A. Nelson, E. Grant, G. Lee, **Using Genetic Algorithms to Capture Behavioral Traits Exhibited by Knowledge Based Robot Agents**, Proceedings of the ISCA 15th International Conference: Computer Applications in Industry and Engineering (CAINE-2002), pp. 92-97, San Diego Ca, Nov. 7-9, 2002

Using Genetic Algorithms to Capture Behavioral Traits Exhibited by Knowledge Based Robot Agents

A. L. Nelson and E. Grant
Center for Robotics and Intelligent Machines
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27695-7911

Gordon Lee
Department of Electrical and Computer Engineering
San Diego State University
5500 Campanile Drive
San Diego, CA 92182

***Abstract*—One of the fundamental issues in the field of Evolutionary Robotics (ER) is that of selection and formulation of an appropriate performance training metric. In recent years, proof-of-concept studies have shown that simple behavioral robotics problems, such as homing and foraging, are amenable to ER methods. However, the question of scalability remains unresolved. Several researchers have shown that straightforward ER methods fail to produce viable results on more complex problems if the problem is not partitioned or preprocessed in some fashion before applying ER methods. In many cases, the knowledge required to preprocess the problem is equivalent to that which would be needed to formulate a purely rule knowledge-based controller, hence, in such cases, ER methods provide no real benefit. In this work, we present research results of an investigation into the feasibility of using observed behavior in an environment to train artificial neural network-based robotic controllers to function in that same environment. Robot agents were allowed to navigate through a selection of artificial life simulation environments under the influence of knowledge-based controllers. At each time-step, the simulated robot sensor inputs and actuator outputs were recorded. The resulting input and output data were used to train artificial neural network based controllers for the different environments. The resulting neural network based controllers were then used to control robots in similar environments and were found to exhibit features of the original knowledge-based controllers.**

***Key Words*--Evolutionary Robotics, Behavioral Robotics, Evolutionary Neural Networks, Artificial Life**

I. INTRODUCTION

The main goal of behavioral robotics is to develop intelligent control for autonomous robotic systems. A fundamental issue is the form and expression of the behavior itself. Various frameworks for behavioral robotics control have been investigated over the years (see [1]). The majority of these approaches have been knowledge-based systems. Recently, the field of evolutionary robotics has received attention as a possible method to achieve complex behavior in robotic systems. See [2] and [3] for recent reviews.

Evolutionary robotics methods apply evolutionary computing techniques to develop robot control systems that produce a desired set of robot behaviors. In general, evolutionary computing methods require the use of a training fitness function or objective function. This fitness function is essential for evaluation of stochastic alterations made to potential solutions during the evolutionary search process. Selection of an appropriate fitness function can be quite difficult for nontrivial behavioral problems. Although simple robotic behaviors have been developed using ER methods [4-6], it has not yet been shown that ER methods can be used to develop sophisticated behavioral robotic control systems. Various researchers have investigated methods such as incremental evolution [7][8] and minimal simulation [9] to overcome the problems associated with the fitness function formulation. These methods offer improvements to the ER methods; however, it has not been shown that they will lead to the development of advanced behavioral control systems.

One potential method to overcome problems associated with the fitness function formulation is to use artificial life simulations, or data derived from such simulations to aid in the training or evaluation of evolutionary robotics control systems. Here, we address issues related to the acquisition of an observed behavior by methods that do not require explicit formulation of a problem specific fitness function. We present an example of extraction of behavioral features using an evolutionary neural network system from observed behavior of robot agents in simulated artificial life environments.

Initially, simple knowledge-based controllers were formulated to allow robot agents to wander around in simulated environments without running into obstacles or getting stuck on material in the environments. These controllers consisted of simple sets of rules that related sensor inputs to wheel motor outputs, so that the motion of each simulated robot agent was a direct function of current sensor inputs. Simulations in which robot agents were allowed to move about the environments under the influence of these knowledge-based controllers were then performed. At each time step during the simulations, all robot sensor inputs and associated actuator outputs were recorded. The resulting data were then used to train an artificial neural network using supervised evolutionary methods similar to those described in [10]. The trained neural networks were used as the bases for robot agent controllers for use in the same simulated environments in which the training data were developed. The neural network base controllers were able to perform similar object avoidance and locomotion functions as the original knowledge-based controllers.

