# Fitness Biasing for the Box Pushing Task

Gary Parker and Jim O'Connor
Computer Science
Connecticut College
New London, CT, USA
parker@conncoll.edu and joconno2@conncoll.edu

***Abstract*—Anytime Learning with Fitness Biasing has been shown in previous works to be an effective tool for evolving hexapod gaits. In this paper, we present the use of Anytime Learning with Fitness Biasing to evolve the controller for a robot learning the box pushing task. The robot that was built for this task, was measured to create an accurate model. The model was used in simulation to test the effectiveness of Anytime Learning with Fitness Biasing for the box pushing task. This work is the first step in new research where an automated system to test the viability of Fitness Biasing will be created, as well as the first application of Fitness Biasing to a high level task such as box pushing.**

***Keywords-anytime learning, evolutionary robotics, learning control, genetic algorithm***

## I. INTRODUCTION

In evolutionary robotics, small or inexpensive robots tend to have issues learning due to relatively small onboard controllers. Pre-programming these controllers may ensure the desired results, but is 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, as well as provide the adaptability required for autonomy. Unfortunately this method can be too computationally intense to be carried out on board the robot. A system of learning that can be carried out offline and then downloaded to the onboard controller will allow the robot to adapt to changes in real time.

The task that is explored in this work is box-pushing, which is popular in the domain of evolutionary computation. It was examined by Lee [1], who used genetic programming to evolve behavior primitives for the Khepera robot. Up until this point, Fitness Biasing had not yet been examined as a learning method for a box pushing task. It adds the flexibility for the anytime learning to take place offline, while the robot operates in real time.

Anytime learning makes use of evolutionary computation in a learning module to periodically update the robot's control module. Grefenstette [2] used evolutionary computation to continually adjust the controller while the robot operated in an environment where the target agent had its capabilities periodically change. Grefenstette's system adapted to changes by modifying the model used to evolve the robot's controller based on changes in the environment, which are interpreted by an observation module on board the robot. Fitness Biasing, the learning system employed in the research reported in this paper, can adapt to changes in the robot's capabilities without the use of an onboard observation module to interpret changes in the environment. Depending only on global observation, this learning system uses evolutionary computation to learn a controller for the robot. This makes it more applicable for small autonomous robots because the learning component can be offline while the operations component, which is on the robot, receives periodic downloads of the best solution.

Most forms of evolutionary computation require that a population of possible solutions be tested over several iterations. In evolutionary robotics 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 offline and the results transferred to the actual robot when it is complete [3], then significant attention must be paid to the model as its accuracy directly affects the results. The time and effort can sometimes exceed the work required to program by hand. If the inaccuracies of the model are not dealt with, the control programs will not transfer to the robot. If most of the training is done offline and then transferred to the actual robot for some remaining generations [5], then a less accurate model is required, but it can take significant time to do the online 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 online [6]. This method precludes the need for any model of the robot, but the training takes *num-generations* * *num-individuals* * *time-to-complete-task* time to complete. An increase in any of these parameters 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. Fitness Biasing is a learning method that couples the simulation to the actual robot while learning is in progress, thereby allowing for the system to adapt to the changes in the robots capabilities.

Fitness Biasing was introduced by Parker [7] to address the problems outlined above. Fitness Biasing was used to evolve gaits for a hexapod robot with a single controller, and in subsequent works the same method was used for a hexapod robot with microcontrollers for each leg [8]. Although

