Automatic Neural Robot Controller Design using Evolutionary Acquisition of Neural Topologies

Yohannes Kassahun and Gerald Sommer

Christian Albrechts University, Institute of Computer Science and Applied Mathematics, Department of Cognitive Systems, Olshausenstr. 40, D-24098, Kiel, Germany

Abstract. In this paper we present an automatic design of neural controllers for robots using a method called Evolutionary Acquisition of Neural Topologies (EANT). The method evolves both the structure and weights of neural networks. It starts with networks of minimal structures determined by the domain expert and increases their complexity along the evolution path. It introduces an efficient and compact genetic encoding of neural networks onto a linear genome that enables one to evaluate the network without decoding it. The method uses a meta-level evolutionary process where new structures are explored at larger time-scale and existing structures are exploited at smaller time-scale. We demonstrate the method by designing a neural controller for a real robot which should be able to move continously in a given environment cluttered with obstacles. We first give an introduction to the evolutionary method and then describe the experiments and results obtained.

1 Evolutionary Acquisition of Neural Topologies

Evolutionary Acquisition of Neural Topologies (ENAT) [6,7] is an evolutionary reinforcement learning system that is suitable for learning and adaptation to the environment through interaction. It combines meaningfully the principles of neural networks, reinforcement learning and evolutionary methods.

The method introduces a novel genetic encoding that uses a linear genome of genes (nodes) that can take different forms. The forms that can be taken by a gene can either be a neuron, or an input to the neural network, or a jumper connecting two neurons. The jumper genes are introduced by the structural mutation along the evolution path. They encode either forward or recurrent connections.

Figure 1 shows an example of encoding a neural network using a linear genome. As can be seen in the figure, a linear genome can be interpreted as a tree based program if one considers all the inputs to the network and all jumper connections as terminals.

The linear genome has some interesting properties that makes it useful for evolution of neural controllers. It encodes the topology of the neural controller

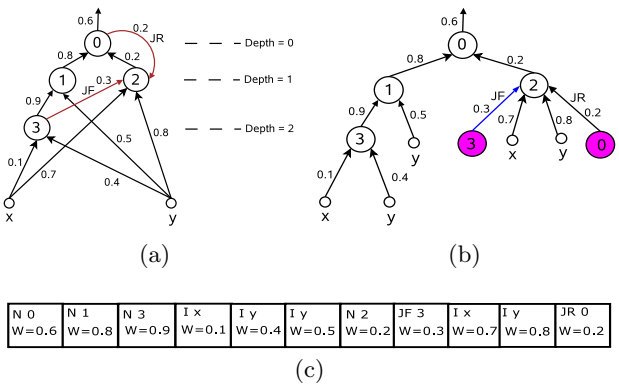


Fig. 1. An example of encoding a neural network using a linear genome. (a) The neural network to be encoded. It has one forward and one recurrent jumper connection. (b) The neural network interpreted as a tree structure, where the jumper connections are considered as terminals. (c) The linear genome encoding the neural network shown in (a). In the linear genome, `N` stands for a neuron, `I` for an input to the neural network, `JF` for a forward jumper connection, and `JR` for a recurrent jumper connection. The numbers beside `N` represent the global identification numbers of the neurons, and `x` or `y` represent the inputs coded by the input gene (node).

implicitly in the ordering of the elements of the linear genome. This enables one to evaluate a neural controller represented by it without decoding the neural controller. The evaluation of a linear genome is closely related to executing a linear program using a postfix notation. In the genetic encoding the operands (inputs and jumper connections) come before the operator (a neuron) if one goes from right to left along the linear genome. The linear genome is *complete* in that it can represent any type of neural network. It is also a *compact* encoding of neural networks since the length of the linear genome is the same as the number of synaptic weights in the neural network. It is *closed* under structural mutation and under a specially designed crossover operator. An encoding scheme is said to be closed if all genotypes produced are mapped into a valid set of phenotype networks [5]. The crossover operator exploits the fact that structures originating from some initial structure have some parts in common. By aligning the common parts of two randomly selected structures, it is possible to generate a third structure which contains the common and disjoint parts of the two mother structures. This type of crossover is introduced by Stanley [10]. An example of the crossover operator under which the linear genome is closed is shown in figure 2.

