Gait Evolution for a Hexapod Robot

Karen Larochelle, Sarah Dashnaw, and Gary Parker

Computer Science
Connecticut College
270 Mohegan Avenue
New London, CT 06320
<kjlar | sedas | parker> @conncoll.edu

Abstract

Gait generation is a major part of legged robot control. Learning gaits is a difficult problem because the simultaneous movement of all six legs must be coordinated. A sequence of commands sent simultaneously to each of the robot's leg actuators will produce forward movement. In order for this movement to be continuous, this sequence needs to be continually repeated. The success of this repetition is contingent on the smooth transition of the robot from one state to the next. Learning cycles of control activations (simultaneous commands to the leg actuators) for hexapod gaits has been done using Cyclic Genetic Algorithms. This form of evolutionary computation was shown to successfully generate gaits when the model used average values for the leg movement rates. Although using average rates resulted in fast convergence, the CGA could not fully exploit the capabilities of the robot. In this paper, we use a CGA working on a model with measured rates for all possible leg movements to generate gaits for a hexapod robot. These gaits are then successfully transferred to the actual robot.

Keywords: genetic, cyclic, robot, hexapod, gait, control, learning

1. Introduction

An important aspect of legged robot control is gait generation. Hexapod gaits require coordinating the simultaneous movement of six legs. To maintain sustained forward movement, the sequence of individual actions that propel the robot needs to be continually repeated. This puts the robot through several states in a cycle and that must be easily transitioned from one to the next. The problem is compounded when the robot's legs have differing capabilities. This also makes it difficult to transfer a solution from one robot to another, as the robots will inevitably vary in construction. These issues should be taken into account by the learning mechanism. Using some form of evolutionary computation operating on a model of the robot can cover these issues as long as the model is adequately accurate. However, since simulations are unable to completely represent an environment, the solution obtained by training on the model needs to be tested on an actual robot to verify it's viability.

Gait generation for hexapod robots has been addressed with several approaches. Rodney Brooks (Brooks,1986) used subsumption architecture in the development of his robots. In later work, he used a genetic algorithm to learn components of this architecture (Brooks, 1989). Randall Beer used a neural net to control a simulated cockroach (Beer,1990). Randall Beer and John Gallagher used a genetic algorithm as a learning mechanism by having it learn the weights in a pre-defined neural network (Beer,1992) and tested it on an actual hexapod robot (Gallagher,1994). These solutions proved to be successful for the robots used but we wanted a solution that could work on the most primitive of robots and required less 'a priori' knowledge of how to walk.

Graham Spencer (Spencer,1994) used genetic programming to develop gaits for a simulated robot using minimal knowledge about the intricacies of walking. Although his results were promising in that the system consistently learned gaits which generated sustained forward movement, we wanted a system that would learn the optimal gait for our model which represented an actual robot with a very primitive control interface and no means of feedback. In addition, we wanted it to be continually adaptive to robot capabilities and be a quickly converging algorithm suitable for any-time learning (Grefenstette, 1992). To this end, Cyclic Genetic Algorithms (CGAs) were developed.

In previous work, CGAs (described in Section 2) were used to develop a set of primitive instructions that can be repeated continually to produce a gait (Parker,1996). Gaits for a hexapod robot were learned in simulation and tested on an actual robot (Parker,1997). These tests showed that the CGA could successfully develop hexapod gaits using a model that had accurate measurements for the leg swing capabilities, but only average figures for the rate of movement. These rates were averages of the up/down and forward/back movement; all legs had the same rate capabilities. Tests using these rates showed that the CGA could produce good tripod gaits. Having the rates of all the legs the same reduces the complexity of the problem, but does not allow the CGA to exploit the peculiarities of a specific robot. This could put limitations on the solution. The best way to generate a gait specific to a robot is to use exact measurements and exact rates from the robot. This method, however, makes it harder for the CGA to develop a gait for the robot, as the definition of optimal is no longer clear. There are several possibilities for leg extensions and for how various leg extensions are used together. The search space is substantially increased, requiring the CGA to explore many more possible combinations in a search for the optimal solution.

In this paper, we use a CGA operating on a model with movement rates measured individually for each leg to learn gaits for a specific hexapod robot. Each leg in the model has separate values for rates from the robot for rate up, rate down, rate forward, and rate backward. The gaits learned in simulation are then downloaded onto the actual robot to confirm the solution's viability. A graph is provided that shows the learning curves in simulation and on the robot.

