Punctuated Anytime Learning for Hexapod Gait Generation

Gary B. Parker

Connecticut College, New London, CT, USA, parker@conncoll.edu

Abstract

Punctuated anytime learning is presented as the solution for two problems: the use of anytime learning with an off-line learning module and the linking of the actual robot to its simulation during evolutionary robotics. Two methods of punctuated anytime learning, fitness biasing and the co-evolution of model parameters, are described and compared using the common task of gait generation for a hexapod robot with changing capabilities.

1. Introduction

In order to explore areas of high risk where multiple robots are more suitable than a few robots, it is required that the multiple robots be relatively inexpensive to maintain the same cost. Inexpensive robots with sufficient computational power to maintain autonomous control, yet insufficient to do on-line learning, are plausible; but would not be capable of adapting to changes in the environment or their own capabilities. A highly capable central computer that is safely out of the area of high risk can be used to carry the learning system as long as its learning can be linked to the actual robot. *Punctuated anytime learning* is a way to link the off-line learning system to the robot's on-line controller.

There are two main issues that need to be addressed for off-line learning: anytime learning without immediate feedback and dynamic simulator accuracy in evolutionary robotics. Anytime algorithms were defined by Dean and Bobby [1] to be algorithms that could be interrupted at any time and return a current best answer as they strive to find the optimal answer. Grefenstette and Ramsey [2] used the term *anytime learning* to refer to a method of learning that integrates the execution (control) module and the learning module. This method had the characteristics of anytime algorithms except that the learning model never stopped; it continually gave a meaningful answer any time the solution was improved. A simplified version of the anytime learning system is shown in Figure 1. There are two main modules. The Control Module contains the control program, which directs what actions to take in response to the input sensors. The Learning Module's function is to use an internal simulation of the robot and its environment to learn a better control program. Feedback from the Control Module can be used to update the internal representation in the Learning Module.

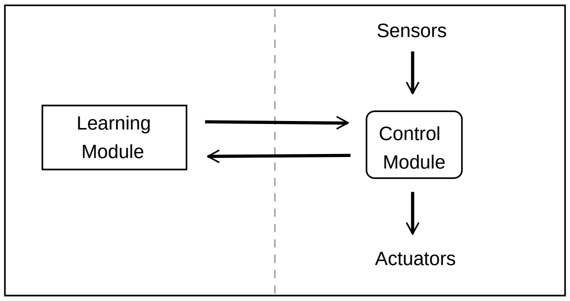

Figure 1: *Typical anytime learning system.*

The anytime learning system was used to solve a simulated problem of cat-and-mouse. The system was to be operating in a mobile agent (the tracker or cat) that was to track a second mobile agent (the target or mouse) that was pre-programmed for avoidance. The anytime learning systems, although they had temporary drops at each phase change, were generally superior to the systems without anytime learning. They allowed a learning component to continually compute a best control solution while the robot operated in the environment using the latest best solution. Grefenstette's system could adapt quickly to changes in the robot or the environment by having the robot's sensors continually update the status of the environment and the robot's capabilities in the learning component's simulation. This is where a difficulty develops when this methodology is applied to off-line learning. For off-line learning, the learning system can only depend on global observation, instead of precise sensors, to determine the robot's capabilities. A modification to anytime learning is required to compensate for the lack of quick feedback, precise sensors, and high computational power in robots.

Evolutionary robotics is a research area that makes use of the various forms of evolutionary computation (EC) to provide a means of learning control systems for robots [4, 9]. Most forms of evolutionary computation require that a population of possible solutions be tested over several iterations. A significant issue of discussion in Evolutionary Robotics is whether or not these tests should be done on simulations. There are three general approaches. One is to do all of the training off line and transfer the results to the actual robot when it is complete [5, 6, 20]. This technique requires that significant attention be paid to the simulation as its accuracy directly affects the results. A model is created to have the key characteristics of the robot. This model can be very complex to better simulate the robot, but this usually means that it will take longer to develop and have increased computation time during learning. A simple model will allow faster computation time, but will not accurately represent the robot. A second method is to do most of the training off line and transfer to the robot for some remaining generations [7, 9]. This requires a less accurate simulation (yet one is still required) and it can take significant time to do the on-line training. The third method does all of the training on line [3, 10, 12]. 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 simulation. The problem is that the training takes *num-generations* * *num-individuals* * *time-to-complete-task* time to do the training. An increase in any of these variables results in a multiplicative increase in training time. The problem with all of these techniques is that either time has to be to put into the simulation or into the training on the actual robot. In addition, none of them are designed to be able to refine 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].

