Reinforcement learning of an obstacle-avoiding and light-seeking neural-network robot controller using HyperNEAT and ant-colony optimization

Alexander Cordero, Bo Cao, David Johnson, and Irene Beckman

Abstract— **In this paper, we compare the performance of the Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) and continuous ant-colony optimization (ACO) algorithms to evolve recurrent neural-network robot controllers with complex behavior. We attempted to train robots in a simulated environment to move around a U-shaped wall towards a light source. We were not able to evolve a neural-network that was able to achieve the goal using the standard implementation of these algorithms. Some success was achieved with HyperNEAT using a modification to the initial conditions of that algorithm. Our results indicate that successful application of these algorithms may require longer training times.**

I. INTRODUCTION

HyperNEAT is a neuroevolutionary algorithm that has recently been used in tasks such as evolving coordinated gaits in quadruped robots and playing Atari games [2], [6]. Ant colony optimization is an algorithm inspired by the foraging tactics of ants and is known for its application to NP-Hard Problems [3]. In this paper, we compare HyperNEAT to ACO on the task of evolving an obstacle-avoiding and light-seeking neural-network robot controller.

Programming a robot to use light sensors to move towards a light source while concurrently using a laser collision detector to avoid obstacles is a difficult task. In this paper, we examine if training a neural-network with HyperNEAT or ACO can be used to generate a robot controller that can navigate around a U-shaped obstacle while moving towards a light source.

II. HYPOTHESIS

Given that HyperNEAT was specifically designed to generate neural networks with complex behavior, we hypothesized that it would be able to evolve a robot controller capable of avoiding a U-shaped obstacle while moving towards a light source. Furthermore, we expected that it would be able to evolve such a controller faster than ACO.

III. SIMULATION ENVIRONMENT AND ROBOT CONSTRUCTION

A. Simulation Environment

Gazebo was used as the simulation environment to train the neural-network robot controller [8]. The experiment was implemented in a virtual word because constructing it in the real world would have been time-consuming and expensive. Although a simulated environment cannot fully represent all of the features of the physical world, Gazebo is a widely used simulation environment for robotics capable of realistic and accurate simulations.

To train the robot controller to avoid obstacles while moving towards a light source, the simulation contained a destination goal placed below the light source and a U-shape obstacle in latter half of the field. These were implemented using Robocup 3D Goal and 18 walls models, respectively. The Robocup 3D Field model was used as the base plane with four outer walls to confine the movements of the robot. For each run of the simulation, the robot started at the center of the Robocup 3D Field with the target of reaching the Robocup 3D Goal despite the U-shaped obstacle obstructing the direct path.

Fig. 1 shows the simulation environment. The initial position of the robot was the center of the field where the coordinates were x = 0.372609m, y = -0.100070 m. The goal was set at the position of x = 14.958200 m and y = -0.113975 m. The light source was directly above the goal at x = 15.010900 m, y = -0.016872 m, z = 7.0 m. Between the initial position of the robot and light source was a U-shape obstacle, consisting of 14 walls. The center of this obstacle was x = 8.999620 m, y = 0 m.

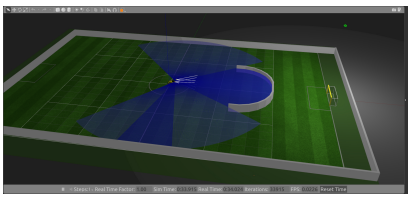

Fig. 1. Simulated world with a Robocup 3D Field, U-shaped obstacle, outer walls, light source and a Robocup 3D Goal.

B. Robot Construction

The Husky robot model was used in this experiment for its simplicity of control. Two cameras were attached to the top front of the robot on the left and right sides as light sensors. They were positioned in this manner so that the robot controller would know the direction of the goal. Additionally, a laser scanner was attached to the top front of the robot (x = 0.3m, y = 0.0m and z = 0.25m relative to husky) to detect the distance of obstacles in a 300° radius around the robot.

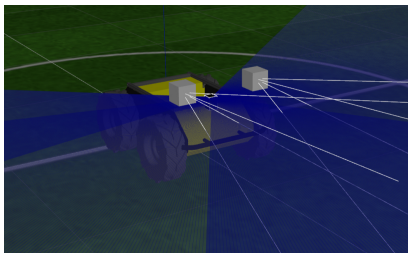

Fig. 2. A husky robot with 2 cameras and a laser scanner

Compared to the experiment in [4], in which the robot's objective was to maximize environment exploration while avoiding obstacles, our design provides a more defined task for the robot to learn. In the experiment in [4], the robot did not have a specific location in the environment to find. However, in our simulation, the robot must use the light sensors to move toward a light source while avoiding obstacles using data from the laser scanner.