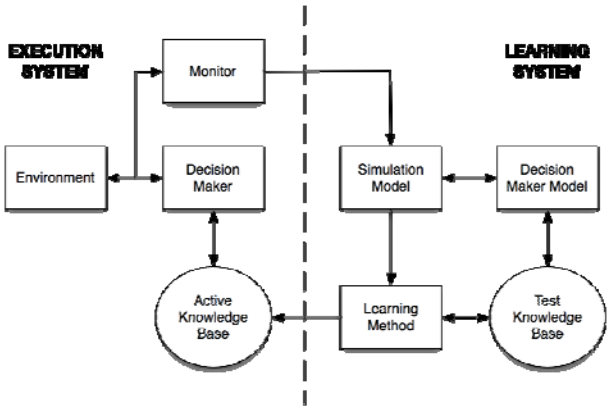


Figure 1. Diagram showing the Anytime Learning as described by Grefenstette [1]

successful in simulation and in preliminary tests on the actual robot, tests were never performed to apply Fitness Biasing to an automated system where learning took place without human interaction. In this paper, we employ Fitness Biasing for the higher level task of box pushing in addition to expanding the realm of areas where Fitness Biasing is applied. This is the first step in ongoing work to create a system where there is an automated link from the model to the actual robot in evolutionary robotics and to provide real-time learning for robots in operation. In this step, Fitness Biasing is employed in simulation to confirm that it can solve the higher level box pushing task. An accurate model of the actual robot is produced through tests on the robot. A genetic algorithm is employed to learn the task using an inaccurate model, with tests on an accurate model, which is to simulate the robot. In parallel to this work, the robot and colony space are being configured for the tests of the Fitness Biasing learning system on the actual robot.

## II. Fitness Biasing

Anytime learning is an approach to evolutionary robotics introduced by Grefenstette [2] that allows for learning in a dynamic environment. The anytime learning system that Grefenstette describes is broken into two systems, as shown in Figure 1: the execution system and the learning system. The execution system is comprised of the environment, the decision making system of the robot, and a monitor. The monitor is on board the robot and observes the environment for changes, in order to update the simulation.

The simulation is contained in the learning system, which is comprised of the simulation, a learning method, a test knowledge base, and a decision maker model. The learning system works to try many possible strategies in the simulation, and provide the execution system with the best current strategy. The learning system adjusts its strategies in response to feedback from the monitor based on the robots environment.

Punctuated Anytime Learning (PAL) [9] is an extension of anytime learning that was developed for robots that lack an observation module on board and allows for interactively connecting the simulation and reality to provide effective evolutionary computation. PAL compensates for the lack of an onboard observation module through global observation. Additionally, the system trains the controller with a genetic algorithm in a simple offline model, while periodic checks on the actual robot find and help to correct any disparities between the simulation and reality.

Fitness Biasing [7] is one of two types of PAL. When using Fitness Biasing, the model of the robot is not changed; instead the genetic algorithm is improved by biasing the fitness of each chromosome according to periodic checks on the actual robot. These periodic checks allow the chromosomes that are more effective in the real world to have a higher probability of selection.

Probability for selection is determined by computing each individual's fitness on the robot model. After each n generations all solutions of the genetic algorithm 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 genetic algorithm 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. To select parents for crossover, the corrected fitness of each individual is determined as *fitness-on-model* * *bias*. When crossover is performed, the biases of the two parents are averaged to determine the bias of the child. This solution requires *population-size* actual tests every n generations.

## III. Colony Space and Simulation

The task being explored in this work is a box-pushing scenario. Box pushing is an important behavior for robotics, because of the many applications that are built around pushing objects.

### A. Colony

The colony space is an eight-foot by eight-foot section in the Connecticut College Autonomous Agent Learning Lab used for robotics research. The colony is surrounded by a one-foot high wall, and is equipped with an overhead camera. The camera is connected to a nearby workstation, and can view the entire colony space at once. This camera is used in our work to simulate a robot's GPS, by providing the coordinates of the robot as they would appear if the colony was a cartesian plane. The colony space is covered with a low pile carpet to retain traction for wheeled robots. The box used in this research is also in the colony space. The box is a standard two-foot by two-foot by four-inch cardboard box, and is easily moved by the robot. A simulation of the colony space was built for this work, and the space was divided into nine equal sections. The


Figure 2. The box pushing robot developed for this work.

simulation incorporated the colony space, the box, and the robot. The parameters of the colony and box were modeled after and in proportion to, the actual colony and cardboard box used for box-pushing research.