The method used in this work partially automates the specification of a domain specific training fitness function. The only explicit training fitness function used is that of minimization of the error in reproduction of the mapping of the observed sensor inputs to the desired actuator outputs, which have been recorded as a static data set. Any complexities of rule-environment interactions are included only implicitly in this static I/O mapping data set. This reduces the need for designers to characterize and preprocess behavior before training a robotic system to reproduce that behavior. Reducing preprocessing requirements would improve the efficiency of the design process by reducing the amount of effort required from human designers. In addition, methods similar to the one described in this work might be applied to very complex systems that are not be easily characterized by human designers.

Although the underlying behavior was generated by the application of a set of simple rules, the artificial neural network does not require these rules to be explicitly stated in order for it to extract the essence or features of the behavior and to reproduce it. The behavior might just as well have been generated by unknown rules or uncharacterized mechanisms. The evolutionary neural network was trained only with data obtained by observing the behavior of the robot agents.

II. SIMULATION ENVIRONMENT AND ROBOTIC AGENT DESCRIPTION

The simulation environments consist of *m* by *m* planar grids in which each grid element is either solid or space. Although the matter arrangement in each environment is discretized, the space itself is continuous and robot agents may exist at any real valued point within the range of the environment.

In each environment, a variable number, *n*, of robot agents are maintained. Each robot agent consists of a data structure that stores the robot's current position, orientation, sensor input readings, and actuator output values. In addition a controller structure is associated with each robot. In this work, all controllers were time independent mappings from the robot's sensor inputs to the robot's actuator outputs and can be written in functional form as follows:

$$\mathbf{A}_n = f_{cn}(\mathbf{S}_n) \qquad (1)$$

where $\mathbf{A}_n$ and $\mathbf{S}_n$ are the sets of robot actuator and sensor values of the *n*th robot agent, respectively, and f_{cn} is the controller mapping associated with the *n*th robot agent.

Each robot was simulated with two motor-wheel actuators, one on each of the right and left sides of the robot. Such a configuration allows a robot agent to turn in any direction or move along any diameter arc by varying the inputs to the wheel motors.

Range finding sensors were simulated so that each sensor is associated with a fixed orientation with respect to the robot's body frame. Each sensor returns a scalar value that corresponds to the linear distance between the sensor and the nearest solid element directly in line with the sensor's orientation. For the work presented here, each robot was given seven forward facing sensors, each oriented at an offset of 15° from one another. This produced a whisker like array of sensors centered along the central axis of each robot body frame. A schematized top view of a robot agent and range sensor array detecting matter in a simulated world is shown in Figure 1. The object on the left in Figure 1 is the robot agent, while the lines represent the magnitude and direction of current sensor readings. The small blocks in the figure represent simulated matter.

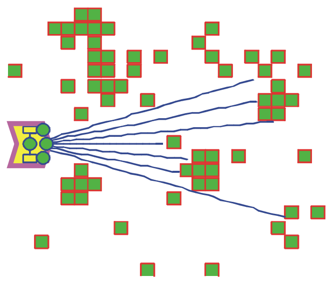

Figure 1. Simulated robot agent with sensor and wheel actuators in a matter-containing environment. The object on the left is the robot agent. The lines represent the magnitude and direction of current sensor readings. The small blocks represent simulated matter.

At each simulation time step, and for each robot, the following actions were preformed:

1. The robot sensor readings were updated.

2. The current sensor readings were fed into the associated controllers and actuator values were calculated and applied to the wheel motors.
3. The robot's next position was calculated as a function of its current position, orientation, the time step size, and the current wheel motor rates
4. If the robot's next position was found to lie within an open space grid element, the robots current position was updated to that next position, otherwise the robot's current position remained unchanged.

