# Evolutionary Learning of a Neural Robot Controller

Pieter Spronck, Ida G. Sprinkhuizen-Kuyper, Eric O. Postma
Universiteit Maastricht, IKAT
Neural networks and adaptive behaviour group
P.O. Box 616, NL-6200 MD Maastricht, The Netherlands
{p.spronck, kuyper, postma}@cs.unimaas.nl

**Abstract**

In our research we use evolutionary algorithms to evolve robot controllers for executing elementary behaviours. This paper focuses on the behaviour of pushing a box between two walls. Successful execution of this behaviour is critically dependent on the robot's ability to distinguish between the walls and the box. In a recent study we already found that a two-layer recurrent network is more suitable for this task than a single-layer feedforward network [8]. In this paper, we conclude that this result still holds if a more general representation of a feedforward network is used. We also conclude that network architecture changing genetic operators may help reducing a starting network configuration to a smaller one, which is easier to train for the task at hand.

## 1. Introduction

Low-level behaviour forms the foundation of complex high-level behaviour [1]. This study focuses on the low-level behaviour of pushing an object (in this case a circular box) between two walls [2]. The elementary behaviour of pushing is relevant for, for instance, the game of robot soccer, a platform for autonomous systems research [3], and underlies many more complex behaviours such as target following, navigation and object manipulation. We employ the design method of evolutionary robotics [4,5,6]. The method applies evolutionary algorithms [5,7] to the development of robot controllers.

In autonomous robots, the execution and co-ordination of behaviour is mediated by a controller. Artificial neural networks are commonly employed to realise a robot controller. The topology of the neural controller determines to a large extent the range and quality exhibited by the robot. Previous research employed evolutionary algorithms to optimise the box-pushing behaviour of specific network topologies [8]. This paper studies how the addition of architecture-changing genetic operators, i.e. operators that change the architecture of the network, affects box-pushing behaviour.

The outline of the paper is as follows. Section 2 gives an overview of previous research. Section 3 describes the experimental procedure employed in the experiments described in this paper. The results are given in section 4 and discussed in section 5. Finally, section 6 concludes and points at future work.

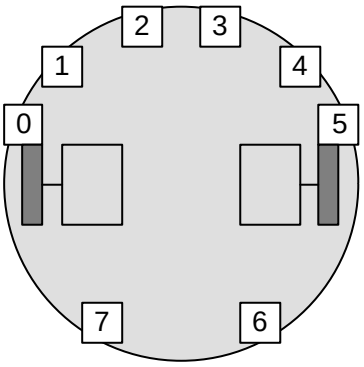
## 2. Previous Research

Recently we have studied several different aspects of the evolution of controllers for box-pushing. In our studies we employed a publicly available mobile robot simulator based on the widely used mobile robot Khepera [9]. A schematic illustration of the robot is shown in figure 1. The (simulated) Khepera is equipped with eight proximity sensors and two motors.

We augmented the simulator with a movable object, i.e. a circular box. The box-pushing task was introduced by Lee, Hallam and Lund [7]. Our work differs from Lee *et al.*'s work in three respects. First, we evolve the box-pushing behaviour between two walls, whereas Lee *et al.* evolved the behaviour in an open, unconstrained space. Second, we use a different fitness function than employed by Lee *et al.* Third, we employ evolutionary algorithms to evolve the controller, whereas Lee *et al.* used genetic programming.

One of our studies focused on the type of fitness function to be used for successful box-pushing behaviour [2]. The study compared four types of fitness functions, each of which was either "global" or "local" in combination with being either "internal" or "external". Global fitness is defined as the fitness of the robot as determined over the complete simulation run, whereas local fitness is defined as the fitness measured for each simulation step. Internal fitness is defined as being determined solely from sensor data, whereas external fitness also includes environmental information (e.g. the position of the robot and the box). Experiments revealed that the combination of global and external fitness would yield the best box-pushing results.