IV. CONTROL ALGORITHMS

A. HyperNEAT

NeuroEvolution of Augmenting Topologies (NEAT) is a neuroevolutionary algorithm that uses genetic methodologies to evolve both the weights and topology of a neural-network [11]. It uses a fitness criterion to search for neural-networks that have behavior which will solve the task at hand. NEAT directly encodes the structure of a neural-network in a genome. The nodes of a neural-network are encoded in node genes that indicate whether or not the node is an input, hidden, or output node. Connection genes indicate which nodes are connected and the strength of the weight between them. The initial population of genomes starts with minimal structure, input nodes are directly connected to output nodes. Structural mutations that add nodes or connections to a genome increase the complexity of the population as it evolves in addition to mutations that change the weights of connection genes.

Connection genes also have an innovation number which is an historical marker indicating when that particular gene appeared in the course of evolution. The innovation number is universal across the population of genomes and the genes of offspring have the same innovation numbers as their parents. It allows two parent genomes to align for cross-over without needing a detailed topological analysis of their structure. During cross-over, connection genes with the same innovation number in both parents are passed to the offspring. Genes not common to both parents are inherited from the parent with the highest fitness.

NEAT subdivides the population of genomes into species to protect novel mutations that may eventually evolve into a solution. A distance measure based on the similarity of genes is used as the criterion for speciation.Genomes within a species compete amongst themselves instead of the wider population.

Hypercube-based NeuroEvolution of Augmenting Topologies (HyperNEAT) is an extension to NEAT that uses an indirect encoding of the neural network structure [10]. Instead of encoding neural networks, the genomes in HyperNEAT encode connective compositional pattern-producing networks (CPPNs) which are then used to generate the actual neural network. Connective CPPNs are similar to neural networks in that they consist of networks of functions like the logistic regression. As input, however, CPPN's take two nodes in the substrate and output the weight of the connection between them. **Fig. 3** is a visualization of a connective CPPN and **Fig. 4** is the corresponding neural network generated by that CPPN.

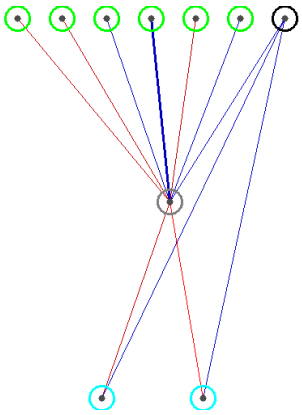

Fig. 3. Visualization of CPPN

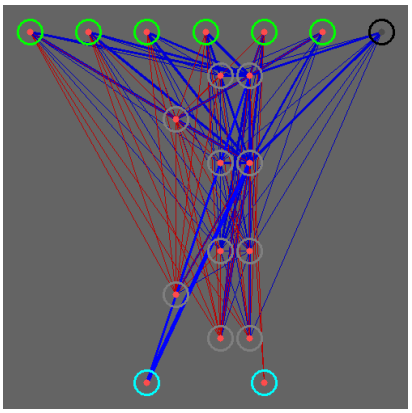

Fig. 4. Visualization of Neural Network

B. Ant Colony Optimization

A generalized ACO can be divided into 4 distinct stages: initialization of parameters and pheromone trails; constructing ant solutions; updating pheromone values; and checking the exit condition [3]. Based on the work by Hsu and Juan in evolutionary wall-following control using continuous ACO, the following stage was added: generating new nodes in the solution space [7]. An outline of these steps can be seen in **Fig. 5**.