The position update function of step 3 is given symbolically by equation (2) below. $\mathbf{A}_n$, $\mathbf{R}_n$ and Δt are the *n*th robot's current actuator values, a set of physical parameters associated with the *n*th robot and the simulation time step size respectively.

$$Pos(n,t+1) = f_p(Pos(n,t), \mathbf{A}_n(t), \Delta t, \mathbf{R}_n) \qquad (2)$$

To ensure that robot agents could not jump over or tunnel through physical objects in the simulation environment, the maximum distance traveled by each robot at any time step was limited to be smaller then the dimensions of a matter grid element.

Three planar simulation environments were used for this work: an environment with linear walls that divide the space to form a maze-like structure, an environment filled with aggregates and clusters of matter, and an environment containing only space. These are referred to as Maze World, Aggregate World and Empty World, respectively and are shown in Figure 2.

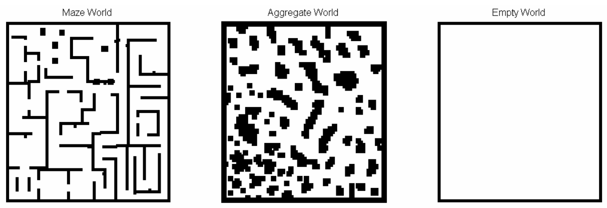

Figure 2. Planar simulation worlds

III. The Neural Network Training Genetic Algorithm

The neural network based controllers were trained using an evolutionary computation based procedure. Fully connected single hidden layer feedforward networks were used. Sigmoid activation functions were used in the hidden layer neurons while linear activation functions were used in the output neurons. Training neural networks of this type involves manipulating scalar weighting functions that operate on the inputs and outputs of the individual neurons in the network. The genetic algorithm used here operates directly on the neural network's set of weights; hence, the chromosome data structure is that of a set of real-valued scalar numbers.

The training data sets generated from the recorded sensor inputs and actuator outputs were formulated as sets of all input values, X, and associated outputs, Y, recorded over the duration of a simulation run. Note each element of X is itself a set of all sensor values recorded at a particular time step and, likewise, each element of Y is a set of all actuator values recorded at a particular time step.

A typical network structure of the type used in this work is shown in Figure 3. The robot agent's current sensor readings, x_1, x_2… x_i are scaled by the hidden layer neuron weights, wh_{in}, summed and fed into each hidden neuron sigmoid activation function, $y_{hid}(u_{hid})$. The summations of weighted inputs are given by equation (3)

$$u_{\text{hid,n}} = \sum_{1}^{i} (\mathrm{x_i} * \mathrm{wh_{in}}) \qquad (3)$$

Similarly, outputs from the hidden layer neurons are scaled by the output neuron weights, wo_{im}, summed, and feed into the output neurons. There can be an arbitrary number of hidden layer neurons; however, each output neuron produces a single actuator value, so the number of output neurons matches the number of actuators in each robot agent.

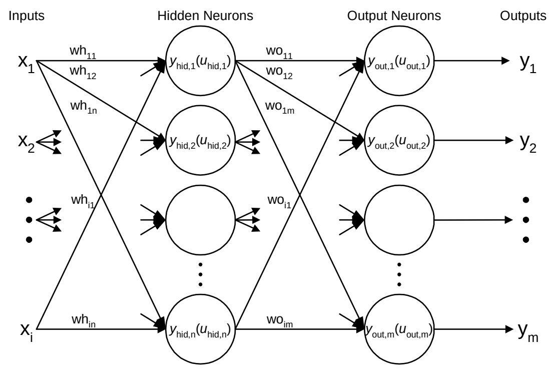

Figure 3. Generalized feed-forward single hidden layer ANN.

