Cyclic Genetic Algorithm with Conditional Branching in a Predator-Prey Scenario

Gary Parker
Computer Science
Connecticut College
New London, CT 06320
parker@conncoll.edu

Ivo Parashkevov
Computer Science
Connecticut College
New London, CT 06320
iipar@conncoll.edu

Abstract - *In its traditional form, the cyclic genetic algorithm (CGA) was found to be a successful method for evolving single loop control programs for legged robots. Its major limitation was the inability to allow for conditional branching, which is required for the integration of sensor inputs in the controller. In recent work, we extended the capabilities of CGAs to evolve multi-loop programs with conditional branching. The design proved successful for the evolution of a controller that allowed a robot to efficiently search for a static target in a square area. In this paper we increase the complexity of the experiment and demonstrate the capability of CGAs with conditional branching to generate a controller for the predator in a predator-prey scenario.*

Keywords: Evolutionary robotics, learning control, program generation, genetic algorithm, hexapod

1 Introduction

Evolving controllers for autonomous legged robots can reduce the time needed for initial program development, but more importantly, is a means for learning adaptive control in a changing environment. In past research, control programs for legged robots have been evolved using evolutionary computation to learn the weights of an artificial neural network, genetic programming, and cyclic genetic algorithms.

One of the most common methods of learning robot control is through the use of a genetic algorithm (or some other form of evolutionary computation) to learn the connection weights and/or architectures for artificial neural networks [1]. Beer and Gallagher [2] demonstrated that genetic algorithms can be used to evolve effective neural networks, and successfully evolved chemo-taxis and legged locomotion controllers. Floreanno and Mondada [3] evolved neural networks to control homing and navigation on a Khepera robot. The robot's task was to navigate through a corridor while performing obstacle avoidance and locating a charging station before the robot's batteries lost power. Lund and Orazio [4] evolved a neural network controller for a Khepera robot capable of avoiding walls and obstacles in an enclosed area.

Another method for learning robot control is through the use of a genetic programming (GP). Busch et al. [5] used GP to evolve robot controllers to produce gaits for simulated robots. This system was able to produce gaits for robots independent of their specific morphology. Lazarus and Hu [6] used GP to integrate sensor input into the development of controllers for simulated robots with sensors that were performing wall-following and obstacle avoidance tasks. Nordin et al [7] also evolved wall-following agents which performed successfully both in simulation and on a Khepera robot.

The *cyclic genetic algorithm* (CGA) was developed [8] with the purpose of facilitating the representation of loops in the chromosome. As a variant of Holland's genetic algorithm [9], it employs the same basic genetic operators – selection, crossover, and mutation – and applies them to a randomly generated set of solutions to a given problem, in order to eventually obtain a set containing a near-optimal solution. Traditionally, the genes of GA chromosomes represent traits or parameters of the solution to a given problem. In a CGA chromosome, the genes represent tasks to be completed by the agent. These tasks could be single actions, or sub-cycles of actions. In the latter case a gene is divided in two distinct sections, the first representing the action to be taken, and the second – the number of times that the action is to be repeated. The entire set of genes in the chromosome can be executed repetitively, in which case the chromosome itself becomes a cycle. Thus, a CGA allows for the implementation of loops on two different levels.

CGAs were successfully used in the past to evolve single-loop robot leg cycles [8], gait cycles for hexapod robots [10], and area coverage patterns [11]. However, they were not suited to allow for dynamic changes of behavior based on sensor information. To address that limitation we developed the CGA with Conditional Branching (CGA/CB), designed to evolve a multi-loop control program that would switch from one loop to another depending on changes of the environment [12]. The design was successfully tested in simulation – we were able to evolve a controller that could handle inputs from two sensors.