1) Topology: To better compare the results of ACO to HyperNEAT, it was decided that the final solution should be a set of neural network weights and input parameters. Based

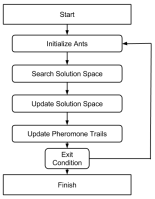

Fig. 5. Continuous ACO Algorithm Outline

off of the network suggested by Floreano and Mondada, the final structure, diagrammed in **Fig. 6**, was developed [5]. This network takes in environmental sensor data as well as current state information from the robot itself. The robot state inputs, inspired by the fitness function put forth in the 1996 paper, are: αV, $\beta \Delta v$, and $\gamma(1-i)$. V is the average velocity for the right and left wheels, Δv is the absolute value of the difference between the two wheels' velocity, and i is the maximum sensor input from the proximity laser [5]. The weights assigned to these extra inputs, α, β, and γ, are determined by ACO. The resulting final topology produced is 4 layers of 10 nodes that are fully connected between each layer. The first layer is a series of recurrent neural networks; this is where the weights are trained. The second, third, and fourth layer control the values of α, β, and γ respectively. **Fig. 7** shows the final search space.

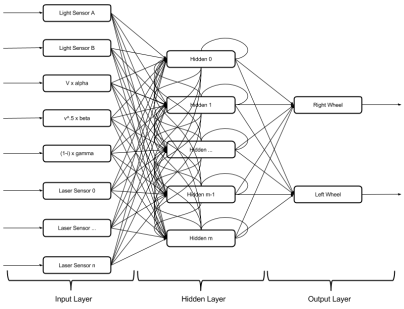

Fig. 6. ACO Neural-Network Structure

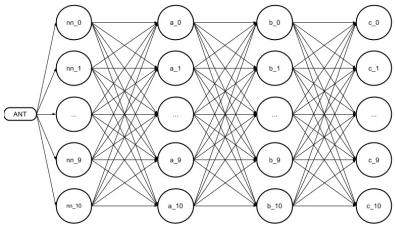

Fig. 7. ACO Topology

2) Initialization and Exit Condition: Layers controlling α,β, and γ, were assigned numbers from 0 to .9 in .1 value increments. The neural-network layer was set using 3 different hidden layers sizes as the only structural difference between the nodes. However, between runs the number of laser sensor inputs used varied between 1 (the maximum value) and 640 (the entire array). Depending on these environmental inputs included, the sizes and node counts were: 8(4 nodes), 10(3 nodes) and 15(3 nodes) or 744(4 nodes), 844(3 nodes) and 944(3 nodes). The weights within the 10 neural-networks were assigned randomly by pybrain, a library used in this project [9]. Between the 4 layers, the pheromone levels were equally distributed and summed to 1. A colony size between 10 and 30 ants was chosen depending on the run. Similarly, the exit condition was set to trigger after 30 or 100 iterations.

3) Constructing Ant Solution: Once an ant sets out on its path, the solution selected at each turn was comprised of a neural network, an α value, a β value, and a γ value (See **Fig. 7**). Selecting a path j in layer i is given below:

$$\sum_{k=0}^{j-1} \tau_{i,k} < x_j < \sum_{k=j+1}^{m} \tau_{i,k} \quad Given: x \in \vec{U}[0,1]$$

Where τ is the pheromone level,i is the current layer, j is the current path, m is the size of layer i, and $\vec{U}[0,1]$ is a set of random numbers uniformly distributed from [0,1]. This method required us to normalize within each layer after every pheromone update.

4) Updating Pheromone Trails: Once every ant traversed the graph, pheromone values were updated according to the equation below [3]:

$$\tau_{i,j} = (1-p)\tau_{i,j} + pF(x_{i,j})$$

Where $\tau_{i,j}$ is the path weight, $F(x_{i,j})$ is the fitness function output for ant x on path j in layer i, and p is the pheromone evaporation weight.p was set to .8 [1]. After all pheromone trails were updated, each layer was normalized for use in path selection during the next iteration.

5) Increasing Exploitation and Exploration: After each iteration, the α, β, and γ node values were updated to increase exploitation and exploration [7]. The lowest two performing nodes of each layer were removed. One node was replaced with the average value of the top two performers. It allowed for more precision than the initial assigned values gave and attempted to narrow in on an optimum input weight. The remaining node was replace with a randomly generated value. The paths to the newly reassigned nodes were updated to the lowest current pheromone trail value and the entire layer was normalized once again.

V. RESULTS

A. HyperNEAT Results

The MultiNEAT python library implementation of HyperNEAT was used to evolve the neural-network controller. The fitness criterion of

$$f = 40 - D$$

where D was the distance between the robot and the goal in meters after 10 seconds was used because the field has a length of 30 meters. Four runs were done to compare its performance to ACO. In the first two runs, the population of candidate solutions was set to 20 and evolved for 100 generations. In the third and fourth runs, the population was set to 100 and evolved for 20 generations. In all cases,

Ants	Iterations	Input Size	Avg Final Fitness
10	30	5	21.84
10	30	645	24.04
20	100	645	29.38
30	100	645	26.29

Fig. 8. ACO Results

HyperNEAT was unable to evolve a neural network that could get around the obstacle. The neural networks with the highest fitness or that got closest to the goal simply reached the center of U-shaped barrier.