Another study focused on the appropriate neural network topology for box-pushing behaviour [8]. A single-layer feedforward network was compared to a two-layer recurrent network with three hidden nodes. In addition, the effect of augmenting the networks with so-called edge-detecting input nodes was studied. Edge-detecting input nodes have activation values defined as the difference between neighbouring robot sensor values (see figure 1). The experimental results showed recurrent networks augmented with edge-detecting inputs to outperform all other configurations.




Neural controller inputs:

| | |
|---|---|
| 0 to 7: | Sensor 0 to 7 distance values. |
| 8: | Sensor 0 - 1 |
| 9: | Sensor 1 - 2 |
| 10: | Sensor 2 - 3 |
| 11: | Sensor 3 - 4 |
| 12: | Sensor 4 - 5 |
| 13: | Sensor 6 - 7 |

Inputs 8 to 13 are the edge detectors.

Figure 1: Schematic top-view of a Khepera robot with eight infra-red proximity sensors (the white squares) and two motors coupled to wheels (the large shaded rectangles). The sensors measure light intensity and proximity. In the present study, only the proximity measurements are used.

## 3. Experimental Procedure

### 3.1. The simulation environment

All simulations described in this paper were performed using a new environment for evolving neural controllers. This environment, called "Elegance" [10,11], was originally developed to evolve neural controllers for simulated plants. Elegance is particularly suited for evolving the

weights in combination with the neural network architecture. It supports several different network topologies and is very flexible in setting up the evolutionary algorithm. We felt that by using this environment we could deepen our study of neural network topologies for box-pushing control.

For the present research, we made five adaptations to Elegance. First, we defined the box-pushing robot as a new plant. Second, the layered recurrent network was added as a new neural controller. Third, a few modifications were made to the parameters of the evolutionary algorithm. Fourth, a new global external fitness measure was incorporated in the simulation. Finally, some functions were added to monitor the behaviour of the robot and to visualise its behaviour on the box-pushing task (see figure 2).

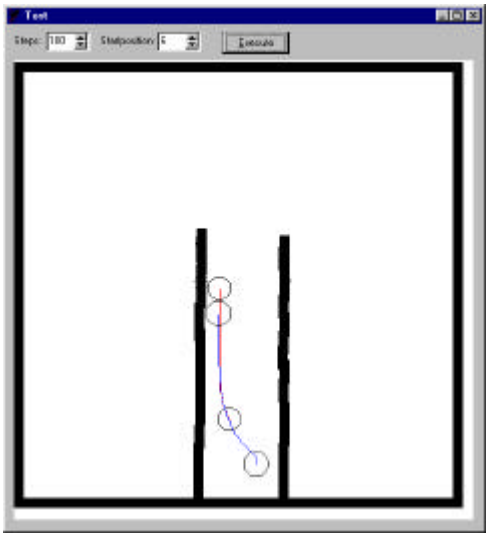


Figure 2: Screenshot of Elegance displaying the behaviour of an evolved neural controller pushing a box between two walls. The circles shown are the robot (largest circles) and the box (smaller circles) in their starting (bottom) and ending (top) positions. The lines running from the starting to the ending positions represent the paths followed by the robot and the box. As can be observed, the robot pushes the box against the left wall and then slides it along the wall towards the top of the field. Note that the walls, especially the right one, are rough, making the task of sliding the box along it a difficult one due to friction.

### 3.2. The neural controllers

In the present study, two neural controller topologies are compared: a feedforward network with five hidden nodes and a layered recurrent network with four hidden nodes. The number of possible connections in these networks is about equal (the recurrent network has a few more). The feedforward network is more general than the more common layered feedforward network, since every connection is allowed as long as a feedforward flow of data through the network is possible. For that purpose, the hidden nodes in the feedforward network are ordered and connections between hidden nodes are restricted to run from "lower" nodes towards "higher" nodes. Also, the input nodes may be directly connected to the output nodes. This topology is more general than the single-layer feedforward network used in our earlier experiments. The layered recurrent network is less general than a completely recurrent network, because only recurrent connections within a layer are allowed. Figure 3 illustrates the two neural network topologies. Both networks are augmented with edge-detecting inputs (cf. section 2, [8]).

