CYCLIC GENETIC ALGORITHMS FOR EVOLVING MULTI-LOOP CONTROL PROGRAMS

GARY B. PARKER, CONNECTICUT COLLEGE, USA, parker@conncoll.edu
IVO I. PARASHKEVOV, CONNECTICUT COLLEGE, USA, iipar@conncoll.edu
H. JOSEPH BLUMENTHAL CONNECTICUT COLLEGE, USA, hjblu@conncoll.edu
TERRENCE W. GUILDMAN, CONNECTICUT COLLEGE, USA, twgul@conncoll.edu

ABSTRACT

The Cyclic Genetic Algorithm (CGA) has proven to be an effective method for evolving single loop control programs such as ones used for gait generation. The current limitation of the CGA is that it does not allow for conditional branching or a multi-loop program, which is required to integrate sensor input. In this work, we extend the capabilities of the CGA to evolve the program for a controller that incorporates sensors. To test our new method, we chose to evolve a robot in simulation that is capable of efficiently finding a stationary target.

KEYWORDS evolutionary robotics, genetic algorithms, sensors, learning, search, hexapod

1. INTRODUCTION

The Cyclic Genetic Algorithm (CGA), a variant of the traditional GA, has successfully been used to evolve single loop control programs for hexapod gaits and area coverage path planning. However, in order for a robot to react properly to sensor input, the controller must be running a multi-loop program, which is only possible if a system of conditional branching can be implemented. In this paper we modify the gene structure of the CGA chromosome to implement a system of conditional branching. To show the success of a CGA evolving multi-loop control program we chose a search task in which an agent is charged with locating a randomly placed target. To test our new method, we chose to evolve a robot in simulation that is capable of efficiently finding a stationary target. The evolved behavior must enable the robot to properly interpret sensor input to avoid walls and locate the desired target. The agent modeled in simulation is a hexapod robot equipped with four sensors. The task of learning search behavior for autonomous robots has been approached in several ways.

A common approach to evolve intelligent agents is to use Genetic Programming (GP) [1]. Busch et al. [2] used GP to evolve robot controllers to produce gaits for simulated robots. This SIGEL system was able to produce gaits for robots independent of their specific morphology. GP has also been used to integrate sensor input into a learning system. One example is Lazarus and Hu [3] who simulated robots with sensors to perform wall-following and obstacle avoidance tasks. Nordin et al [4] also evolved wall-following agents which performed successfully both in simulation and on a Khepera robot.

Another method for learning controllers responding to sensor input is the evolution of an artificial neural network for autonomous agents. This method involves employing the evolutionary algorithm to evolve connection weights and/or architectures for artificial neural networks [5]. Beer and Gallagher [6] demonstrated that genetic algorithms can be used to evolve effective neural networks, and successfully evolved chemo-taxis and legged locomotion controllers. Floreanno and Mondada [7] 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. This was a challenging task given that the robot was never told the coordinates of the charging station nor did the fitness function directly reward the individual for reaching the station itself. Lund and Orazio [8] evolved a neural network controller for a Khepera robot capable of avoiding walls and obstacles in an enclosed area. They then successfully transferred the neural network control system from simulation to the actual Khepera robot for further evolution.

The goal of our work is to use a CGA to evolve the controller for a hexapod robot equipped with four sensors to enable the agent to search a given space while avoiding the walls. Cyclic Genetic Algorithms have been applied to several areas of research, but have not yet been used to evolve control programs which incorporate sensor input. In this paper, we propose a method of arranging the genes of the CGA chromosome such that it can implement conditional branching to allow for the processing of sensor information in a multi-loop program.

2. CYCLIC GENETIC ALGORITHM

The genetic algorithm (GA) was introduced by John Holland [9] as a parallel search algorithm which simulated the process of natural selection and survival of the fittest. In a GA, the three main genetic operators of selection, crossover, and mutation are used to evolve a randomly generated population into a set that includes a near optimal solution. Each individual in the population represents a possible solution to the problem, usually encoded as a bit string called a chromosome. Groups of bits in this chromosome can be considered to be traits of the solution. Every individual in the population is evaluated objectively using a mathematical formula called a fitness function. The probability of selection for crossover is based on the individual's fitness relative to the other members of its population. Crossover is applied to two selected individuals by splitting the chromosomes in a random place, and combining the first part of one of the chromosomes with the second part of the other. Mutation is then applied to the new chromosome and a random bit or collection of bits can be altered based on some probability.