### *B. Robot*

The robot used in this work is a small, inexpensive wheeled robot, as shown in Figure 2. The body of this robot is a six-inch wide, six-inch long, five-inch tall acrylic box. The robot has two wheels powered by servo motors modified for continuous rotation, and one omni-wheel. The robot has an array of ultrasonic sensors mounted in its front panel, which allow the robot to distinguish between the box and a wall, and to receive a measurement for the distance from each. The servo motors and the ultrasonic sensors are connected to an Arduino Duemillanove board, which contains an ATMega microcontroller. The robot was designed and assembled in the Connecticut College Autonomous Agent Learning Lab.

To create a simulation of the robot measurements of its capabilities were taken. The robot was placed in the colony space and ran twenty times down a measured track. The highest and lowest values for the speed of the robot were discarded before the data set was averaged to find the speed of the robot when moving forward. Similar experiments were repeated to find the turning rate for the robot, and the model parameters in the simulation were created accordingly. The parameters that are used in the simulation are the robots speed, turn rate, and the distance that it can see with its wall sensor, and with its box sensor.

### *C. Controller*

The controller for the robot involves using the information from the overheard camera to divide the colony space up into nine equal sections. When the robot makes a decision, it will turn left, turn right, or move forward based on which of these nine sections it is in. The controller has a separate set of rules (Figure 3) for each of these nine sections. The rules are placed in priority ordering, but the control bits (bit one, two, etc.) control whether or not the rule will ever fire. It is the job of the genetic algorithm to determine the values of these control bits.

```
IF in Section One
        IF box is seen AND bit one
                THEN move forward
        ELSE IF box is seen AND bit two
                THEN turn left
        ELSE IF box is seen AND bit three
                THEN turn right
        ELSE IF wall is seen AND bit four
                THEN move forward
        ELSE IF wall is seen AND bit five
                THEN turn left
        ELSE IF wall is seen AND bit six
                THEN turn right
        ELSE IF first bit of final section
                THEN move forward
        ELSE
                turn left
```

Figure 3. An example of how the first section of the chromosome was read by the controller. This refers to the six bits that make up the first section as well as the first bit of the final section.

## IV. Fitness Biasing Applied to Box Pushing

### *A. Genetic Algorithm*

The learning was done using a genetic algorithm, with a population of one hundred chromosomes, each composed of sixty-four bits. Each chromosome was split into nine sets of six bits, with ten additional bits (Figure 4). The first nine sets of six bits determined the robot's actions in each of the nine sections of the colony space. As shown in Figure 3, each of the six bits in each section of the chromosome controlled a different behavior. If the robot could see the box, the first three bits caused it to move forward, turn left, or turn right when activated, respectively. If the robot could see a wall, the same was true for the next three bits. In this way, the system allowed the robot to acquire a different behavior in each of the nine sections of the colony space based upon which of those

[**110101**011101**…**

Rule parameters for section one.

…**1**001011100**]**

The last ten bits. The first, bolded bit is used to determine whether or not the robot should move forward in section one.

Figure 4. Break down of the chromosome. The top six bolded bits correspond to bits one, two, …, six in Figure 3. The bolded bit in the lower group is the first bit of the final section in Figure 3.

six bits were activated in each of the nine sections of its chromosome. Nine of the last ten bits that make up the final section of the chromosome are used to determine what to do if nothing is seen while in one of the nine sections; if activated, move forward, otherwise turn left. The sixty-fourth bit is not used.

The fitness of each chromosome is measured by the number of times the robot completes the task in six minutes, with the location of the box and the robot being reset to random locations in the colony space every minute. A fitness proportionate (roulette wheel) method is used for selection of chromosomes, and chromosomes are combined with one hundred percent crossover and 0.05% chance of mutation.