2. Cyclic Genetic Algorithms

Cyclic Genetic Algorithms were developed as a modification of the standard Genetic Algorithm first introduced by John Holland (Holland,1975). Genetic Algorithms use ideas from the laws of selection and survival of the fittest. They use the three standard genetic operations of selection, crossover, and mutation on a randomly generated population to search for an optimal solution to a problem. The population is a collection of individuals that are possible solutions to the problem and usually represented as a bit string of fixed length, which is called a chromosome. Whether or not an individual survives depends on how well its solution performs compared to the other individuals in the population, which is the solution's fitness.

Cyclic Genetic Algorithms (Figure 1) were designed to solve problems requiring a solution made up of a sequence of tasks that is to be continually repeated (Parker,1996). The genes of the CGA chromosome are tasks (as opposed to traits in a standard GA) and the chromosome is a cycle of bits (as opposed to linear bit string in a standard GA). In addition to the cycle, the CGA can have two tails. These tails allow for the completion of tasks during the pre and post-cycle procedures when necessary. In the case of gait generation, the pre-cycle could be used to position the legs in a ready to walk stance, and the post-cycle could be used to return the legs to a stable stance once the desired forward movement was complete. The genes of a CGA chromosome can be as simple as individual tasks that are to be completed in a specific amount of time up to complicated sub-cycles that can be trained separately on a CGA.

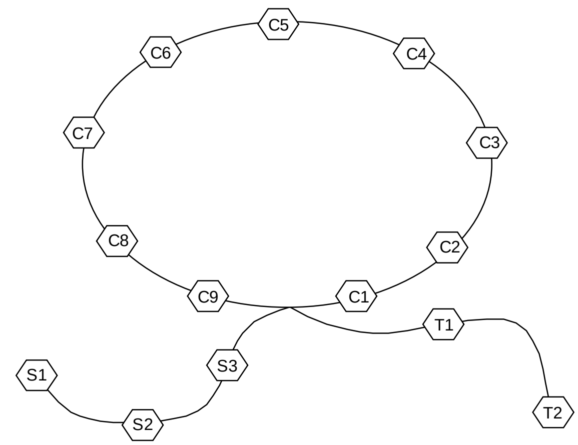

Figure 1: A CGA chromosome. It has a start section (in this case made up of three genes), an iterative or cyclic section, and a tail section.

CGAs can be represented by both fixed and variable length chromosomes. In order for the CGA to have the flexibility to complete a full cycle, the system must be capable of assigning the appropriate number of tasks to each phase. Variable length chromosomes allow for the maximum possible flexibility in this regard, but their variability can slow convergence. Using fixed length chromosomes, with the task plus the number of times it is repeated encoded in the gene, the flexibility of a variable length chromosome can be represented in a fixed length. However, the chromosome must be designed to have enough genes so that enough signal changes can occur to solve the problem.

For our tests, the chromosome was cyclic with no tails (previous tests showed that the best solutions were tailless). The genes of the chromosome were made up of two parts: activations (tasks delineating the movement of each actuator) and repetitions (the number of time to consecutively repeat the activation. The resulting trained chromosome will contain a cycle of primitive instructions that can be continually repeated by our robot's simple controller to produce a gait.

3. ServoBot

The robot used for our tests was a ServoBot (Figure 2). The ServoBot is a legged robot with two degrees of freedom per leg. Each leg has two servos, resulting in twelve servos that provide forward thrust and vertical movement. The robot used in these tests has a different servo placement than the ServoBots used in previously reported research (Parker,1997). The servos controlling vertical movement are installed in such a way that gives the robot increased potential to hold itself higher off the ground. The servos controlling the horizontal movement for the front pairs of legs on the current robot are further back from the legs, closer to the middle pair of legs, than in the previous robot. This difference in placement in the horizontal servos meant that the front legs' full extension capabilities had to be reduced to avoid collisions with the middle legs. Both of these changes made the gait control solutions significantly different from those learned in previous experimentation.