This issue was addressed by Nordin et al. [13] as they used genetic programming to learn the best response to a set of stimuli for a Khepera robot. They stored memories of past tests on the robot to use in the evolution of a control function plus a more accurate model of the world. This solution worked well for the reactive control systems they addressed, but could not be applied to the learning of a cycle of activations to control the gait of a hexapod robot.

2. Punctuated Anytime Learning

A solution that provides this simulator/robot coupling and the needed modification to anytime learning is the use of a *punctuated anytime learning* (PAL) system. Training with evolutionary computation takes place offline on a simple model while periodic checks on the actual robot help to improve the learning systems output. This type of anytime learning is referred to as punctuated because the learning, although it is continuous, has break points where it is working on newly updated information about the actual robot. Figure 2 shows a punctuated anytime learning system. In this system, the Learning Module is not part of the robot. Being a separate unit, it no longer has direct input to the Control Module. Updated control programs are sent via some form of external communication. In addition, feedback is no longer sent via a direct link. It can be transmitted, such as the control program, if there is an evaluation unit on the robot. Alternatively, it may be provided to the Learning Module from an outside observer (pictured in Figure 2). The second option becomes more likely as the complexity of the robot goes down. There are currently two methods of punctuated anytime learning: *fitness biasing* and the *co-evolution of model parameters*.

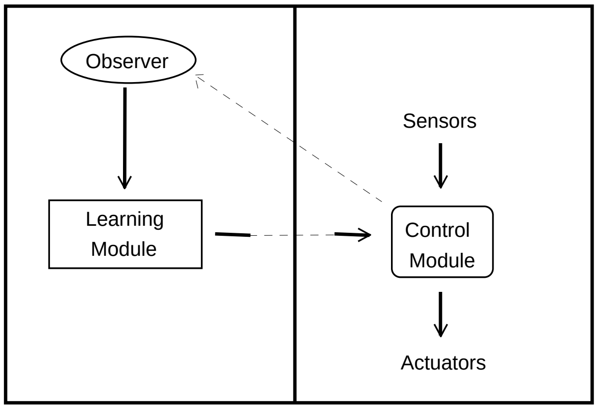

Figure 2: *Punctuated anytime learning system where the learning module cannot be on the robot.*

2.1 Fitness Biasing

Anytime learning with fitness biasing allows the system to modify the workings of the evolutionary computation based on the performance of the robot. The model is not affected; its parameters were set before anytime learning began. The periodic checks on the actual robot alter the processing within the evolutionary computation through the biasing of evolved solution fitnesses by comparing their performance on the model to their performance on the actual robot.

Probability for selection is determined by computing each individual's fitness on the robot model. This fitness is computed for each individual in the population. After each *n* generations all solutions of the EC population are tested on the actual robot. These measurements are used to bias the fitnesses found on the model to make them equal to the actual robot fitnesses. These biases are used as the EC continues training. In this way, solutions that are good on the actual robot but poor on the model have boosted fitnesses during training, which results in their production of more offspring.

This solution requires *population-size* actual tests every *n* generations.

```
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
```

Figure 3: *Algorithm to compute bias.*

A bias for each solution of the population is computed using the algorithm in Figure 3. This bias is stored with its corresponding solution. It is used in subsequent generations of the EC to alter the fitness of the solution computed on the model of the robot. . This is done by multiplying the fitness computed on the model by the bias:

$$\textit{Corrected-Fitness} = \textit{Model-Fitness} * \textit{Bias} \quad (1)$$

These *Corrected-Fitness*es are used for selection during the subsequent training being done by the EC. Pairs of individuals are stochastically selected for reproduction using the *Corrected-Fitness*es. The two individuals produce a single offspring for the next generation; combining their attributes by crossover with possible random variations caused by mutation. The new offspring's bias is computed by averaging the biases of its parents.