### B. *Fitness Biasing*

To test the effectiveness of Fitness Biasing to solve the box pushing task, the first step was to test in simulation. This is the work reported in this paper. The simulation was to use a model of the actual robot, with its parameters carefully measured, to represent the actual robot. The measured parameters of this model were slightly skewed to create a model that represented the training model, which was to be used for evolution. To simulate a Fitness Biasing system, the genetic algorithm used the training model (inaccurate) to learn a controller with the periodic tests during the punctuated generations done on the robot (accurate model). These tests determined the biases, which were returned to the genetic algorithm to bias the chromosome fitnesses and continue evolution on the training model. The punctuated generations were every ten generations of the genetic algorithm learning system.

## V. Results

Three separate methods of learning were tested with five runs of each, to show the effectiveness of Fitness Biasing for evolving a box pushing controller. First, a standard genetic algorithm was run for two-hundred generations in a simulation of the task using an accurate model of the robot. This test was representative of running the genetic algorithm on the actual robot for a full two-hundred generations. This was the method of learning that produced the best controller after two-hundred generations, but it would be very taxing on the robot. Using a genetic algorithm in this way to evolve the control system for box-pushing entirely on the robot would be a viable method if not for the extreme amount of time that is required, as well as the wear on the actual robot. Second, a genetic algorithm was run in the simulation using the model that was slightly skewed; the parameters were each randomly off by a small amount, which is representative of running the genetic algorithm purely in a simulated model of the world. This method of learning produced the worst controller, due to the inaccuracies in the model. This problem of accuracy is the primary flaw with learning a control program entirely in