The controller directs the two wheels of the robot. However, if a neural controller for one of the wheels is developed, the same network, with some of its inputs mirrored and a few signs switched, can be used to direct the other wheel. We exploited this mirror-symmetry of the

control problem (as we did in our earlier experiments) by creating a network with two output nodes, one for each of the motors, copying the appropriate connections and disallowing the superfluous ones. A linear transfer function was used to compute the output of the nodes (i.e. $f(x) = x$ where $f(x)$ is the output of a neuron and $x$ is the weighted sum of the inputs). A sigmoid transfer function was used for the output node to map the value to the range [-10,+10]. Subsequently, the output value was rounded to the nearest integer value, as is required for controlling the motors of the simulated robot.

It should be noted that our controllers contain redundancies. The edge-detecting input nodes do not add to the theoretical potential of the controller. A network with edge-detecting inputs can be converted into a network without edge-detecting inputs, yet be functionally equivalent. Similarly, the linear hidden nodes in the feedforward network do not add to the expressiveness of the controller. However, our earlier experiments [8] proved the feasibility of incorporating redundancies in the controllers for evolving good box-pushing performance.

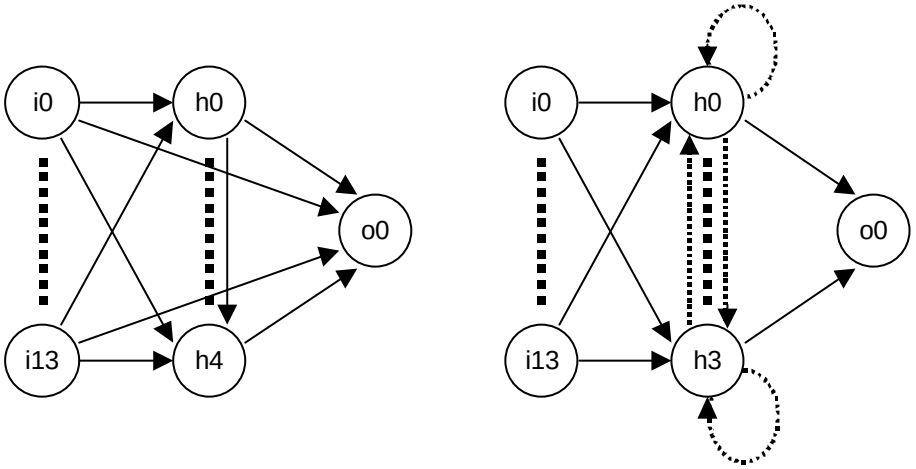



Figure 3: Illustration of a feedforward network (left) and a recurrent network (right) as used in the experiments. The hidden nodes in the feedforward network are ordered. Connections between hidden nodes in this network are restricted in the sense that "lower" nodes are only allowed to connect to "higher" nodes. Recurrent connections are represented by dotted arrows.

### 3.3. The evolutionary algorithm

The chromosome representing a neural network consists of an array of connection genes. Each connection gene represents a single possible connection of the network. A connection gene is one bit coupled to a real number. The bit represents the presence or absence of a connection and the real value specifies the weight of the connection. Even if a connection is absent, the weight value is maintained in the chromosome to facilitate the evolution process by functioning as a "memory" for removed weight values.

Initialisation of the feedforward networks in the starting population was each connection having a 50% chance of being present, and the connection weights having a random value in the range [-25,+25]. For the recurrent network, each connection also had a chance of 50% of being present. Because large-valued recurrent connections tend to overflow the activation values, the weight initialisation was done with random values in the smaller range of [-1,+1].

The genetic operators that were used are:

- Uniform crossover;
- Biased weight mutation [12] with a chance of 10% to change each weight, in a range of [-5,+5] for the feedforward network and in a range of [-0.3,+0.3] for the recurrent network;
- Biased nodes mutation [12], changing just one node within the same range as the biased weight mutation;
- Nodes crossover [12] picking half the nodes of each parent;
- Node existence mutation [10], with a chance of 95% to remove a node and a chance of 5% to completely activate a node;
- Connectivity mutation [13] with each connection having a 10% chance to flip.