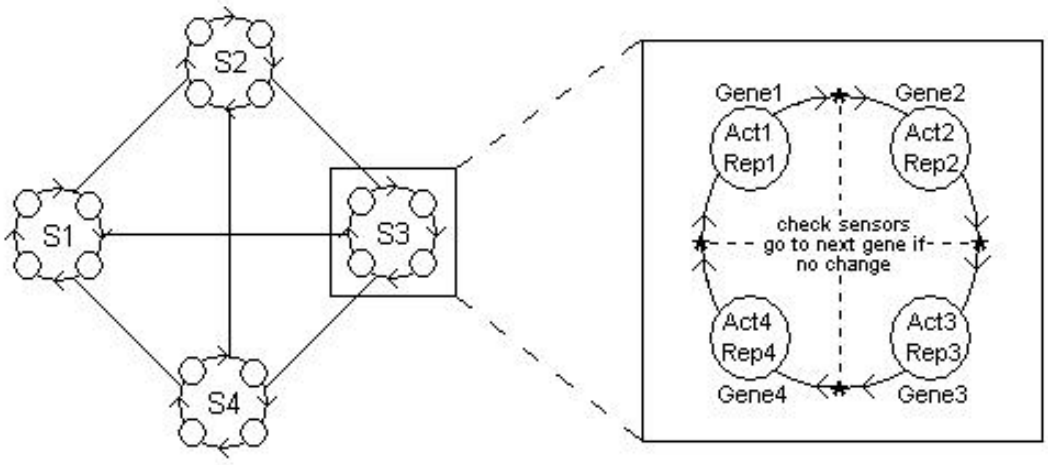

Figure 1. CGA with Conditional Branching Chromosome and a detailed view of one of its segments. The example chromosome shown contains 4 segments. Each segment is a loop.

2 Previous Work

In our previous research [12], we addressed the CGA's major drawback and developed the CGA with Conditional Branching (CGA/CB). It was designed to evolve multi-loop programs that would switch from one loop to another depending on sensor inputs. The method was successfully tested in simulation: a robot controller was evolved that enabled a hexapod robot to efficiently find a stationary target in a square area.

The chromosome was divided into separate segments (Figure 1, originally published in [12]). Each segment represented a control loop, a cycle that the robot repeated as long as the sensor inputs stayed the same. There was a segment for every possible combination of sensor inputs, and each segment was linked to all the others.

The segments, in turn, were divided into genes. Each gene was a simple loop, and was represented by a pair of integers. The first integer determined which action was to be taken by the robot (all possible actions being listed in a table), and the other indicated the number of repetitions of that action. After executing one gene by performing the respective action the specified number of repetitions, the robot checked the state of its sensors. If the sensor inputs were the same, it continued with the next gene of the same segment. If control reached the last gene, it simply started over with the first one – each segment was a loop. If the sensor inputs changed, the control jumped to the first gene of the segment that corresponded to those inputs.

This design was employed to evolve a controller of a robot charged with the task of finding a randomly located target in a square area. It was equipped with four sensors – two to detect the presence of a wall, and two to detect the target. In this previous research the two wall sensors were implemented in the chromosome, as the simulation ended as soon as the robot was able to find the target. Since we needed a segment for all possible combinations of sensor inputs, we had a total of four segments.

After evolving 2048 generations, the fittest individual was able to find the target with significantly fewer steps than the fittest individual of the random population. It had successfully learned to avoid the walls of the search area and employed a zigzag motion that enabled it to maximize the area covered by its sensors [12].

3 Simulation Details

In extending our previous experiment, we kept many of the simulation details the same. Instead of a static target, however, we have another robot with a preprogrammed controller that is actively trying to avoid the predator.

The predator and the prey are a simulation of the ServoBot, which was developed by David Brown at Indiana University for legged robot and colony experimentation. It is a small, inexpensive hexapod robot constructed from Masonite (hard-pressed wood). Equipped with a BASIC Stamp II for coordinating the motion of all six legs, it has some of the characteristics of larger, more complex robots.

The simulation area is square with each wall 500 units in length. There are no obstacles. The robots' positions in the simulation area are fully described by their X and Y coordinates, as well as a number between 0 and 359 that

shows the direction of their heading; 0 means heading South, 90 – East, 180 – North, and 270 – West.

Both the predator and the prey are allowed to take one of thirty-two possible actions or steps at a time. Each of these steps represents one complete gait cycle [11,12], defined as the timed and coordinated motion of the legs of a robot such that the legs return to the positions from which they began the motion. The resultant position of the agent after executing a full gait cycle is a simulation of the measured movements of an actual ServoBot. It is calculated from a table of stored values that show the change of the robot's X and Y coordinates and heading. In the table of values, fifteen of these gait cycles result in a left turn, another fifteen result in a right turn, and one gait is designed to move the robot straight forward. One remaining option is for the robot to remain motionless for the time it takes to complete a full gait cycle. In the selection of gait cycles, the robot controller faces a trade-off between large displacement and small rotation, or vice versa.