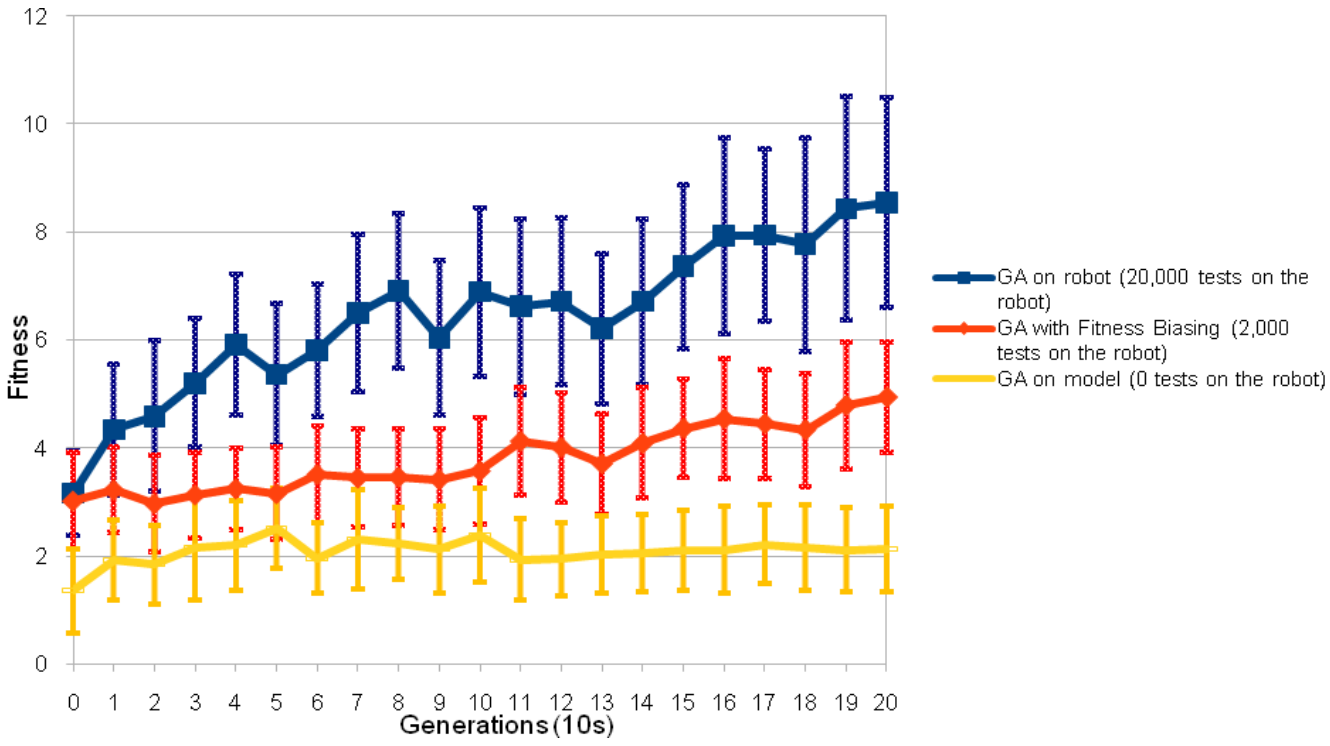


Figure 5. The best chromosome for every tenth generation for a genetic algorithm on the robot, on the model, and with Fitness Biasing. Error bars represent standard deviation.