The genetic training algorithm used is a population of one, mutation based, greedy gradient decent algorithm similar to that described in [10]. Instead of using bit strings to represent candidate solutions, a set of real numbers representing the weights of the neural networks were used. Using the notation of Figure 3, a chromosome structure is given by:

$$\mathbf{c} = [\mathrm{wh_{11}}, ..., \mathrm{wh_{1n}}, ..., \mathrm{wh_{in}}, \mathrm{wo_{11}}, ..., \mathrm{wo_{1im}}, ..., \mathrm{wo_{im}}] \qquad (4)$$

At the beginning of the training, all weights in each robot agent neural network are initialized with small random numbers. At each iteration of the training algorithm, perturbations (mutations) are made randomly to one or more of the elements of **c** to make a new chromosome, **c**'. The neural net is then tested with the weights specified by **c**' and its performance is compared to that obtained with the weights specified by **c**. If the performance is improved, the mutations in **c**' are kept; otherwise the original **c** is retained and the process is repeated. The measure of quality of performance used was that of mean squared error, *E*, as:

$$E_j = \frac{\sum_{m=1}^{M} (y_m(\mathbf{x}_j) - y_{tm}(\mathbf{x}_j))^2}{M} \quad (5)$$

Note $y_m(\mathbf{x}_j)$ is the output produced by the *m*th output neuron when it is presented with the *j*th set of training inputs, $\mathbf{x}_j$, and *M* is the total number of outputs. Further, y_m is the actual output of the *m*th output neuron, while y_{tm} is the desired, or training output of the *m*th output neuron associated with the *j*th set of training inputs.

During training, mutation occurs after presentation of each example in the training set rather that after presentation of the full training set. This allows the solution (weight set chromosome) to step out of a local minimum with respect to one training example with the occurrence of mutation that reduces error for another training example.

A main feature of the algorithm is that the probability of mutation of each number in the chromosome at each generation is dependant on the current quality of the solution (here, a solution refers to a neural net and associated weight set as specified by the current chromosome **c)**. Early in training when solutions are very poor, the mutation probability is high. As training continues, and the solution becomes more refined, the mutation probability decrease. The following formula was used to calculate the probability of mutation for each element of **c**:

$$\text{MutateProb} = \text{BaseMutateProb} + HB * \frac{E_{current}}{E_{base}} \quad (6)$$

$E_{current}$ is the present training error, *E*, as calculated by equation (5). In this work, E_{base} was set to be the training error calculated at the first iteration of training. *HB* is a scaling factor used to regulate the degree to which the mutation probability was affected by training error. Values used for HB in this work were generally close to unity. BaseMutateProb, the minimum mutation probability, was set to be one mutation out of all the weights in **c**, on average. The effect of (6) is that early in the training, while the error is high, the mutation probability is near one so that most elements of **c** are mutated at each training iteration. As *E* decreases, fewer and fewer mutations occur during each training iteration. When *E* is very small, the BaseMutateProb term dominates and only 1 mutation occurs at each training iteration, on average.

Making the mutation probability dynamic and related to the quality of the current solution is thought to allow initial, poorer solutions to move more quickly through the solution space. As the solution improves, smaller steps in the solution space are taken. A low mutation rate is desirable near the end of training as the solution is fine-tuned.

The magnitude of mutation was random, and normally distributed. The center of the distribution was kept constant for the duration of a particular training. For the work presented here, mutations magnitudes were on average between 1% and 5% of the weight magnitudes at the start of training.

IV. Experimental Procedure

Three robot agent controllers were formulated for use in the simulated environments. These were: a simple knowledge-based controller, a trained neural network based controller, and a random controller.

Robot agents were initially controlled with a knowledge-based controller that produced wheel speed values as a function of range-finding sensor input values. Two simple rules were found to be sufficient to keep robot agents from getting stuck in most environments These are formulated as follows:

> Rule 1: Each wheel motor speed is made proportional to the sum of the range finding sensor readings on the opposite side of the robot.
>
> Rule 2: If the total sum of all the range finding sensor readings is less than some threshold, then the right hand side wheel motor is reversed.

The effect of Rule 1 is that of causing a robot to veer away from objects or to remain in the middle of a corridor. Rule 2 allows the robots to escape from corners.

The method used to extract behavioral traits observed in the robots agents operating with the knowledge-based controllers was as follows. A simulation world was arbitrarily constructed. Robot agents were initialized to random positions within the simulated world. A simulation was preformed using the simple rule based control. During the simulation, all robot agent sensor readings and associated motor output rates were recorded. These data were recorded as real-valued input and output values. These were in turn used to train an artificial neural network using the supervised evolutionary training algorithm described in the previous section.