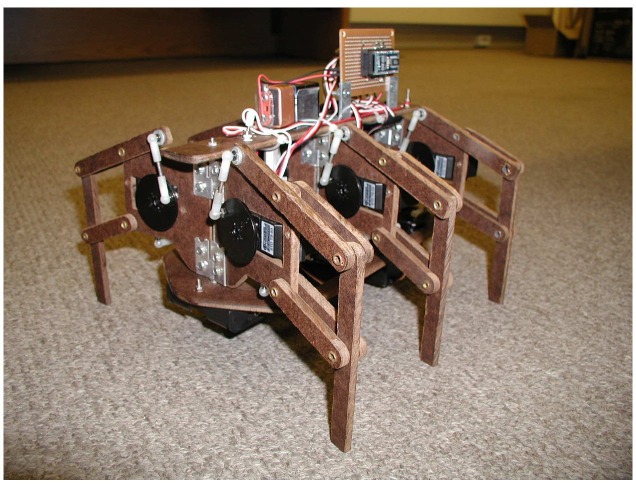

Figure 2: The ServoBot Robot used in Testing

A BASIC Stamp IISX controls the ServoBot. The ServoBot is powered by a 9 volt battery and either four 1.5 volt batteries carried on-board or an external power supply. The control program is written in a BASIC Stamp Editor on a PC and then downloaded onto the control chip via the PC serial port connection. Once the program has been downloaded, the ServoBot can be disconnected from the computer. The BASIC Stamp will store and execute the sequence of instructions in the control program. Each instruction in the sequence corresponds to an activation number and the number of repetitions for that activation.

An activation for the ServoBot is in the form of a 12-bit number, which controls the actuators of the robot. Each bit in the number corresponds to a servo. When the bit is set at 1, the leg is up if it is a vertical servo and back if it is a horizontal servo. When the bit is set at 0, the leg is down or forward, respectively. Figure 3 gives an example of an activation and shows the resulting movement on the robot. The bits of the activation number can be divided into six pairs, where each pair controls the leg's vertical and horizontal movement. The first bit of a pair represents the vertical movement and the second bit represents the horizontal movement. The legs on the ServoBot are numbered 0, 2, and 4 from front to back on the right side, and 1, 3, and 5 from front to back on the left side.

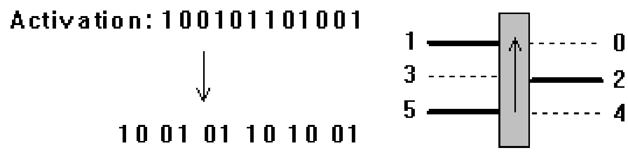

Figure 3: A Robot Activation and Its Results

A model of the ServoBot had to be simulated by the computer in order to allow the CGA to generate a gait. Measurements of each leg's capabilities were used in a simple data structure (Figure 4) that also recorded each leg's current state. These capabilities, combined with the input activations, were used to determine how much to change the current state of each leg. Each leg actuator was activated individually so that the leg's maximum throw could be recorded. These measurements were then used to fill the max leg position fields in the computer model. An average rate per activation was calculated for each leg's horizontal and vertical movement by dividing the maximum throw by the number of activations required to attain it.

In previous work, the rate up/down and rate forward/back were applicable to all legs, meaning there was just one value for the rate up/down and one for the rate forward/back. In the tests reported in this paper, measurements were taken from each of the legs to give six specific rate up values, six specific rate down values, six specific rate forward values, and six specific rate back values. These rates were then used in the computer model for the CGA to generate possible gaits.

current up - current vertical position of the leg
max up - position off the ground when completely up
max down - position off the ground when completely down
current back - current horizontal position of the leg
max back - position relative to completely forward when completely back
rate up - rate of vertical upward movement when servo activated
rate down – rate of vertical downward movement when servo activated
rate back - rate of horizontal backward movement when servo activated
rate forward – rate of horizontal forward movement when servo activated

Figure 4: Model Data Structure

4. CGA Applied to Gait Generation

4.1 Chromosome

The fixed length chromosome (Figure 5) is a list consisting of the individual's global coordinators and inhibitors and a 12 gene cyclic section. The genes in the cyclic section were each made up of two integers: the activations integer (12 bit number) and the repetitions integer (8 bit number).

$$(C\ \ I\ \ ((A\ R)_1\ (A\ R)_2\ (A\ R)_3\ (A\ R)_4\ (A\ R)_5\ (A\ R)_6\ (A\ R)_7\ (A\ R)_8\ (A\ R)_9\ (A\ R)_{10}\ (A\ R)_{11}\ (A\ R)_{12}))$$

Figure 5: Fixed Length CGA Chromosome with Global Coordinators & Inhibitors

Twelve in the iterative section was used because it was thought to be enough move changes to handle every possibility (two per leg for the six-legged). Coordinators and inhibitors were a part of the robot's coordination, which could evolve to increase leg control and proper movement. They were initiated as random numbers and were learned by the algorithm.