2.2 Co-Evolution Of Model Parameters

The co-evolution of model parameters dynamically links the model to the actual robot. Periodic tests of evolved solutions on the actual robot are used to co-evolve the accuracy of the robot's model with the EC produced solution. Every *n* generations of training on the model, three solutions (best, random, and worst (or another random)) are tested on the actual robot. Each of these three is run on the actual robot to gain a true fitness. This fitness should be general enough so that it can be acquired by external observation (such as an overhead camera). The three gait solutions are tested on each of the individuals in the model population. A comparison of each gait solution's fitness on the actual robot verses the model is used to determine the fitness of each individual of the model population (Figure 4).

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. A comparison of each gait solution's fitness on the actual robot verses the model is used to determine the fitness of each individual of the model population. A model that performs similarly to the robot on all three tests is assumed to have accurate parameters and is given a high fitness. The population of model parameters is started out through random generation. Alternatively, individuals can be generated by producing perturbations of the original measured parameters. After the model parameters are checked at the *n*th generation, the best is used to continue evolving gait solutions. The fitnesses applied to the individuals in the population of model parameters are used in the co-evolution of the model parameters, which will continue to evolve until interrupted after another *n* generations of gait learning.

```
R1 = Test best solution on the robot
R2 = Test random solution on the robot
R3 = Test worst solution on the robot
For each individual in the model population
    M1 = Get model fitness using best solution
    M2 = Get model fitness using random solution
    M3 = Get model fitness using worst solution
    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 either Order-Same or Sign-Same are True
          Return .001 x Fit_Comp
    If both are True
          Return Fit_Comp
```

Figure 4: *Algorithm to compute the fitness of each model's set of parameters in a population.*

For some problems, a genetic algorithm will not be the best optimization technique for finding the model parameters. It may be hill climbing or any other form of greedy algorithm. While the CGA is learning the cycle for the gaits, or whatever else is being learned, the model parameters can be learned at the same time, whether it be by some form of evolutionary computation or through some other means of learning. In either case, the concept of co-learning model parameters is the same.

3. Using PAL to Learn Gaits

Learning gaits that are adaptable in changing environments and with variable robot capabilities is of importance in the actual implementation of autonomous legged robots. A satisfactory gait requires that the proper sequence of commands are given to the various actuators of the robot in an order that results in the coordinated movement of each leg, plus the coordinated movement of the set of legs. Learning must be done continuously with the best possible gait to increase the effectiveness of the robot. Both forms of PAL have been used to adapt evolved gaits to changes in the robot's capabilities. The co-evolution of model parameters has been tested in simulation [17] and on the actual robot [19]. Fitness biasing, which requires a greater number of tests on the robot, has been tested in simulation only [16, 18]. Tests on the actual robot will be done following the completion of a training environment.

3.1 The Robot

The ServoBot was developed by David Braun to facilitate legged robot control experimentation. It is an inexpensive hexapod robot that has two degrees of freedom per leg. Its body is made of pressed wood fiber board that has been cut into the shapes of leg segments and structural support pieces. Twelve hobby servos, two per leg, provide thrust and vertical movement.

The controller, a BASIC Stamp II, can produce a pulse signal that drives the servos. An input to the robot controller is a twelve bit number (an *activation*) where each bit represents an actuator. A signal of 1 moves the leg back if it is a horizontal actuator and up if it is a vertical actuator. A signal of 0 moves it in the opposite direction. A control sequence is transmitted to a controller through lines that connect to a workstation. Once the control sequence is transmitted, the connection can be interrupted allowing autonomous operations. The controller can store and execute the sequence of primitive instructions downloaded. Some section of the sequence of instructions can be designated for repetition and the controller will continually loop through these instructions.