The predator is equipped with two types of sensors – one that would enable it to detect the presence of a wall and the other to detect the target. It has two sensors of each type for a total of four. Since each sensor has two possible states – 0 (inactive) and 1 (active), there are sixteen possible combinations of sensor inputs. The activation distance for each sensor is 80 units and the span of its vision is 45 degrees. The two sensors of each type are situated at the front of the predator, with their spans overlapping by 10 degrees. Thus it has a total sight span of 80 degrees and can detect walls or the prey at a maximum distance of 80 units away.

The prey has a preprogrammed controller. The range distance of its sensors is the same as that of the predator – 80 units. However, it is equipped with a span of 360 degrees of vision and is capable of identifying the exact position of any object (the predator or a wall) relative to its own position. This represents a significant advantage over the predator, as the latter is only capable of estimating the relative position of the objects, and its vision span is only 80 degrees. When in danger, the prey runs away from the predator, and stays away from walls, so that it does not get cornered. If it is near any object (the predator or a wall), it will calculate the best escape path, and will take the action that would take it as far away from that object as possible. If it manages to escape and is in a state where it no longer detects any object, it will still remember the direction of the last object seen, and will take five more steps in the opposite direction. After those five steps, unless there is a new danger detected, the prey will not be "scared" anymore. In that case, there is a 10% chance that it will move with a random gait cycle. Otherwise it simply stands still.

Thus modeled, the prey is superior to the predator. However, if the predator evolves the ability to chase it well enough, it will keep decreasing the distance between them whenever the prey has to make a turn, and will eventually be able to capture it. A "capture" occurs when the prey is within the span of vision of the predator and is 10 units or less away from it.

The predator and the prey started each simulation run from random positions. The simulation ended as soon as the predator captured the prey or took more than the maximum limit of 400 steps.

4 Evolution

In order to successfully capture the prey, the predator needed to learn two major patterns of behavior – searching and chasing. The evolution of the latter, however, was contingent on the successful evolution of the former, as the predator would never have the chance to learn to chase the prey if it could not find it often enough. To ensure that the predator did in fact find the prey often (and early) enough, we used an incremental approach to learning. The control program to search was learned first [12]. The resultant best individual was then used to generate a population to continue evolution to learn both search and capture.

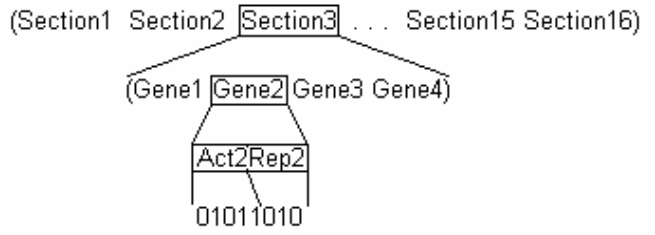

Figure 2. The chromosome, with its 16 segments. Each segment consists of four genes that are to be executed consecutively. Each gene contains information about the action and the number of repetitions it is to be performed.

4.1 Chromosome

The chromosome of each individual consisted of sixteen segments (Figure 2, from [12]), as there were sixteen possible combinations of sensor inputs. Twelve of the segments were randomly generated for every individual in all initial populations. The remaining four were taken from the chromosome of the individual that performed best after the evolution of the search in our previous work. Thus, for the sensor inputs equivalent to *no_object_detected, left_wall_only, right_wall_only, left_and_right_wall_only* each individual had the same four segments before training began. However, all sixteen segments were subject to mutation and crossover, so all sixteen of them were evolved.

To keep the design of the chromosome consistent, each segment had four genes. Each gene was 8 bits long (5 bits to represent the action number and 3 bits to represent the number of repetitions). As a result, each segment was 32 bits; the chromosome had a total of 512 bits.

4.2 Training

Training was done on five populations, each consisting of 256 individuals, with chromosomes initially created as described above. Evolution was carried out for 10240 generations. Each individual was evaluated ten times, from ten randomly generated starting positions. All individuals from the same generation ran the simulation from the same ten positions. The fittest individual from each generation was automatically included in the next generation of training. The rest were produced through the application of the CGA operators. The populations at periodic generations from 0 up to 10240 were saved during training.