Coordinators were 12 bit numbers that directed the coordination of individual leg movement. This number could be looked at as six pairs of bits, one pair for each leg. The first being the back-down coordinator which, if activated, ensured that the leg would be down or moving in that direction if it was moving back. The second bit was the forward-up coordinator, which ensured that the leg would be moving forward if it was up.

The inhibitors affected pairs of legs. They prevented pairs of legs from moving back at the same time. The 2,3 inhibitor prevented both legs 2 and 3 from going back at the same time. It allowed 2 to move back, but inhibited 3. The inhibitors were stored in a single 15 bit number (one bit per possible pair). This 15 bit number could be thought of as 5 groups. The first group made up of 5 bits indicated which legs would be inhibited from moving in the same direction as leg 0. Five bits were required to cover the remaining legs 1 through 5. The second group was made up of 4 bits showing what legs would be inhibited from moving in the same direction as leg 1. Since leg 0 had already been matched with all legs in the first group it does not appear in the second. This continues until all possible leg matchings have been addressed.

4.2 Genetic Operators

The probability for selection was based on fitness, which was computed by estimating the millimeters of travel after 1000 activations. The was determined on the model by:

1) taking the current state of legs
2) applying the vertical movement of the activation
3) calculating the balance and probable legs on the ground from the model's current vertical position of each leg
4) applying the horizontal movement of the activation to alter the leg's state, but only counting legs on the ground in computation of the movement (fitness)
5) taking off some deduction for lack of balance and/or asymmetry of movement
6) repeat using the next activation and the new state of the robot

This was sequentially done from activation to activation and repeated as many times as required in the cyclic section to reach a total of 1000 activations. Using this fitness, the best individual was preserved; the rest of the new population was formed by stochastic selection of mates with the probability of selection proportional to the fitness.

Crossover was done by randomly picking corresponding spots in the chromosome. In the cyclic section, since it could be considered a circle, crossover was performed at two points (equivalent positions in both chromosomes). The effect was to swap sections within the circle. An alternate type of crossover was a gene-by-gene crossover, which would perform crossover in each of the corresponding genes of the two chromosomes. In the case where these genes were represented as lists, crosses could happen between the individual members of the list or within the bits of the specific numbers in the list.

Two types of mutation were used. *Gene replace* gave each gene a random chance of being replaced by a new completely random gene. With *gene mutate* each part of the gene had a random chance of having one of its bits altered.

Gene-by-Gene Evaluation was a clean up operator that randomly picked one or two individuals from the population on each set of trails and examined each gene one at a time. Genes were evaluated on the whole and move-by-move by comparing the previous move fitness to the present. Genes that were worse than a preset minimum were eliminated. Genes that were good in the execution of their early repetitions and subsequently dropped below a threshold in the later repetitions were modified by reducing their repetitions. Genes that had zero repetitions were moved out so that only active genes were at the start of the iterative section. Following these eliminations, if the number of genes or the total number of gene repetitions fell below some threshold, additional random genes were added until the thresholds were met.

5. Tests

Tests were run to determine if a plausible hexapod gait could be developed using the actual rates from the robot. The testing done in simulation and on the robot used the actual measurements for each leg of max up, max down, max back, rate up, rate down, rate back, and rate forward gathered from the robot. The tests were run on five distinct randomly generated populations for 5000 generations. Each population was made up of 64 individuals (chromosomes). In order to see the progression of learning, intermediate generations looking at the 10^{th}, 100^{th}, 200^{th}, 500^{th}, 1000^{th}, and 2000^{th} generations were saved to file.

To determine the fitness of each resulting population, performance tests were run in simulation. The simulated forward movement produced by the generation's best individual for each of the intermediate generations was computed using the computer model. These gaits were then downloaded and tested on the actual robot to determine if the solution was transferable. The distance traveled by the robot in each of these tests was recorded.

6. Results

Figure 6 shows the simulated and actual values for each generation averaged over the five populations. The horizontal axis represents the generations completed and the vertical axis represents the distances traveled at each generation in centimeters.