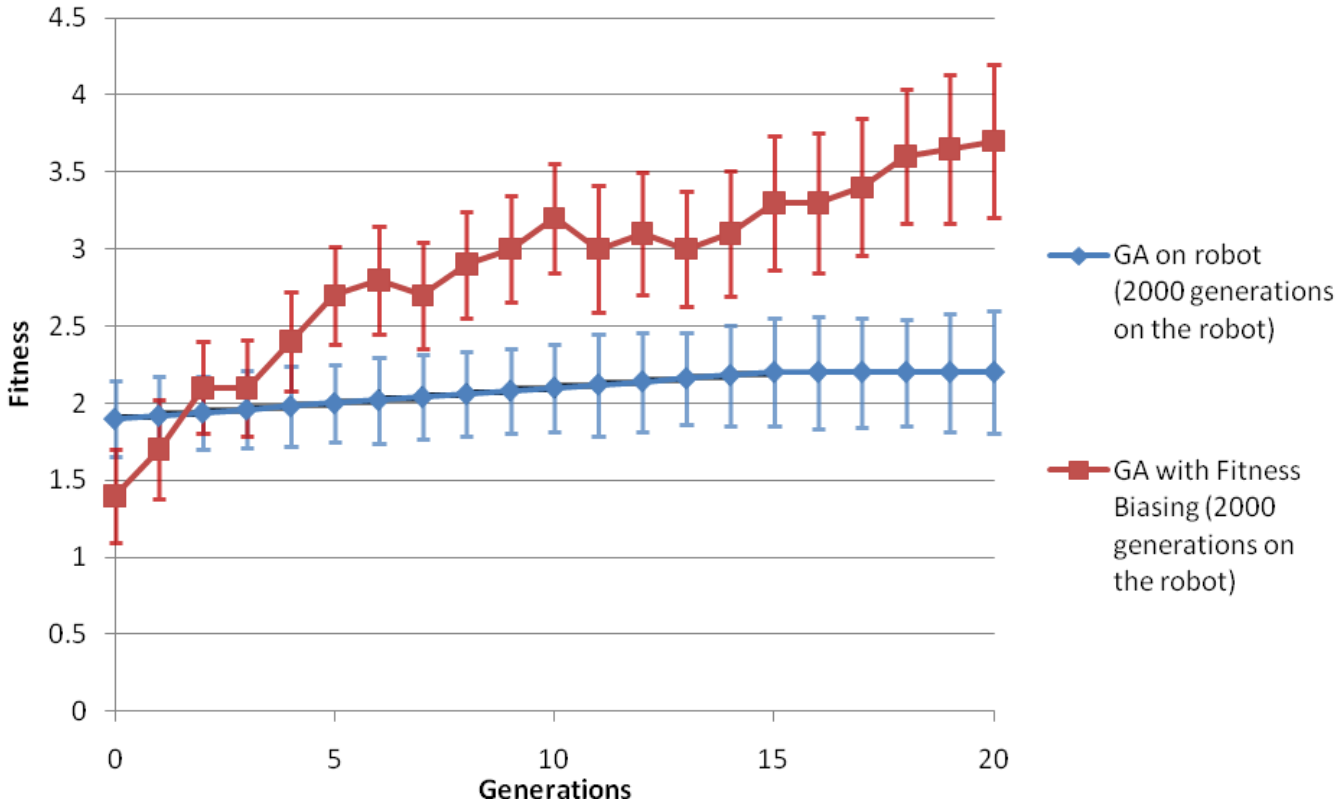


Figure 6. The genetic algorithm on the robot and the genetic algorithm with Fitness Biasing, measured in tests performed on the robot. Error bars represent standard deviation.

simulation; inaccuracies in the model can never be entirely overcome in a pure simulation method. Third, Fitness Biasing was tested by running the genetic algorithm in simulation with each of the model parameters being skewed, but having the system test every tenth population on the robot (for this phase of the research, this was on the simulation with accurate parameters). This was the most effective method of learning, due to the ability of Fitness Biasing to combine the strengths of learning on the simulation and learning on the robot. Due to the Fitness Biasing system running mostly in simulation, the long runtime and the wear on the robot that are present in learning on the robot are both significantly reduced. Additionally, periodic tests on the robot retain much of the accuracy absent from the pure simulation method.

The results of the three separate experiments are shown in Figure 5. As can be seen in this graph, the genetic algorithm operating directly on the robot produced the best results in two-hundred generations. However, this came at a price, since twenty thousand evaluations *200 generations * 100 individuals* were done on the robot. This would not only take a long time to get the results, it would be detrimental to the robot. The genetic algorithm operating only on the model would be the fastest and not require any tests on the robot, but the resultant solution was not very good. The genetic algorithm with Fitness Biasing required two thousand evaluations on the actual robot (one tenth of all tests on the robot), yet yielded effective results with a fitness near to the actual robot genetic algorithm. It's clear from these tests that Fitness Biasing produces a better controller than the train-on-model method without Fitness Biasing, when one considers the total set of generations. However, what may not be as clear from this graph is that Fitness Biasing is superior to testing entirely on the robot. Although nearly attaining the on-robot fitness after two-hundred generations, the main argument for Fitness Biasing is the extra time and wear caused by training solely on the robot.

Figure 6 helps to better compare the effectiveness of Fitness Biasing with a train-on-robot approach. It shows the genetic algorithm on the robot versus the genetic algorithm with Fitness Biasing when considering only tests on the robot. Over the full two hundred generations of training, the genetic algorithm with Fitness Biasing had only twenty generations of tests on the robot, so this is plotted against the first twenty generations of training for the genetic algorithm on the robot.

It's clear from this graph that the genetic algorithm with Fitness Biasing was the superior method when considering the number of tests required on the robot.

## VI. Conclusion

Tests done in simulation show that Fitness Biasing is an effective method for evolving a controller to complete the box pushing task. This is the first use of Fitness Biasing for problems other than gait generation. The system performed well for this problem requiring the proper responses to sensors in order to complete a task. In this work, we showed its efficiency in learning the task. What was not shown, but is one of the advantages of Fitness Biasing is that if there are changes in the robot or environment that affect the robot's performance, Fitness Biasing has the capacity of compensating for this while the robot is in operation. In future research, we will continue this work by testing the Fitness Biasing learning system on the actual robot operating in our colony space. In addition, we plan to change the capabilities of the robot during training to observe how it adapts and modifies the control program.

## References


[1] 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.

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

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

[4] 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.

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

[6] 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).

[7] 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 1999).

[8] Parker, G. (2004). "Fitness Biasing to Produce Adaptive Gaits for Hexapod Robots" Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems. (IROS 2004).

[9] Parker, G. (2004)."Punctuated Anytime Learning for Hexapod Gait Generation" Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2002).