverage. The CGA was successful in generating strategies for five out of five random start populations within 5000 generations. The anytime learning based on co-evolution of model parameters was shown to successfully adapt all five populations to the specifics of the simulated robot.

Further research will be conducted to test the use of the co-evolution of model parameters in tests on the actual robot as it performs practical applications. Research will also be done to explore the usefulness of partial recombination in other domains of evolutionary computation.

References

Choset, H. and Pignon, P. (1997). Coverage Path Planning: The Boustrophedon Cellular Decomposition. *Proceedings of the International Conference on Field and Service Robotics.*

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

Goldberg, D., Deb, K., and Korb, B. (1991). Don't Worry, Be Messy. *Proceedings of the Fourth International Conference in Genetic Algorithms and Their Applications*, (pp. 24-30).

Hert, S., Tiwari, S., and Lumelsky, V. (1996). A Terrain-Covering Algorithm for an Autonomous Underwater Vehicle. *Journal of Autonomous Robots*, (Spec. Issue on Underwater Robotics), 3, (pp. 91-119).

Hofner C. and Schmidt, G. (1995). Path Planning and Guidance Techniques for Autonomous Mobile Cleaning Robot. *Robotics and Autonomous Systems*, 14, (pp. 91-119).

Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, Mi: The University of Michigan Press.

Ollis M. and Stentz, A. (1997). Vision-Based Perception for an Autonomous Harvester. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robotic Systems*.

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. (2000). Co-Evolving Model Parameters for Anytime Learning in Evolutionary Robotics. *Robotics and Autonomous Systems*, Volume 33, Issue 1, 31 October 2000 (pp. 13-30).

Parker, G. (2001). Evolving Cyclic Control for a Hexapod Robot Performing Area Coverage. *Proceedings of 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2001)*. (pp. 561-566).

Zelinsky, A., Jarvis, R., Byrne, J., and Yuta, S. (1993). Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot. *Proceedings of International conference on Advanced Robotics*. (pp. 533-538).