If one assigns integer values to the nodes of a linear genome such that the integer values show the difference between the number of outputs and number of inputs to the nodes, one obtains the following rules useful in the evolution of the neural controllers. The first is that the sum of integer values is the same as

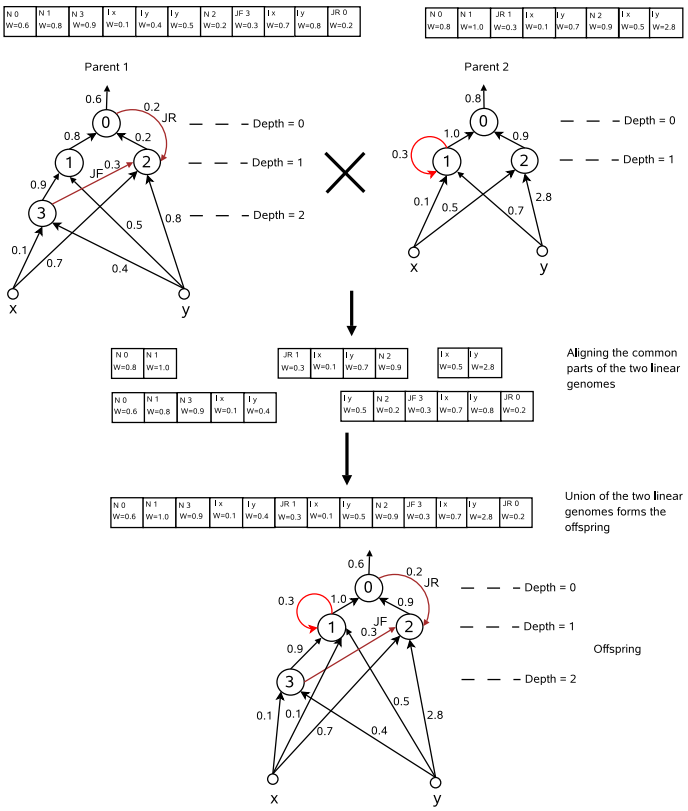


Fig. 2. Performing crossover between two linear genomes. The genetic encoding is closed under this type of crossover operator since the resulting linear genome maps to a valid phenotype network. The weights of the nodes of the resulting linear genomes are inherited randomly from both parents.

the number of outputs of the neural controller encoded by the linear genome. The second is that a sub-network (sub-linear genome) is a collection of nodes starting from a neuron node and ending at a node where the sum of integer values assigned to the nodes between and including the start neuron node and the end node is *one*. Figure 3 illustrates the concept.

The main search operators in EANT are the structural mutation, parametric mutation and crossover operator. The structural mutation adds or removes a forward or a recurrent jumper connection between neurons, or adds a new sub-network to the linear genome. It does not remove sub-networks since removing sub-networks causes a tremendous loss of the performance of the neural controller. The structural mutation operates only on neuron nodes. The weights of a newly acquired topology are initialized to zero so as not to disturb the performance of the network. The parametric mutation is accomplished by per-

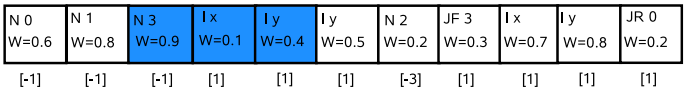


Fig. 3. An example of the use of assigning integer values to the nodes of the linear genome. The linear genome encodes the neural network shown in figure 1. The numbers in the square brackets below the linear genome show the integer values assigned to the nodes of the linear genome. Note that the sum of the integer values is *one* showing that the neural network encoded by the linear genome has only *one* output. The shaded nodes form a sub-network. Note also that the sum of the integer values assigned to a sub-network is always *one*.

turbing the weights of the controllers according to the uncorrelated mutation in evolution strategy or evolutionary programming. Figure 4 shows an example of structural mutation where a neuron node lost connection to an input and received a self-recurrent connection.