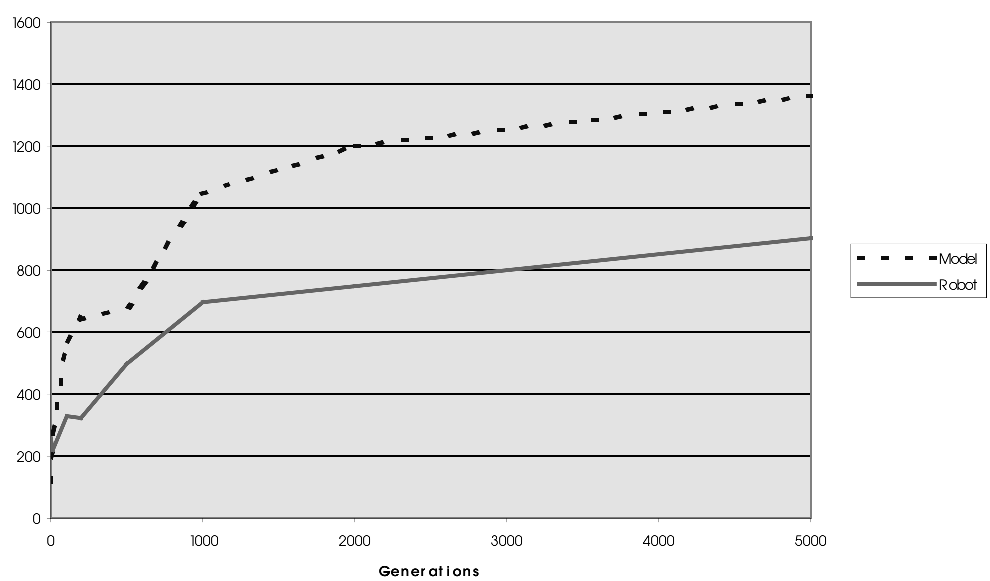

Figure 6: Gait Learning Tests in Simulation and on the Actual Robot Averaged over Five Trials

The results showed that using the CGA learning on a model having the actual rate measurements for the legs was successful both in simulation and on the actual robot (Figure 6). In simulation the results showed a steady improvement in the distances traveled over the 5000 generations. In most cases, the makings of a tripod gait were visible by the 100th generation, yet negative or stationary movements caused by awkward leg arrangements during gait phase transitions hindered the success of the gaits. In three of the five cases, near optimal tripod gaits were achieved by 2000 generations. Figures 3 and 7 illustrate the tripod gait. The two outermost legs on a side work with the middle leg on the opposite side, forming a triangular shape that maintains static stability. Once the tripod gait was developed in these three populations, any remaining irregularities in the stepping pattern were meticulously smoothed out. In two of the five cases, the gaits were not predominantly tripod, but they still resulted in increased forward movement. The nature of these gaits is described later in this section.

Once the simulations were complete, the gaits were downloaded and tested on the robot. The results of the actual testing on the robot showed a similar learning curve to the simulated results although the fitness was consistently less (Figure 6). During actual tests of the group of three tripod gait populations, the tripod nature was not seen until around 200 generations. What may have contributed to the robot's inability to produce a tripod gait when the simulated results showed one was the inaccuracy of the model in the computer. Not all outside forces were transferred to the computer environment. For example, the simulation did not take into account gravity or variances in friction. The weight of the robot was ignored. In simulation, the center of gravity of the model was not as close to the ground as the robot was in actuality. Though the model fully utilized the capabilities of its legs, once the actual robot began walking, leg extension was hindered by the weight of the robot. For this reason, often when a gait was expected to perform well, it was either stationary or experienced dragging of the legs. A tripod or near tripod gait was reached by the 1000th generation. In some cases, after a tripod gait was found, a slight decrease in the distance traveled would occur. This can be attributed to the model perceiving that the most optimal solution would require using the maximum vertical and horizontal movement possible although the optimal thrust could actually be found without utilizing the entire range of motion. When this occurred, it would take the robot longer to go the same distance.

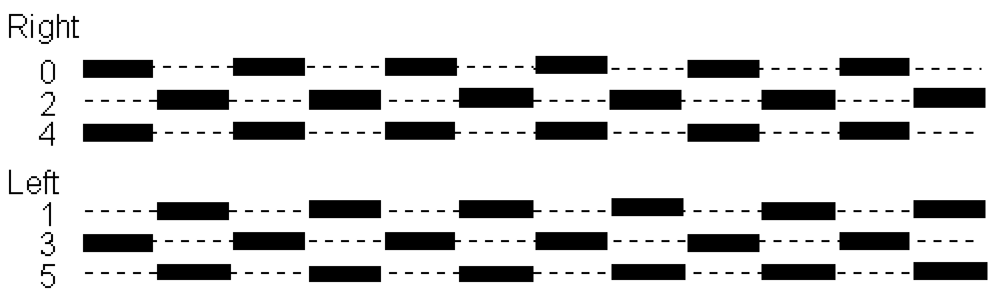

Figure 7: Representation of the tripod gait produced for the ServoBot using CGAs. The diagram shows a timeline of what all the feet are concurrently doing. The dark blocks indicate times that the legs are in swing (off the ground) and the absence of a block means the leg is in stance (providing thrust). At least three legs are on the ground at all times. This gait is statically stable.