4.3 Fitness Function

The fitness score was assigned to each individual based on its average performance over all ten runs of the simulation. To evaluate the performance of the predator for each simulation run, we took the maximum number of steps the predator was allowed to take (400), and subtracted the number of actual steps taken when the prey was not in sight. In case of a capture, we added a capture bonus of 125. The final result was raised to the power of 1.5 to amplify the differences between different individuals. The resulting function gave credit for chasing the prey since no points were subtracted if the prey was in sight after a given step. It gave a very high bonus for a capture, which was the eventual goal. In addition, an individual ended up receiving more points for fast, as opposed to slow captures, as slow captures usually meant more steps taken with the prey out of sight.

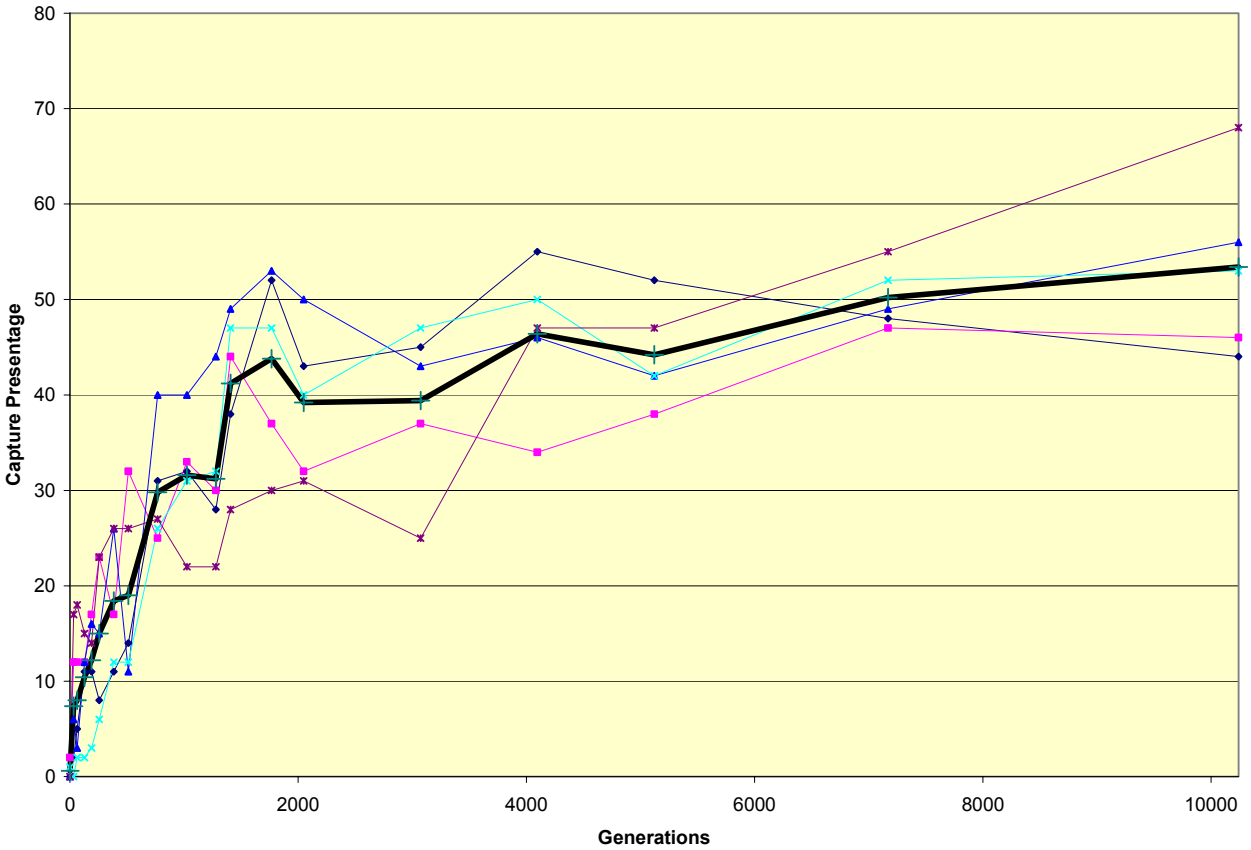

Figure 3. Learning curve for five tests of the CGA in learning predator control for the predator-prey problem, expressed in terms of the capture rate of the best individual for the respective generation. The bold line shows the average of all five tests.