For the work presented in this paper, the training set for the neural controller was derived from 50 time steps in Maze World.

To apply the neural controllers in simulation, the robot sensor readings were fed into the trained neural network and the resulting network output values were applied to the wheel motors.

For purposes of experimental control, a third 'random' controller was formulated. The random controller produced random wheel motor rates that had no relationship the sensor inputs. The wheel motor rates were normally distributed values around the mean of the robot agent speeds when operating under the knowledge-based controller in the simulation world used to derive the data used to train the neural net controller.

V. Results

Each of the three controllers were used to control robot agents in each of the three simulation environments. The results of these simulations are presented in this section.

In each simulation environment, three simulations were preformed, using ten robot agents in every case. The positions and orientations of the robots were initialized randomly for each environment, but were kept the same for all the controllers used in a particular environment so that the results could be compared. The robot agents do not interact; thus, this is equivalent to ten repetitions of a simulation with a single agent.

The Empty World and the random controller were included as comparative experimental controls.

During each simulation, the velocity, position, and orientation of each of the robot agents were recorded at each time point. The metric used for comparison of performance of the controllers was total distance traveled during a simulation. For a particular environment, robot agents using each of the three controllers were simulated for an equal amount of time. Simulation times were 100 time steps for Empty World and 1000 time steps for Maze World and Aggregate World.

Figure 4 presents data in bar graph form comparing the mean and standard deviation of distance travel by the robot agents during each simulation.

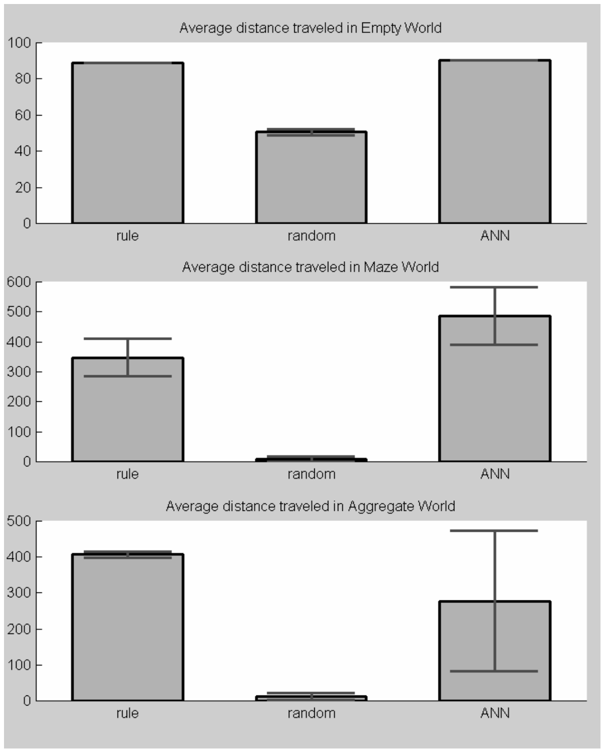

Figure 4. Mean distance traveled by robot agents using the rule-bases controller, the random controller and the neural network based controller, in each of the three simulation environments

VI. Discussion

The results shown in the first panel of Figure 4 indicate that all three of the controllers produce motion of the robot agents in Empty World. Since there is no matter in this environment, the robots can travel without the need to avoid obstacles. The speed of the random controller was set to have a mean value equal to that of the knowledge-based controller operating in Maze World. With the knowledge-based controller, wheel motor speed is proportional to range finder sensor input values; hence robots being controlled by this controller move much more quickly in Empty World than they do in either of the other environments because there is no matter to detect.

In Maze World and Aggregate World, obstacle avoiding behavior is much more important. The random controller causes the robot agents to become stuck or ensnared very quickly, resulting in very short net travel distances. Both the knowledge-based and the neural controllers, produce successful obstacle avoidance, resulting in longer total travel distances. These results are shown in the second and third panels of Figure 4.