A Cyclic Genetic Algorithm (CGA) is much like a regular GA except that the gene groupings of the chromosome represent tasks to be completed as opposed to traits of the solution. These tasks can be anything from a single action to a sub-cycle of tasks. Using this method of representation, it is possible to break up a chromosome into multiple genes with each gene acting as a cycle. Using this method of representation, it is possible to break up a chromosome into multiple genes with each gene acting as a cycle. Each gene or sub-cycle contains two distinct sections, one part representing an action or set of actions, and the second part representing the number of times that action is to be repeated. The entire set of genes in the chromosome can also be executed repetitively, in which case the whole chromosome becomes a cycle.

Parker used CGAs to evolve single-loop programs for robotic control of individual leg cycles [10], gait cycles for hexapod robots [11], and area coverage patterns [12]. The CGA was well suited for these problems because the solutions are cyclic in nature and required a single loop for control. Problems that require dynamic changes in behavior depending on sensor input call for multi-loop control programs for which a system of conditional branching must be implemented in the CGA.

3. THE SEARCH SIMULATION

The searching agent modeled in simulation is the ServoBot, which was developed for legged robot and colony experimentation. It is a small inexpensive hexapod robot constructed from masonite (a hard pressed wood), with motorized servos for actuators and a BASIC Stamp II for coordinating the motion of all six legs.

The simulation area is a square with each side of length 500 units. The agent's position in the simulation area is described by its X and Y coordinates, as well as a number between 0 and 359 that represents its heading. Since we are evolving searching behavior the target remains stationary, the target's position can be represented using only X and Y coordinates. At the start of each simulated evaluation, the agent and the target are placed randomly in the square area. The evaluation ends as soon as the agent locates the target or it reaches the predefined limit of 400 steps.

The agent can execute one of thirty-two possible gait cycles. For more information see [12]. A gait cycle is 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. The agent can remain motionless for the time it takes to complete a full gait cycle and represents the robot's 32nd gait. Unless the robot stands still it faces a trade-off between large displacement and small rotation, or vice versa.

The agent is equipped with two types of sensors, one that would enable it to detect the presence of a wall within its range, and the other is capable of detecting the target when it is within range. The two simulated sensors for wall detection are modeled after sonar sensors. The other two sensors on the robot are simulated to detect a light source that is the target. The robot has two sensors of each type or a total of four sensors. Since each sensor has two possible states – 0 (inactive) and 1 (active), there are sixteen possible combinations of sensor inputs. Although the robot has four sensors, the chromosome is designed with only four different states because the two sensors dedicated for light detection end the simulation if either or both are triggered. The activation distance for each sensor is 80 units and the range of its vision spans 45 degrees. The two sensors of each type are situated at the front of the robot with their ranges overlapping 10 degrees. Thus the robot has a total sight range of 80 degrees and can detect walls or the target at a maximum distance of 80 units away.

The agent and the target started the simulation from random positions for every evaluation. We assigned a fitness score for each individual based on its performance averaged over a fixed number of trials, with the random starting positions. To ensure that the score of each individual was representative of its fitness relative to the other individuals of the population, all individuals in the same generation started the simulation from the same randomly generated positions. Those starting positions were then reset every generation.

To evaluate the performance of each individual we used the maximum number of steps the agent was allowed to take and subtracted the number of steps it actually took before it detected the target. We then took the average of that number for all runs with different starting positions, and squared it. The resulting fitness function is shown in Eq. (1), where n was the number of starting positions for each generation. If the agent could not find the target once during a generation of training, it was assigned a nominal fitness.

$$\left(\frac{\sum_{i=1}^{n} (400 - steps)}{n} \right)^2 \tag{1}$$

4. CGA WITH CONDITIONAL BRANCHING

In order to produce an efficient search, we modify the standard CGA so that it evolves a multi-loop program that would switch from one loop to another depending on sensor inputs. The resulting algorithm we call *cyclic genetic algorithm with conditional branching*.