5 Results

The results of all five runs, plus the average are shown in Figure 3. These are tests using stored data at each of the specified generations that were run after the training was complete. The plot shows the percentage of captures of the best individual for each of the five populations at each of the training generations where the population of solutions was saved. The results show the success of the training system as all five tests result in significant improvements in the control program as training continues. The initial random programs resulted in a capture an average of less than one percent of the time. After training, the control program that was generated resulted in an average capture rate of over 50 percent.

The predator quickly learns to go after the prey when the prey is right in front of it. This explains the fact that it is able to score a few captures (out of a hundred trials) rather early in the course of the evolution. These captures are mainly due to fortuitous circumstances, such as when the prey runs into the predator head on and has no time to turn around and start running away.

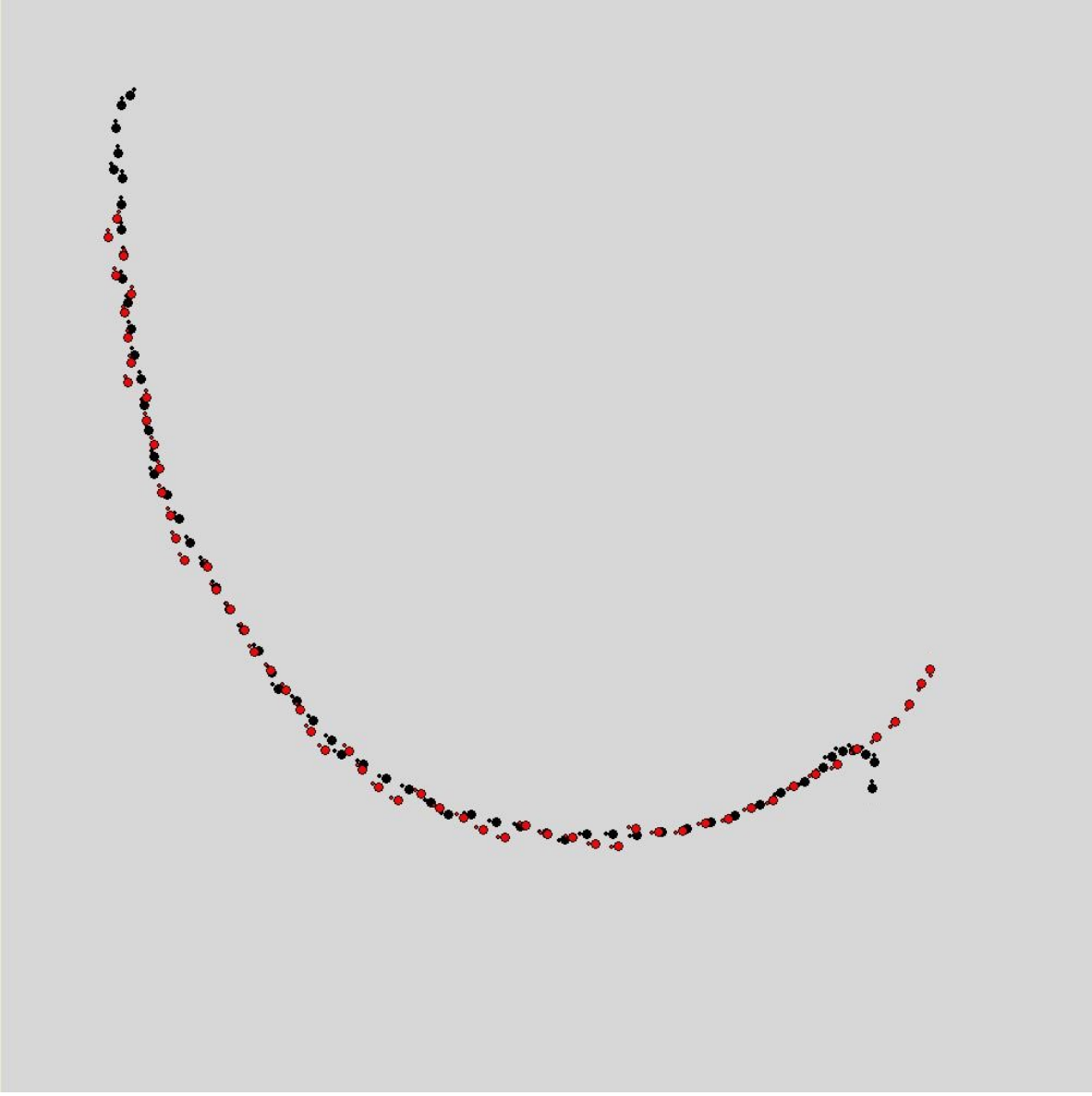

Figure 4. The predator (lightly shaded) goes after the prey as soon as its sensors are able to detect it and stays with it as it turns.

In later generations, the predator learns to chase the prey effectively as the latter changes directions. Figure 4 shows an example of a portion of a run where the predator successfully follows the prey. Its response to the sensor inputs is finely tuned so that its trajectory matches that of the prey.

The best individual obtained using our design method captured the prey 68 times out of a hundred. Although not a perfect solution, this controller is a dramatic improvement over the initial population.

6 Conclusions

Our results demonstrate that the CGA/CB design is viable for relatively complex tasks. The predator in our predator-prey scenario learned to chase and improved dramatically in its ability to capture the prey, despite the latter's superiority. The CGA/CB is capable of generating a multi-loop program for robot control and we speculate that it can be used for other applications that require multi-loop programs where there is a known limit to the number of loops.