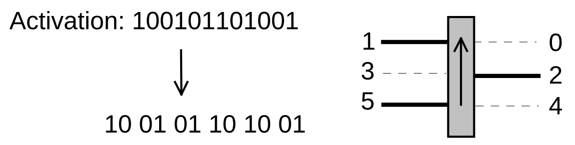

Figure 5: *Results of an activation on the robot.*

An activation (Figure 5) 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. The controller can store and execute (repeating the designated section) the sequence of primitive instructions downloaded. Some sequence of these activations will result in a pattern of leg movement that will produce a viable gait.

The model used for EC is a simple data structure that held each leg's capabilities and current state (Figure 6). The current up and down fields hold the leg's state. The combination of all the leg's states defines the robot's state. The remaining fields in the data structure define the leg's capabilities, which were measured before training. Input activations use the capabilities to determine how much to change the current state of the model during training. The parameters chosen for evolution were the max up and max back for each leg. The robot was adjusted so that max down was 0 for each leg. A random start population of models was produced and checked against the robot as described previously.

Fields specific for each leg:
- current up -- current vertical position of the leg.
- max up -- position off the ground when completely up.
- max down -- position off the ground, completely down.
- 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.
- rate back/forward -- rate of horizontal movement.

Figure 6: *Simulation model data structure.*

3.2 Cyclic Genetic Algorithms

The cyclic genetic algorithm (CGA) is a variation of GA that is designed for learning tasks that are to be continually repeated in a cycle. It has been successfully used to learn the sequence of servo activations required to produce a gait in hexapod robots. In these tests, the CGA learned control programs while training on a model of the robot. The generated programs were then downloaded into the robot's controller. Tests showed that the CGA could effectively learn gaits in a static environment [14, 15], but there was no provision for the system to dynamically correct for changes in the robot's capabilities or the environment.

3.3 Using Anytime Learning with Fitness Biasing

Tests in simulation were used to verify that fitness biasing was an effective means of improving robot gait performance during training [16, 18]. Each test involved training on the model, with tests on a simulation (more accurate model) of the robot. The calculated distance (in centimeters) traveled by the simulated robot after ten seconds of activation of the current gait was recorded. Each test was done using five random start populations. The results reported show the most fit individual in each population at different generations during evolution.

The decreasing capabilities test (Figure 7) was a series of changes where the simulated robot's capabilities continually decreased. The robot's leg 5 would lose 0.125 of its total horizontal movement capability every 500 generations; going from fully functional to no horizontal movement in 4000 generations. This could be a natural occurrence in actual robots operating in the field as one leg's servo gradually broke down or mud packed into the leg's joints.

The increasing capabilities test (Figure 8) simulated the gradual return of horizontal movement capabilities of leg 5. This is also a plausible natural occurrence, although less likely than the decrease, since packed mud could eventually dry and start to chip away.

The baseline shows the results of using the best gait generated in these 500 pre-graph generations (the optimal for the model) for each of the capability changes. This would be the resulting fitness without anytime learning since the learning system would assume the model is current. Also shown is a maximum fitness. This maximum was attained by running the CGA for 2000 generations using an entirely accurate model. The solid lines show the resultant fitnesses for the 5 test populations throughout the capability changes if anytime learning with fitness biasing is being used. This graph shows that the fitness biasing consistently improves the solution and in some cases can attain the maximum fitness when the maximum is not too far from the baseline.

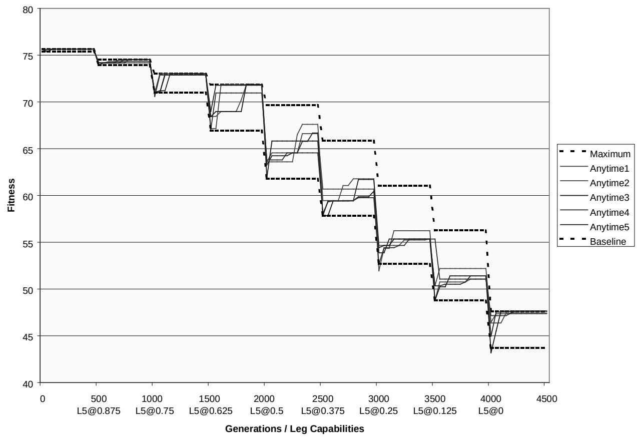

Figure 7: *Decreasing capabilities tests for fitness biasing. All five tests (solid lines), along with the baseline and maximum parameters (dashed lines), are shown.*