When a genetic operator was to be applied, one of the operators was randomly selected, whereby uniform crossover and biased weight mutation were selected twice as often as the others. For the crossover operators, the best of the children was added to the population, and the other one removed.

A population of size 100 was used, wherein newly generated individuals replaced existing individuals in the population, taking into account elitism [14]. Crowding [14] with a factor of 3 was used as replacement policy. For the selection process tournament selection with a tournament size of 3 was used. A maximum of 3500 new individuals were generated during each evolution run.

The fitness measure used was the same as was determined by earlier experiments as giving the best results [2], namely the distance between the box at its start position and the box at its end position minus half the distance between the robot and the box at their end positions (to see if the robot is still pushing). The robot had 100 time steps available to push the box. Three different starting positions for the box were used, and three different starting positions for the robot, making a total of nine different starting positions for the experiments, which are shown in figure 4. The fitness was defined as the mean fitness obtained over the nine runs. For fitter controllers, we averaged over multiple trials to reduce the effects of noise intrinsic to the robot simulator. Controllers with a fitness higher than 250 were averaged over 10 trials and the fittest controller in the population was averaged over 100 trials.

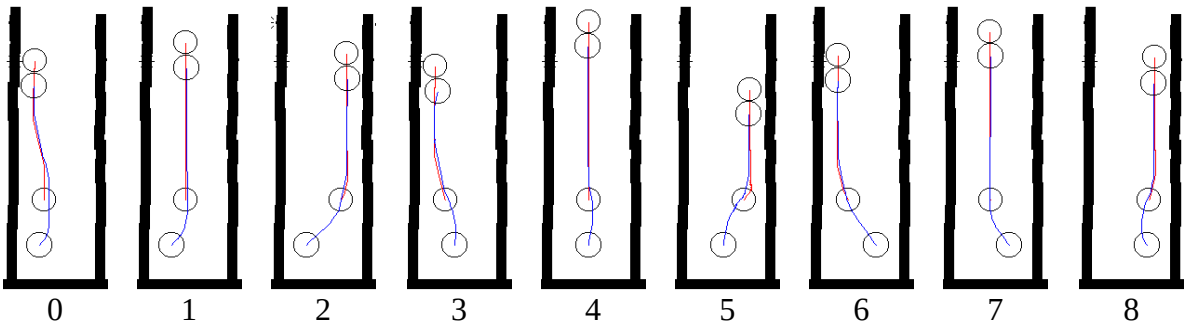



Figure 4: The nine different starting positions used in the experiments. The lines and ending positions illustrate typical box-pushing behaviour.

## 4. Results

### 4.1. Suitability of topologies

As can be concluded from table 1, the experiments confirmed our earlier findings [8] that recurrent controllers tend to perform better (have a higher fitness) and more reliable (have a lower standard deviation) than feedforward controllers. We also discovered the probable reason for that. When observing how the wheels of a feedforward controlled robot move, we noticed that they almost exclusively take on one the two extreme control values: -10 and +10, which initiate the behaviours "very fast backwards" and "very fast forwards". This behaviour works well when pushing the box without hindrance, but makes it difficult for the robot to manoeuvre itself out of a problematic position. This causes the robot controlled by a feedforward network to get stuck regularly. A recurrent controller, on the other hand, shows more subtle behaviour, and even though it sporadically gets stuck, it does so less often than the feedforward controller. This explains both the higher fitness and the lower standard deviation of recurrent controllers.

| | Feedforward controller | | Recurrent controller | |
|---|---|---|---|---|
| Starting position | Mean fitness | St. deviation | Mean fitness | St. deviation |
| Position 0 | 345.7 | 20.5 | 341.2 | 8.7 |
| Position 1 | 333.6 | 28.1 | 336.1 | 7.1 |
| Position 2 | 224.0 | 43.3 | 257.3 | 15.5 |
| Position 3 | 245.8 | 83.2 | 307.1 | 23.0 |
| Position 4 | 387.5 | 6.0 | 376.8 | 5.5 |
| Position 5 | 190.5 | 39.4 | 237.0 | 20.0 |
| Position 6 | 209.9 | 71.5 | 277.9 | 52.0 |
| Position 7 | 307.5 | 56.5 | 336.6 | 7.5 |
| Position 8 | 301.3 | 22.1 | 330.9 | 21.2 |
| Overall | 282.9 | 90.2 | 311.2 | 49.6 |

Table 1: Mean values of the results of the best feedforward and recurrent controllers averaged over seven evolution runs.