One limitation of the CGA/CB is that it requires a segment of the chromosome that fully describes the behavior of the agent for any given combination of sensor inputs. The fact that the size of the chromosome grows exponentially as we increase the number of sensors to integrate makes us believe that there is great room for improvement on the way sensor inputs are handled. One possibility for future work would include the use of a neural network, in conjunction to the CGA/CB, that would map more than one combination of sensor inputs to the same segment of the chromosome. Such a design would require evolution of chromosome segments that can be applicable for more than one situation.

References

[1] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, Vol. 87, No. 9, 1999, pp.1423-1447.

[2] R. D. Beer and J. C. Gallagher, "Evolving Dynamical Neural Networks For Adaptive Behavior," *Adaptive Behavior,* Vol. 1, No. 1, 1992, pp. 91-122.

[3] D. Floreano and F. Mondada, "Evolution of Homing Navigation in a Real Mobile Robot," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 26, No. 3, 1996, pp 396-407.

[4] H. H. Lund and O. Miglino, "From Simulated to Real Robots," *Proc. IEEE Third International Conference on Evolutionary Computation*, NJ, 1996.

[5] J. Busch, J. Ziegler, C. Aue, A. Ross, D. Sawitzki, and W. Banzhaf, "Automatic Generation of Control Programs for Walking Robots Using Genetic Programming," *EuroGP 2002*, LNCS 2278, 2002, pp. 258-267.

[6] C. Lazarus and H. Hu, "Using Genetic Programming to Evolve Robot Behaviours," *Proc. Third British Conference on Autonomous Mobile Robotics & Autonomous Systems*, Manchester, UK 2001.

[7] P. Nordin, W. Banzhaf, and M. Brameier, "Evolution of a World Model for a Miniature Robot using Genetic Programming," *Robotics and Autonomous Systems*, Vol. 25, 1998, pp. 105-116.

[8] G. B. Parker, "Evolving Leg Cycles to Produce Hexapod Gaits," *Proc. World Automation Congress*, Vol. 10, Robotic and Manufacturing Systems, 2000, pp. 250-255.

[9] J. H. Holland, *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI, The University of Michigan Press, 1975.

[10] G. B. Parker and G. J. E. Rawlins "Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots," *Proc. World Automation Congress*, Vol. 3, Robotic and Manufacturing Systems, 1996, pp. 617-622.

[11] G. B. Parker, "Learning Control Cycles for Area coverage with Cyclic Genetic Algorithms," *Proc. Second WSES International Conference on Evolutionary Computation*, 2001, pp. 283-289.

[12] G. B. Parker, I. I. Parashkevov, H. J. Blumenthal, and T. W. Guildman, "Cyclic Genetic Algorithms for Evolving Multi-Loop Control Programs," *Proceedings of the World Automation Congress (WAC 2004),* June 2004.