Anytime learning with fitness biasing results in improved performance. The amount of improvement is primarily affected by two factors—the amount of improvement possible and the difference between the optimal gait for the actual robot and optimal gait for the model. Anytime learning using fitness biasing is most effective when the optimal gaits for the robot and model are similar. Since this form of anytime learning only affects the genetic algorithm's fitnesses, learning is still done on the original model. The closer this model is to reality, the easier it is to correct the inaccuracy using fitness biasing. Minimal differences between the optimal gaits for the model and robot are also the times when there is less improvement to be made. The gait produced for the original model is close to the optimal gait for the robot so the baseline is closer to the optimal fitness.

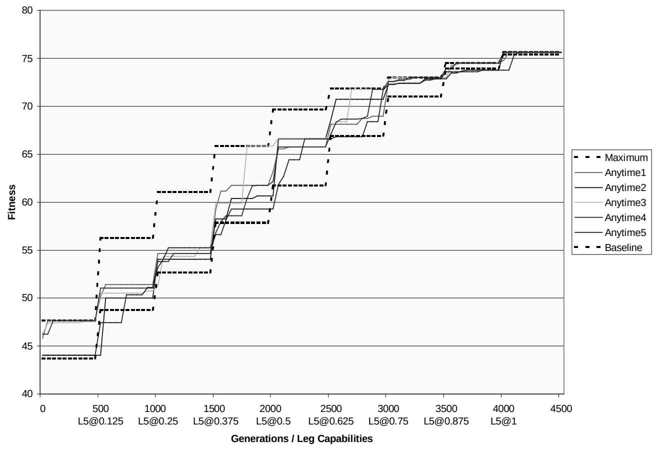

Figure 8: *Increasing capabilities tests for fitness biasing. All five tests (solid lines), with the baseline and maximum parameters (dashed lines), are shown.*

These times of minimal difference tend to be during very high and low capabilities. The general solution to correcting horizontal movement limitations is for the learning system to decrease the length of the strides to avoid leg dragging. This results in somewhat less fitness since the initiation of the stride produces less thrust than when the stride is already in motion. As capabilities continue to decrease, stride length reduction starts to produce diminishing returns and it becomes advantageous to simply drag a leg for a few activations. At this point, the CGA typically increases the stride again. Although the robot loses total thrust by dragging a leg, it gains by decreasing the initiation time during the stride. Therefore, gaits produced with very low leg capabilities are more similar than mid leg capability gaits to the fully capable gaits. This means there is less possible deviation from the baseline when leg capabilities are very high or low. On the other hand, these are also the situations when the anytime learning with fitness biasing is most effective. The end result is that the greatest deviation from the baseline will happen in the mid ranges (around 0.5 capability), but the learning system is more likely to achieve optimal gaits in the high and low capability ranges.

3.4 Using The Co-Evolution Of Model Parameters

To test the co-evolution of model parameters on gait generation [17, 19] a population of gaits was produced using the CGA applied to the initial model (a rough estimation of the robot). After 500 generations of training using only the CGA, co-evolution of model parameters began. The parameters chosen for evolution were the max up and max back (Figure 6) for each leg. The robot was adjusted so that max down was 0 for each leg. 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 became the new current model. The CGA was then allowed to run for ten generations using the current model to produce a new population of gaits. This process continued until told to stop. The model and gait were constantly changing to keep up with dynamic changes in the robot's capabilities.

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 improved or stayed the same, unless there was a degradation of capabilities. This could, in the worst case, add 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 would be required was when there was a reduction in the robot's capabilities.