B. ACO Results

To allow for better comparison, ACO used the same fitness function described above. Four complete runs were executed and the results of these can be seen in **Fig. 8**. It never successfully reached the goal.

VI. ANALYSIS

A. HyperNEAT Analysis

There are several possible reasons why HyperNEAT was unable to find a solution. The simplest is that the population of candidate solutions was too low at 20 or 100 as was the number of generations. A population size of 200 - 300 evolved through 1000 generations may have found a solution. However, given that a run of the algorithm with our parameters took 6.5 hours, increasing the number of generations or population size would require runs taking several days.

Another possibility is that timeout of 10 seconds for the robot to reach the goal was too short. To examine this possibility, a run with a population of 100 candidate solution was evolved for 10 generations with a timeout of 30 seconds was done. This run resulted in a robot that was able to get around the obstacle but only because it continually turned right so that it crashed first into the right side wall, then the left before curving around to the goal. This was the best run produced both algorithms.

The simplicity of the fitness criterion may have also played a role. The fitness criterion simply evaluated how close the robot got to the light source. It is possible that a fitness criterion that also took into account the distance to the obstacle as well as the goal would have yielded better results.

B. ACO Analysis

Similarly, there are multiple reasons why ACO was unable to perform the desired task. The first is that ACO often iterates on the order of 100 as opposed to 10. As such, it is possible that given enough time the current implementation would have found the goal. Having the ability to run robots concurrently for 400 or 500 iterations might have produced better results.

Furthermore, since the previous velocity was used as an input, the robot appeared to speed up with every iteration. By the later rounds, it frequently flipped over, rendering it unable to move, and minimizing the effect that trial had on the final solution. Developing a way to pre-process the velocity into reasonable bounds might help keep the robot functioning through all rounds.

Altering the input could have positively influenced the performance. When only the maximum sensor value was included in the controller neural network, the robot was too readily attracted to the wall. It never fully explored the search space and shortly found itself returning to middle of the U. However, when all sensor inputs were allowed, the robot never developed a single clear path. The laser inputs outnumber αV, $\beta \Delta v$, and $\gamma(1 - i)$ to such an extent that we believe they negated any impact these additional values could have had. Reducing the input layer by 30 or 50 percent might have allowed the robots to come to a final solution.

VII. CONCLUSIONS

The results of this paper indicate that naive applications of HyperNEAT and ACO may not yield the best outcomes, with HyperNEAT preforming slightly better. HyperNEAT may require many generations and large population sizes to get to usable solutions which can take a long time in robot simulations. ACO would also benefit from a longer training time and exploring alternate inputs sizes for the neural-network controller.

REFERENCES

[1] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Number 1. Oxford university press, 1999.

[2] Jeff Clune, Benjamin E Beckmann, Charles Ofria, and Robert T Pennock. Evolving coordinated quadruped gaits with the hyperneat generative encoding. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2764–2771. IEEE, 2009.

[3] Marco Dorigo, M Birattari, and T Stutzle. Ant colony optimization: artificial ant as a computational intelligence technique. university libre de bruxelles. Technical report, IRIDIA Technical report Series, Belgium, 2006.

[4] Dario Floreano and Francesco Mondada. Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3: From Animals to Animats 3*, SAB94, pages 421–430, Cambridge, MA, USA, 1994. MIT Press.

[5] Dario Floreano and Francesco Mondada. Evolution of homing navigation in a real mobile robot. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(3):396–407, 1996.

[6] Matthew Hausknecht, Piyush Khandelwal, Risto Miikkulainen, and Peter Stone. Hyperneat-ggp: A hyperneat-based atari general game player. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 217–224. ACM, 2012.

[7] Chia-Hung Hsu and Chia-Feng Juang. Evolutionary robot wall-following control using type-2 fuzzy controller with species-de-activated continuous aco. *Fuzzy Systems, IEEE Transactions on*, 21(1):100–112, 2013.

[8] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[9] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 11:743–746, 2010.

[10] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[11] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.