### 4.2. Architecture evolution

Two architecture-changing genetic operators were used, "node existence mutation" [10] and "connectivity mutation" [13]. The main rationale for using these operators is as follows. Smaller networks are easier to evolve, but if one starts with too small a network, the evolution process might not succeed at all. Therefore, the use of architecture-changing genetic operators allows for starting with larger neural networks, which are allowed to have superfluous nodes and connections, because unnecessary nodes and connections will be removed during the evolution. Inspection of the fittest controllers revealed that from the feedforward network typically two or three of the five hidden nodes, and several of the input connections were removed. Surprisingly, in the fittest recurrent networks, most of the recurrent connections were removed. Usually, of the sixteen possible recurrent connections only three or four remained.

The solutions found with these experiments are at least equivalent to solutions for similar neural network topologies discovered in the past. This shows that the architecture-changing genetic operators are effective in discovering suitable and parsimonious box-pushing controllers, even when starting out with a larger-than-needed network.

## 5. Discussion

### 5.1. Expressiveness

As stated in section 3.2, the hidden nodes of the feedforward network do not add to the expressiveness of the network. However, when we converted the best feedforward controllers discovered with these experiments to a single-layer feedforward network (that is, without the hidden nodes), the weights in the resulting network became very large, sometimes even hundreds of times larger than the largest weight in the original network. This suggests that a redundant feedforward network with linear hidden nodes is easier to evolve than its single-layer equivalent.

### 5.2. Initialisation ranges

As stated in section 4.1, the recurrent network's outperformance of the feedforward network can be explained by the fact that the recurrent network shows more subtle behaviour than the feedforward network. The large initialisation range of [-25,+25] used for the feedforward network as opposed to the smaller range of [-1,+1] used for the recurrent network might be an explanation for this more subtle behaviour, rather than the topological differences between the networks. To see if the behaviour of the feedforward networks can be made more subtle if they are initialised with smaller weights, we did some experiments with initialisation values in a range of [-5,+5], and a smaller mutation range of [-0.3,+0.3] for the "biased weight mutation" and "biased nodes mutation" genetic operators. We found that the resulting controllers tend to have considerably lower fitness ratings than the feedforward controllers we evolved with the larger initialisation range. Therefore, the large initialisation range we use for the feedforward controllers is entirely warranted.

### 5.3. The individual starting positions

Positions 0, 4 and 8 put the box almost vertically above the robot and are therefore considered to be "easy". As table 1 shows, fitnesses for these positions score high for both network types. Positions 2 and 6 put the box and robot furthest apart and should, therefore, be the most difficult starting positions for the box-pushing robot. Surprisingly, we find that the performance on starting position 5 is worse than those on starting positions 2 and 6. For the feedforward network the performance on starting position 3 is also very low. The explanation for these bad performances is that, due to the angle under which the robot and box start in relation to the walls, sometimes (rather often actually for the feedforward network) the box-pushing behaviour of the robot gets the box stuck against the left wall (i.e. starting position 3). It also suffers considerably from the roughness of the right wall (i.e. starting position 5). As is evident from table 1, the recurrent network manages to solve both problems (i.e. starting positions 3 and 5) much better than the feedforward network.

### 5.4. Differences with other experiments

The fitnesses derived with these experiments are much better than the fitnesses reported for earlier experiments [2,8]. This is mainly because of an idiosyncrasy in the previous implementation.