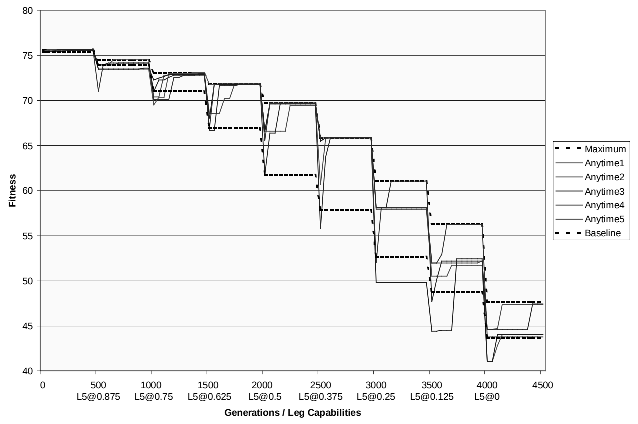

Figure 9: *Decreasing capabilities tests for the co-evolution of model parameters. All five tests (solid lines), with the baseline and maximum parameters (dashed lines), are shown.*

The decreasing and increasing capability tests were done to check the significance of using the co-evolution of model parameters. Figure 9 shows the decreasing capabilities test. This graph shows that the co-evolution of model parameters almost always improves the solution and in most cases attains the maximum fitness. One notable exception was *Anytime2* where the anytime solution was below the baseline for the entire time that the leg was at 0.25 capability. The model parameters must have become inaccurate enough that they could not be corrected during this phase. Its fitness was low again after the leg capability went to 0.125, but this time changes were made after 200 generations that corrected the situation. The system (on an average) produces a fitter gait with anytime learning. The exceptions are when capability changes initially occur. Although the fitness drops are greater at the changes with anytime learning, the times of decreased fitness only last for 50 to 100 generations while the system quickly compensates for the change. These compensations tend to improve the robot's fitness at each stage until a plateau (which could possibly be the optimum) is reached. The final observation is that anytime learning yields the largest improvement over the baseline in the mid capability ranges (around .5). In the upper ranges the static model is still close and in the lower ranges there simply is not a lot that can be done to improve performance.

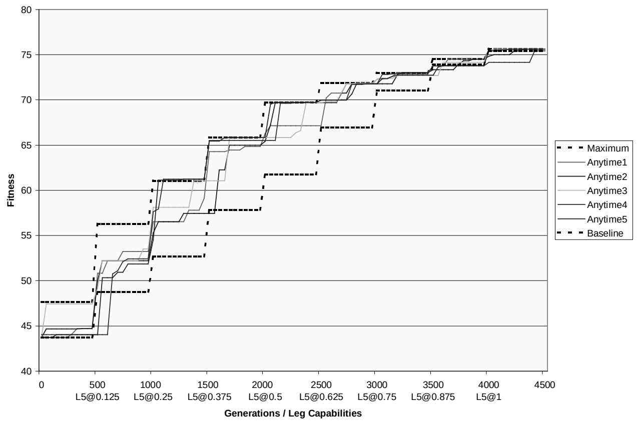

Figure 10: *Increasing capabilities tests for the co-evolution of model parameters. All five tests (solid lines), with the baseline and maximum parameters (dashed lines), are shown.*

The results of the increasing capabilities test can be seen in Figure 10. Here again the effectiveness of anytime learning can be observed. The biggest differences in the anytime learning line during increasing compared to decreasing capabilities are that changes never result in fitness drops and that the plateaus are not as well defined. Drops do not occur because gaits for lower capability robots will work on higher capability robots. The inverse is not always true. The formation of plateaus is also affected by this fact. Small changes in a decreasing capabilities situation can make significant improvements as the robot goes from leg dragging to not. On the other hand, small changes as the capabilities increase would result in small fitness increases because the leg produces only slightly more thrust by moving full throw before returning for another thrust. The result is slow, steady learning during increasing capabilities.