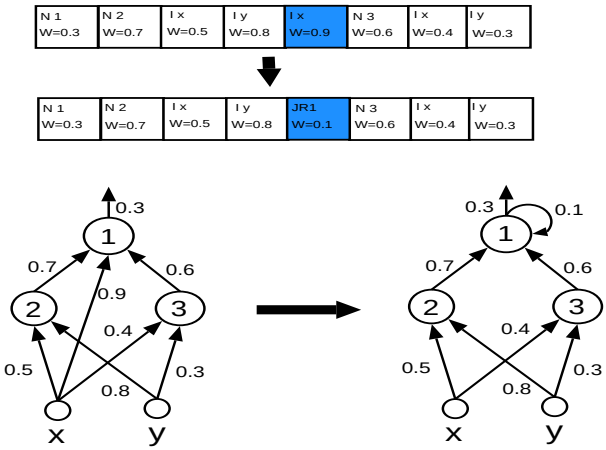


Fig. 4. An example of structural mutation. Note that the structural mutation deleted the input connection to `N1` and added a self-recurrent connection to it.

The initial structures are generated using either the grow or full method [2]. The initial complexity of the neural structures is determined by the domain expert and is specified by the maximum depth that can be assumed by the initial structures. The depth of a neuron node in a linear genome is the minimal number of neuron nodes that must be traversed to get from the output neuron to the neuron node, where the output neuron and the neuron node lie within the same sub-network that starts from the output neuron. The system starts with

exploitation of structures that are already in the system. By exploitation, we mean optimization of the weights of the structures. This is accomplished by an evolutionary process that occurs at smaller time-scale. The evolutionary process at smaller time-scale uses parametric mutation and recombination operators as a search operator. Exploration of structures is done through structural mutation and crossover operator. The structural selection operator that occurs at larger time-scale selects the first half of the population to form the next generation. In order to protect the structural innovations or discoveries of the evolution, young structures that are less than few generations old with respect to the larger time-scale are carried over along the evolution regardless of the results of the selection operator. New structures that are introduced through structural mutation and which are better according to the fitness evaluations survive and continue to exist. Since sub-networks that are introduced are not removed, there is a gradual increase in the complexity of structures along the evolution. This allows EANT to search for a solution starting from a minimum structural complexity specified by the domain expert. The search stops when a neural controller with the necessary optimal structure that solves a given task is obtained.

2 Evolving Neural Controller for Navigation

The aim of this experiment is to demonstrate the automatic design of neural controllers for robots using EANT. We start with sonar based robot navigation for developing the controllers in simulation and then transfer the developed controllers to real robot. We evolved the structure and weights of the neural controller which enables B21 [9] robots to autonomously explore the environment and avoid obstacles. The controller is expected to avoid dead lock situations where Braitenberg-like controllers [3] have difficulties of escaping them. In these situations, they either come to a rest or start to oscillate left to right.

We used the sonar sensors of the B21 robot for detecting the obstacles. The B21 robot has 24 sonar sensors which are symmetrically distributed around its cylindrical body. We used the 8 in front and 2 in the rear sonar sensors as inputs to the neural controller. The sonar sensors give the distance of obstacles in millimeters measured from the center of the robot. The values returned by the sonar sensors are transformed using equation(1) before feeding them to the neural controller.

$$V_n = \begin{cases} \frac{-V_s+2000}{2000} & \text{if } V_s < 1000 \\ 0 & \text{otherwise} \end{cases}. \tag{1}$$

In the equation, V_n is the transformed and normalized sonar reading and V_s is the actual reading returned by a particular sonar sensor. The value V_n lies between 0 and 1 for obstacles which are located at a distance less than 2 m from the center of the robot.

The initial controller has two output neurons and each neuron is connected to all sensors. The outputs of the neurons are connected to the motor apparatus of the robot. In addition to the sensor inputs, each neuron has a constant bias

input connected to it. The forward translational velocity and rotational velocity of the robot are given by $V_t = 0.5(O_1 + O_2)$ and $R_t = O_1 - O_2$, respectively. The quantities O_1 and O_2 are the outputs of the neural network. Since the output of the neurons is between -1 and 1, the maximum and minimum forward velocity of the robot is 1 m/s and -1 m/s, respectively. The rotational velocity is bounded between 2 rad/s and -2 rad/s.