The chromosome shown in Figure 1 is a representation of a chromosome that can be used for CGAs with conditional branching. Each segment represents a control loop, a cycle that the robot repeats as long as the sensors' inputs stay the same. Each segment is linked to any number of other segments. This means that one segment must be dedicated to every possible combination of sensor inputs.

Figure 2 shows how the chromosome is represented and shows the breakdown of one of these segments. The genes consist of a fixed number of pairs of integers. The first integer of the gene determines which action is to be taken and the other dictates the number of repetitions of that action. After performing one action the specified number of repetitions, the robot checks the state of the sensors. If the sensor states are the same as the last time they were checked, the robot goes on with the next gene in the same segment. If it reaches the last gene in the segment, it continues to cycle by beginning again with the execution of the first gene in the segment. If the sensor inputs are different than the last ones, the robot halts the cycle and jumps to the first gene of the segment that corresponds to the new sensor inputs.

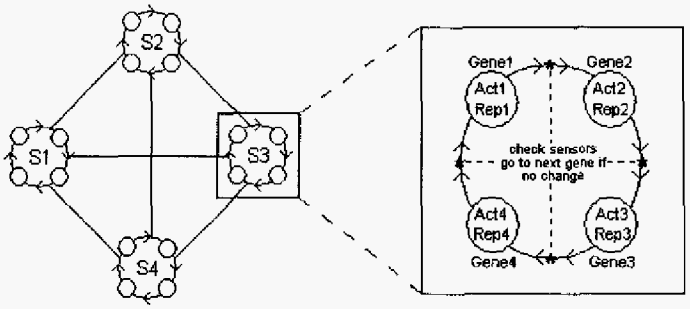

Figure 1. CGA with Conditional Branching Chromosome and a detailed view of one of its segments. The example chromosome shown contains 5 segments. Each segment is a loop.

Our design requires one chromosome segment for each possible combination of sensor inputs. Since the simulation of the search ended as soon as the agent found the target, only the two wall sensors were implemented in the chromosome. This gives four possible sensor input combinations and requires the chromosome to have four segments, each with four genes. A gene is 8 bits long; 5 bits for the denoted gait (32 possible actions), and 3 bits for the number of repetitions of that gait cycle. Thus the robot could repeat the same action a maximum of seven times before it checked its sensor inputs. Each segment had a resulting size of 32 bits, and the entire chromosome had 128 bits. Figure 6 gives a schematic of the chromosome.

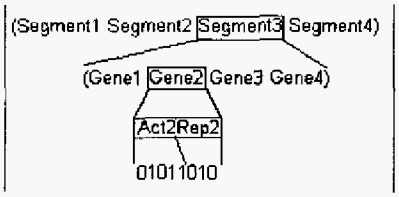

Figure 2. The chromosome.

5. RESULTS

Initial experiments were done using ten, thirty, and fifty trials per individual per generation. It was determined that the best trade-off between computation time and objective representation of the overall performance of each individual was obtained at ten trials per generation. To conduct a test of this method we randomly generated five populations, each consisting of 64 individuals and evolved them for 2048 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 generations 0, 32, 64, 128, 256, 512, 768, 1024, 1408, and 2048 were saved during training. Since the individuals from the different generations started the simulation from different starting positions, the direct comparison of their fitness scores would not be representative of their relative performance. Therefore we used a test program that generated 100 random starting positions, and had all individuals from all saved generations run the simulation from them.

Figure 3 summarizes the results by comparing the average number of steps taken to find the target by the best individual of each generation of training. The fittest randomly generated solution (generation 0)

needed 201 steps on average, but by the 2048th generation the best solution needed only 154.2. The oscillations of the curves on the graph demonstrate how the outcome of the evolution is affected by the starting positions of the set of trials. Overall, the graphs show a steady learning process through time, and a significant improvement of performance.