4. Comparison of Fitness Biasing and the Co-Evolution of Model Parameters

Fitness biasing takes minimal effort by the programmer to set up and can be applied to any evolutionary computation (EC) problem that is running on a model of the actual robot. The only provisions the programmer has to make are to provide a means of making the actual tests, performing the biasing, and determining how to pass an individual's bias on to its offspring. Any time these three steps can be taken, anytime learning with fitness biasing is an option. The difficulty with anytime learning with fitness biasing is that, since it does not change the model and training continues on an inaccurate model, it can only go so far. It cannot compensate for very inaccurate models. As observed in Figure 11, it is only likely to find the optimal solution when the optimal solution is not far from the baseline.

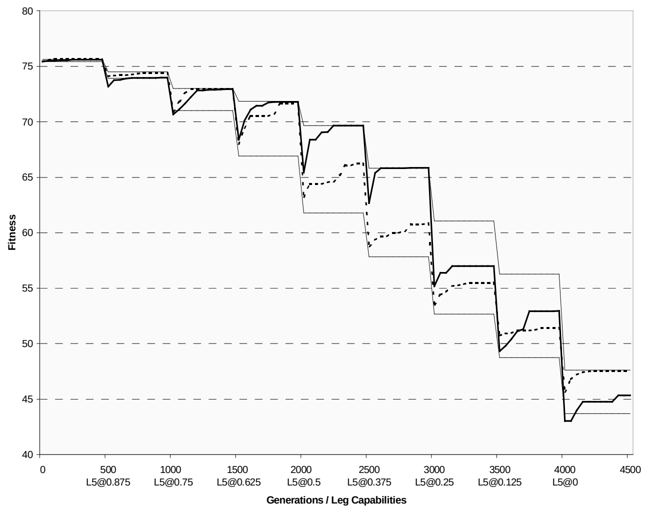

Figure 11: *Comparison of fitness biasing (heavy dashed line) and the co-evolution of model parameters (heavy solid line) during the gait generation decreasing capabilities tests. Results are the average of the 5 test populations. The baseline and maximum possible fitness are shown as light solid lines.*

Another problem with fitness biasing is that it requires that all individuals in the population be tested. This can be very time consuming and disruptive to the actions of the robot if the control sequence being optimized takes long to execute. Nevertheless, anytime learning with fitness biasing is an improvement over doing all testing on the robot because less online testing is required and it is an improvement over doing all testing on the model since the model accuracy does not have to be painfully precise. It is even an improvement over doing the initial training on the model and then transferring to the actual robot because it allows the system to be continually adaptive throughout training.

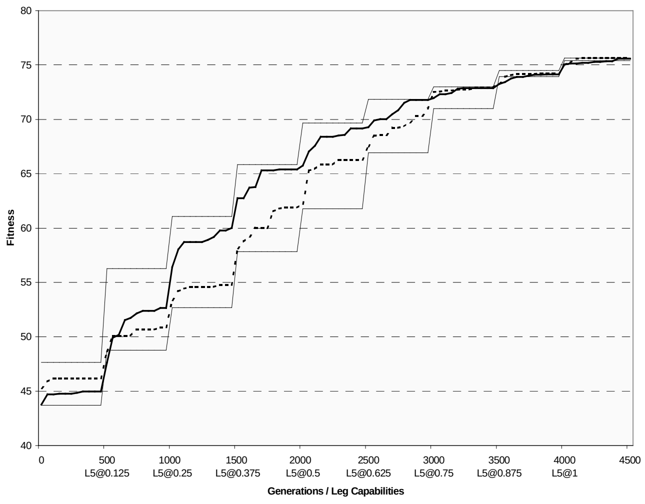

Figure 12: *Comparison of fitness biasing (dashed) and the co-evolution of model parameters (solid) during the gait generation increasing capabilities tests. Results are the average of the 5 test populations.*

Anytime learning using the co-evolution of model parameters addresses the two major issues that are shortcomings of anytime learning with fitness biasing. A look at Figures 11 and 12 shows that fitness biasing is capable of attaining near optimal performance in the very high and very low ranges where the baseline is closer to the optimal. As can be observed in the areas in-between, where the optimal is comparatively higher than the baseline, only the anytime learning that uses the co-evolution of model parameters can attain the higher fitnesses. This is because fitness biasing does change the model. It can compensate for the inaccuracies in the model by adjusting the model to be more like the actual robot, where as the fitness biasing does not change the model in any way.