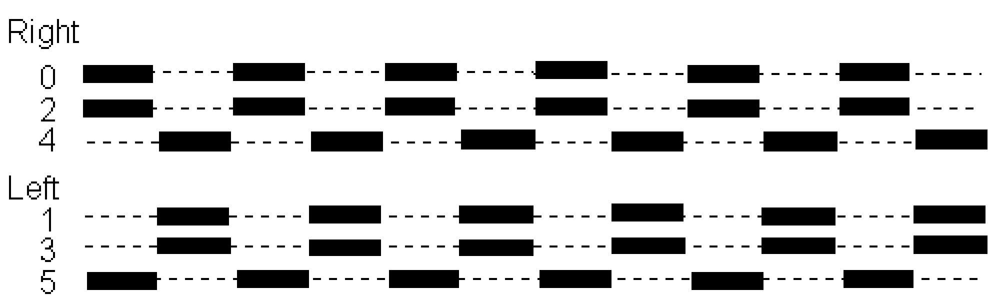

Figure 8: Alternate Gait Developed by the CGA for the ServoBot

The first alternate gait developed can be seen in Figure 8. This gait uses the front and middle leg on one side with the back leg on the opposite side of the body. Although this is not a tripod gait, it was still able to perform at virtually the same level as the tripod, resulting in a similar learning curve. The second alternate gait incorporates aspects from both the previous gait and the tripod gait. First it uses the front left leg and the middle and back right legs. Then it uses the front and middle right legs and the back left leg. The last phase uses a tripod arrangement, but is unable to fully develop the tripod gait. Instead it cycles back, repeating the three phases. While this gait was not as clean as the other four, it still produced reasonable forward progress.

7. Conclusions

Our tests illustrate that the Cyclic Genetic Algorithm was able to produce reasonable gaits, both in simulation and in tests done on the robot. Due to the lack of external forces in the model such as accurate friction and gravity, the simulation regularly performed better than the tests that were done on the robot. Once the gaits were downloaded onto the robot and tested, our results showed that, in three of the five tests run, the model was able to achieve a tripod gait. In the two cases where a tripod gait did not develop, the tests produced alternate gaits that showed positive learning curves. The CGA was successful in generating a gait using the actual values for rate up, rate down, rate back, and rate forward that could be applied to the hexapod robot. This shows that CGAs can learn despite the increased complications of a model with specific rate information and can successfully be used to develop gaits for individual robots.

Acknowledgments

This research was supported in part by funds provided by Keck Undergraduate Science Program and the Jean C. Tempel '65 Scholarship Fund.

References

1. Beer, R. (1990). *Intelligence as Adaptive Behavior*. Academic Press, Inc., Boston.
2. Beer, R., and Gallagher, J. (1992). Evolving Dynamical Neural Networks for Adaptive Behavior. *Adaptive Behavior*, 1 (pp. 91-122). Cambridge, MA: MIT Press.
3. Brooks, R. (1986). A Robust Layered control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, Vol 1, Number 1.
4. Brooks, R. (1989). A Robot That Walks: Emergent Behaviors from a Carefully Evolved Network. *Neural Computation* (pp. 254-262).
5. 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.
6. Grefenstette, J. and Ramsey, C. (1992). An Approach to Anytime Learning. *Proceedings of the Ninth International Conference on Machine Learning*, (pp. 189-195).
7. Holland, J. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: The University of Michigan Press.
8. Parker, G., and Rawlings, G. (1996). Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots. *Proceedings of the World Automation Congress (WAC'96), Volume 3, Robotics and Manufacturing Systems*. (pp. 617-622).
9. 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).
10. Spencer, G. (1994). Automatic Generation of Programs for Crawling and Walking. *Advances in Genetic Programming*. (pp. 335-353) K. Kinnear, Jr. (ed.), Cambridge, MA: MIT Press.