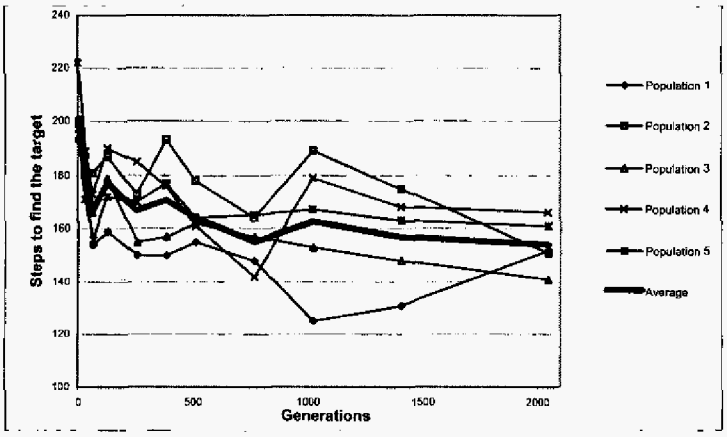

Figure 3. The results of the evolution of five populations with 64 individuals for 2048 generations. Results were recorded at generation 1, 32, 64, 128, 256, 384, 512, 1024, 1408, and 2048. The best individual at each generation is shown. The average of the best from each of the five populations is shown in bold.

When reviewing the results of these tests, one needs to take into consideration that the robot could always randomly run into the target from the very beginning, regardless of the level of sophistication of its searching behavior. In addition, even a perfect solution would necessitate taking a large number of steps if the robot and the target started the simulation far apart, or if the robot simply did not face in the direction of the target from the beginning.

Visual observations of the performance of best individual from later generations showed that they had successfully learned to avoid the walls of the square area. In the case of the best individual from generation 2048 from Population 3, the agent makes a sharp turn left upon wall detection. When the sensors detect no walls the agent swings out in a wide circle turning to the right by using a zigzag motion that enables it to maximize the area covered by its sensors. Figure 4 shows two of its resulting search patterns. It is evident that the robot is able to cover most of the area without doubling back into an area it has already covered.

6. CONCLUSIONS

Our results demonstrate that we were successful in accommodation sensors in control programs evolved by a CGA. The implementation of conditional branching allowed us to evolve a multi-loop program that would jump from one loop to another depending on sensor inputs. In the future we plan to apply this design to problems with greater complexity. Specifically, we plan on extending the current simulation into a predator-prey scenario by replacing the target with another simulated agent that will attempt to elude the predator.

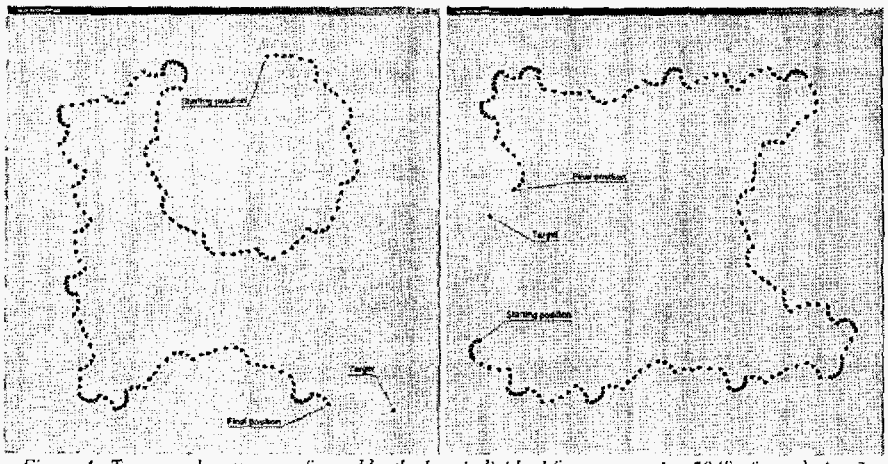

Figure 4. Two search pattern performed by the best individual from generation 2048 of population 3. The agent's position and direction after each step is shown.

7. REFERENCES

[1] J. Koza, *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

[2] 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.

[3] 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.

[4] 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.

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

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

[7] 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.

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

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

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

[11] 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.

[12] 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.