There are, however, some disadvantages to this form of anytime learning. One of the major disadvantages is that it is not applicable for all problems. Some problems are such that it is very difficult to set up the model parameter co-evolution. There are those situations where we cannot specifically define the model parameters that can be co-evolved and there are those situations where the model parameters are too numerous to be co-evolved. Another difficulty with the co-evolution of model parameters is that it is possible to get a solution that is not as good as the one produced without anytime learning. See Figure 11 where the average solution temporarily falls below the baseline. This does not appear to be a problem with fitness biasing.

The other benefit of anytime learning using the co-evolution of model parameters is that it does not require that a test be done on all individuals of the population. It only requires three tests: the best individual, the worst individual, and a random individual. Using the information from these three tests alone, it co-evolves the model parameters and in this way changes the model so it is more accurate. In addition, it shares with fitness biasing the benefit that only a simple model is required. The model does not have to be extremely accurate in the initial phase so the programmer does not need to put in as much time as he would if all the training took place on the model. The time costs of testing on the robot have also been saved since only three tests are needed for each test generation.

5. Conclusion

Tests done in simulation showed that both anytime learning with fitness biasing and the co-evolution of model parameters 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. It is best to use the co-evolution of model parameters when one can easily define the model parameters in such a way that they can be co-learned along with the evolution of the solution that we are attempting to attain. In addition, it is good to use co-evolution of model parameters when the problem that we are trying to learn takes a long time to test each of the possible solutions. Fitness biasing is better in situations where one cannot define the parameters in such a way that they can be co-learned along with the solution and when the solution being tested is not so time consuming on the actual robot that it significantly disrupts the mission of the robot.

References

1. Dean, T. and Boddy, M. (1988). An Analysis of Time-Dependent Planning. *Proceedings of the Seventh National Conference on AI* (AAAI-88).
2. Grefenstette, J. and Ramsey, C. (1992). An Approach to Anytime Learning. *Proceedings of the Ninth International Conference on Machine Learning*, (pp. 189-195).
3. Harvey, I. (1996). Artificial Evolution and Real Robots. *Proceedings of International Symposium on Artificial Life and Robotics (AROB)*. (pp. 138-141).
4. Husbands, P., Harvey, I., Cliff, D., and Miller G. (1997). Artificial Evolution: A New Path for Artificial Intelligence? *Brain and Cognition*, Vol. 34, (pp. 130-159).
5. Jakobi, N. (1997). Half-baked, Ad-hoc and Noisy: Minimal Simulations for Evolutionary Robotics. *Advances in Artificial Life: Proceedings of the 4th European Conference on Artificial Life*.
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, (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. Nordin, P. and Banzhaf, W. (1995). Genetic Programming Controlling a Miniature Robot. AAAI Fall Symposium Series 1995, Symposium on Genetic Programming.
13. Nordin, P., Banzhaf, W., and Brameier, M. (1998). Evolution of a World Model for a Miniature Robot using Genetic Programming. *Robotics and Autonomous Systems*, 25, (pp. 105-116).
14. 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).
15. 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).
16. Parker, G. and Mills, J. (1999). Adaptive Hexapod Gait Control Using Anytime Learning with Fitness Biasing. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'99)*. (pp. 519-524).
17. Parker, G. (1999). The Co-Evolution of Model Parameters and Control Programs in Evolutionary Robotics. *Proceedings of the 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA'99).* (pp. 162-167).
18. Parker, G. and Larochelle, K. (2000). Punctuated Anytime Learning for Evolutionary Robotics. *Proceedings of the World Automation Congress (WAC2000), Volume 10, Robotic and Manufacturing Systems*. (pp. 268-273).
19. 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).
20. Schultz, A., Grefenstette, J., and Adams W. (1996). RoboShepherd: Learning a Complex Behavior. *Proceedings of the Robots and Learning Workshop (RoboLearn '96)*. (pp. 105-113).