The second panel of Figure 4 shows that the neural network-based controller performs as well as the knowledge-based controller in Maze World, although the variability in individual robot agent performance is greater, as reflected in the greater standard deviation of distance traveled. The training set for the neural controller was derived from 50 time steps in Maze World. The simulations above were performed for a period of 1000 time steps. This indicates that, at least in Maze World, the neural controller was able to generalize its performance to many situations not seen by the training set data. In fact, robots using the neural controller were able to operate in Maze World indefinitely without getting stuck on walls or in corners.

In Aggregate World, the neural controller did not perform as well as the original rule based controller, but it did significantly outperform the random controller. It is likely that the data set used to train the neural controller did not reflect some aspects of the rule base when applied to Aggregate World, since the training set for the neural controller was derived in Maze World.

VII. Future Research

The work presented here is of a preliminary nature. Tests have been conducted using the evolved neural-controllers in a real mobile robot platform. This work is being reported in another paper that focuses on the development of a mobile robot platform consisting of a colony of eight autonomous robots. This platform will be used to generate training data sets and to implement additional derived controllers similar to those described in this work. One eventual goal is to provide a method of extracting features of complex behavior without the need to fully understand or decompose that behavior. In this work, two simple rules were used to produce interactions between simulated robot agents and various environments. The expression of the rules, that is, the observed behavior, may be considerably more complicated than the rules themselves. We used a neural network-based system to back-extract behavioral features from the observed behavior of robot agents operating under the influence of these simple rules. In simple animals, such as insects, it is often not clear what rules are being followed in the expression of a particular behavior. Back-extraction of the observed behavior may be very useful in the field of behavioral robotic as a method to develop robotic control systems that produce behaviors not explicitly understood by human designers.

The neural network training data set was derived from the observed inputs and outputs of the robot agents operating under the influence of a simple rule set. This need not be the case, however. The robot agents could have been moved by an outside observer or made to follow a predefined path through a particular environment; the resulting data set could still be used as a training set. It is possible that this could be used as a method to extract behavior with out the need to decompose it into a rule set.

VIII. References

[1] R. C. Arkin, BEHAVIOR-BASED ROBOTICS, The MIT Press, Cambridge, Ma, 1998.

[2] S. Nolfi, Evolutionary Robotics: Exploiting the Full Power of Self-Organization, Connection Science, Vol. 10, pp. 167-183, 1998.

[3] D. Floreano, J. Urzelai, (2000) Evolutionary Robotics: The Next Generation. In T. Gomi (ed.), Evolutionary Robotics III, Ontario (Canada): AAI Books, 231-266.

[4] F. Gruau, Automatic definition of modular neural networks. Adaptive Behavior, 2, pp.151-183, 1995.

[5] H. Lund, J Hallman, Evolving Sufficient Robot Controllers, Proceedings of the IEEE International Conference on Evolutionary Computation, pp. 495-499, 1997.

[6] Watson, R. A., Ficici, S. G., and Pollack, J. B. (1999). Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In Angeline, P., Michalewicz, Z., Schoenauer, M., Yao, X., and Zalzala, A., editors, 1999 Congress on Evolutionary Computation, pages 335--342. IEEE Press, 1999.

[7] F. Gomez, R. Miikkulainen, Incremental Evolution of Complex General Behavior, Adaptive Behavior, Vol. 5, pp. 317-342, 1997

[8] D. Filliat, J. Kodjabachian J. Meyer, Incremental evolution of neural controllers for navigation in a 6 legged robot, In Sugisaka and Tanaka, editors, Proc. of the Fourth International Symposium on Artificial Life and Robotics. Oita Univ. Press, 1999

[9] N. Jakobi, Running Across the Reality Gap: Octopod Locomotion Evolved in a Minimal Simulation, Proceedings of the First European Workshop on Evolutionary Robotics: EvoRobot'98, 1998

[10] D. Levi, HereBoy: a fast evolutionary algorithm, The Second NASA/DoD Workshop on Evolvable Hardware 2000 Proceedings, pp. 17 -24, 2000.

Corresponding Author E-mail:
Andrew Nelson
alnelson@ieee.org
http://www.nelsonrobotics.org