The initial controller is similar to Braitenberg-like controller and is not capable of avoiding dead lock situations. The algorithm is expected to find a controller which is complex enough for solving the navigation problem with the ability of avoiding dead lock situations. The fitness function used to evaluate the controllers is given by

$$F = \sum_{t=1}^{T} D(t) e^{-100(H(t)-H(t-1))^2} (1 - S_{max}(t)), \tag{2}$$

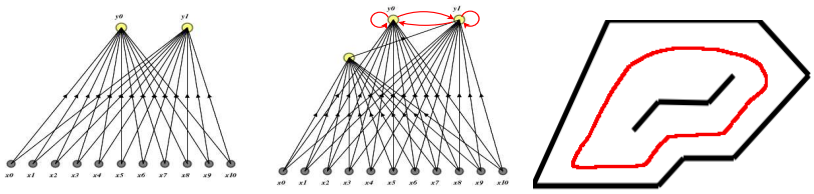
where $D(t)$, $H(t)$ and $S_{max}(t)$ are the distance traveled, the heading of the robot, and the maximum value of the currently perceived normalized sonar readings respectively. The fitness function favors controllers that move straight as long and as fast as possible and controllers that give the robot the maximum distance from the obstacles. Figure 5 shows the initial neural controller and the final controller obtained by our algorithm. The ability of avoiding the deal lock situations comes because of the recurrent connections. The result is similar to that obtained by Nolfi and Floreano [8] and Hülse and Pasemann [4] but in both cases the structure of the neural controller is determined manually beforehand. Ahrns et.al [1] designed a fuzzy-neuro controller for solving the robot navigation with obstacle avoidance. They solved the dead lock situations problem by designing a feature extraction mechanism that extracts a free space direction closest to the heading of the vehicle. They further stored the sonar readings in a short time memory to extract the coarse model for the direct robot surroundings. They used the feature extraction mechanism since the fuzzy-neuro controller does not have recurrent connections.

3 Conclusion

In this paper we have demonstrated the automatic design of controllers for real robots using EANT taking as an example robot navigation with reactive obstacle avoidance. EANT found a clever solution that gives the robot the ability to explore the environment without being trapped in dead-lock situations, where simple designs like Braitenberg-like controllers have difficulties of escaping them.

Acknowledgment

This work is sponsored by the German Academic Exchange Service (DAAD) under grant code R-413-A-01-46029 and the COSPAL project under the EC contract no. FP6-2003-IST-2004176, which are duly acknowledged.



(a) (b) (c)

Fig. 5. (a) The initial Braitenberg-like controller (b) The best neural controller found by EANT that is capable of avoiding dead lock situations. (c) Trajectory of the robot controlled by the best neural controller.

.

References

1. I. Ahrns, J. Bruske, G. Hailu, and G. Sommer. Neural fuzzy techniques in sonar-based collision avoidance. In L.C. Jain and T. Fukuda, editors, *Soft Computing for Intelligent Robotic Systems*, Studies in Fuzziness and Soft Computing, pages 185–214. Physica-Verlag (Springer), 1998.
2. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Applications.* Morgan Kaufmann, San Francisco, CA, 1998.
3. V. Braitenberg. *Vehicles. Experiments in Synthetic Psychology.* The MIT Press, Massachusetts, London, 1994.
4. M. Hülse and F. Pasemann. Dynamical neural schmitt trigger for robot control. In *Proceedings of International Conference on Artificial Neural Networks (ICANN 2002)*, pages 783–788. Springer-Verlag, 2002.
5. J. Jung and J. Reggia. A descriptive encoding language for evolving modular neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 519–530. Springer-Verlag, 2004.
6. Y. Kassahun and G. Sommer. Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *Proceedings of the 13th European Symposium on Artificial Neural Networks (ESANN)*, pages 259–266, Bruges, Belgium, April 2005. d-side publications.
7. Y. Kassahun and G. Sommer. Evolution of neural networks through incremental acquisition of neural structures. Technical Report 0508, Institute of Computer Science and Applied Mathematics, Christian-Albrechts University, Kiel, Germany, June 2005.
8. S. Nolfi and D. Floreano. *Evolutionary Robotics. The Biology, Intelligence, and Technology of Self-Organizing Machines.* The MIT Press, Massachusetts, London, 2000.
9. RWI. *B21 Users Guide.* Real World Interface, Inc., 1997.
10. K. O. Stanley. *Efficient Evolution of Neural Networks through Complexification.* PhD thesis, Artificial Intelligence Laboratory. The University of Texas at Austin., Austin, USA, August 2004.