Although in the present study a linear transfer function has been used, in preliminary experiments performed with recurrent controllers with sigmoid transfer functions we obtained even better results than the best reported here. Further studies are needed to elaborate on these results.

## 6. Conclusions and future work

We have confirmed our previous conclusions that a recurrent network can be used with more success for the box-pushing task than a feedforward network. We found that this is probably due to the fact that a recurrent network can direct the wheels of the robot with more subtlety then a feedforward network. In addition, we conclude that architecture-changing genetic operators help the evolution process by reducing an initial neural network configuration to a simpler configuration which can be trained more easily.

Practice shows that the controllers developed with these experiments can direct a real-life Khepera robot very well, even better than in the simulations. This is probably due to the fact that the setup of the simulation uses rough walls and has a very high noise level. This indicates that the technique can be used to create a controller which can steer a robot in more difficult environments. This is one of the objectives for future research. Besides that, we will examine the effects of the transfer functions and will experiment with different setups for the evolutionary algorithm.

## References

[1] R.A. Brooks. A robust layered control system for a mobile robot. *IEEE journal of robotics and automation*, pp. 14-23, 1986.

[2] I.G. Sprinkhuizen-Kuyper, R. Kortmann and E.O. Postma. Fitness functions for evolving box-pushing behaviour. In A. van den Bosch and H. Weigand, eds., *Proceedings of the Twelfth Belgium-Netherlands Artificial Intelligence Conference*, pp. 275-282, 2000.

[3] M. Asada and H. Kitano. The robocup challenge. *Robotics and Autonomous Systems*, 29(1):3-12, 1999.

[4] R.Arkin. *Behavior-based robotics*. MIT-press, Cambridge, 1998.

[5] I. Harvey, P. Husbands, D. Cliff, A. Thompson and N. Jakobi. Evolutionary robotics: the sussex approach. *Robotics and autonomous systems*, 20:205-224, 1997.

[6] S. Nolfi. Evolutionary robotics: Exploiting the full power of self-organization. *Connection Science*, 10(3-4):167-183, 1998.

[7] W-P. Lee, J. Hallam and H.H. Lund. Applying genetic programming to evolve behavior primitives and arbitrators for mobile robots. In *Proceedings of IEEE 4th International Conference on Evolutionary Computation*, IEEE Press, 1997.

[8] I. Sprinkhuizen-Kuyper, E.O. Postma and R. Kortmann, Evolutionary Learning of a Robot Controller: Effect of Neural Network Topology. In A. Feelers, ed., *Proceedings of the Tenth Belgium-Dutch Conference on Machine Learning (BENELEARN'01)*, pp. 55-60, 2001.

[9] F. Mondada, E. Franzi and P. Jenne. Mobile robot miniaturisation: A tool for investigating in control algorithms. In T. Yoshikawa and F. Miyazaki, eds., *Proceedings of the Third International Symposium on Experimental Robotics*, pp. 501-513. Springer-Verlag, Berlin, 1993.

[10] P. Spronck and E. Kerckhoffs. Using genetic algorithms to design neural reinforcement controllers for simulated plants. In A. Kaylan and A. Lehmann, eds., *Proceedings of the 11th European Simulation Conference*, pp. 292-299, 1997.

[11] J. Jarmulak, P. Spronck, E. Kerckhoffs, Neural networks in process control: model-based and reinforcement controllers. *Computers and Electronics in Agriculture*, 18:149-166, 1997.

[12] D. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 762-767, Morgan Kaufman, California, 1989.

[13] V. Maniezzo. Searching among search spaces: hastening the genetic evolution of feedforward neural networks. R.F. Albrecht, C.R. Reeves and N.C. Steel, eds., *Artificial Neural Nets and Genetic Algorithms*, pp. 635-642, Springer-Verlag, New York, 1993.

[14] D.E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, 1989.

**Software Availability**

The Khepera simulator described in section 2 is available from the WWW:
http://diwww.epfl.ch/lami/team/michel/khep-sim/.

The program "Elegance" is available from the WWW:
http://www.cs.unimaas.nl/p.spronck/.