Evolving Neural Networks which Control a Simple Robotic Agent

Neruda, Roman
2006

Institute of Computer Science
Academy of Sciences of the Czech Republic

Evolving neural networks which control a simple robotic agent

Roman Neruda, Stanislav Slušný

Technical report No. 986

December 2006

Pod Vodárenskou věží 2, 182 07 Prague 8, phone: +420 266 053 111, fax: +420 286 585 789, e-mail:ics@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Evolving neural networks which control a simple robotic agent

Roman Neruda, Stanislav Slušný[1]

Technical report No. 986

December 2006

Abstract:

The design of intelligent embodied agents represents one of the key research topics of today's artificial intelligence. These agents should be able to adopt to changes of the environment and to modify their behaviour according to acquired knowledge. The goal of this work is the study of emergence of intelligent behaviour within a simple intelligent agent. Cognitive agent functions will be realized by mechanisms based on neural networks and evolutionary algorithms, where the evolutionary algorithm will be responsible for the adaptation of a neural network in a simulated environment.

Keywords:
Evolutionary algorithms, neural networks, robotic agent.

[1]Acknowledgment This work was supported by the Ministry of Education of the Czech Republic under the project Center of Applied Cybernetics No. 1M684077004 (1M0567).

Contents

Chapter 1

Introduction

One of the main goals of Artificial Intelligence (AI) is to gain insight into natural intelligence through a synthetic approach, by generating and analyzing artificial intelligent behavior. In order to glean an understanding of a phenomenon as complex as natural intelligence, we need to study complex behavior in complex environments. Traditionally, AI has concerned itself with complex agents in relatively simple environments, simple in the sense that they could be precisely modeled and involved little or no noise and uncertainty.

In contrast to traditional systems, reactive and behavior based systems have placed agents with low levels of cognitive complexity into complex, noisy and uncertain environments. One of the many characteristics of intelligence is that it arises as a result of an agent's interaction with complex environments. One approach to develop autonomous intelligent agents, called Evolutionary robotics (ER), is through a self-organization process based on artificial evolution. Its main advantage is that it is an ideal framework for synthesizing agents whose behaviour emerge from a large number of interactions among their constituent parts[3].

In the following sections we introduce neural networks and evolutionary algorithms, two basic tools used in evolutionary robotics. Then we look at Khepera robots and related simulation software. In the next chapter, we present two experiments with real Khepera robots. In both of them, artificial evolution will be guiding self-organization process. In first of them, we expect emergence of wall avoidance behaviour. In the second, besides wall avoidance behaviour, we expect emergence of behaviour, that guarantees full maze exploration. In both experiments, agents are controlled by simple feed forward neural networks, and this restriction causes their pure reactive behaviour. However, results are surprisingly good, as we will show. In the last chapter, we draw some conclusions and present directions for our future work.

Chapter 2

Preliminaries

2.1 Neural networks

The robot's control algorithm is based on the neural networks. Neural networks are popular in robotics from various reasons. They provide straightforward mapping from input signals to output signals, several levels of adaptation and are robust to noise.

An multilayer feedforward ANN is an interconnected network of simple computing units called *neurons*, which are ordered in layers, starting with an *input layer* and ending with an *output layer*. [1]. Between these two layers there can be a number of *hidden layers*. Connections in this kind of networks only go forward from one layer to the next. Each neuron has n real inputs, and each input x_i has assigned a real weight w_i.

The output $y(x)$ of a neuron is defined in equation 2.1:

$$y(x) = g\left(\sum_{i=1}^{n} w_i x_i\right), \tag{2.1}$$

where x is the neuron with n input dendrites (x_0 ... x_n), one output axon $y(x)$, (w_0 ... w_n) are weights and $g : \Re \rightarrow \Re$ is the *activation function*. One of the most common activation function is a logistic sigmoid function 2.2

$$\sigma(\xi) = 1/(1 + e^{-\xi t}), \tag{2.2}$$

where t determines its steepness.

In this paper, by training of neural network, we mean modification of its weights. Usually, the sets of examples are provided with learning task, from which neural network is trained by some supervised learning algorithm. However, training neural networks for robot controllers is not that case - training sets are usually not available. The adaptation of neural network is not carried out by a supervised learning algorithm as usually, but instead, an evolutionary algorithm is used.

2.2 Evolutionary algorithms

Evolutionary algorithm has been investigated by John Holland [2], with a marked increase of interest within the last few years. Evolutionary search refers to a class of stochastic optimization techniques — loosely based on processes believed to operate in biological evolution

— in which a population of candidate solutions (chromosomes) evolves under selection and random "genetic" diversification operators. The problem is to find maximum of *fitness function*. Every member of population is called *individual* and represents a potential solution to a problem.

The algorithm 1 reveals skeleton of genetic algorithm used in our experiments. To fully describe genetic algorithm, it is needed to define how each solution will be represented, how initial population will be created and what genetic operators will be used in the Reproduction step.

The standard genetic algorithm uses binary strings to encode alternative solutions. Real-world experiments demand big precision, what causes too long chromosomes, for which the evolution process becomes non-efficient. Therefore, different encoding method is used. The synaptic weights of neural networks are directly encoded on the genotype as a string of real values. In our experiments, traditional one-point crossover and biased mutation operators were used. Initial population consisted of neural networks with randomly initialized weights.

We have used the *roulette wheel selection* in all experiments, which performs the equivalent role to natural selection — it chooses individuals for next population proportionally to their fitness values.

Evaluation of an individual by a fitness value was carried out in a simulation. To evaluate the individual, neural network is constructed from chromosome, environment is initialized and robot is put into randomly chosen starting location (4.1). The inputs of neural networks are interconnected with robot's sensors and outputs with robot's motors. The robot is then left to "live" in the simulated environment for some (fixed) time period, fully controlled by neural network. Depending on how well is robot doing, individual is evaluated by fitness value. The higher evaluation, the more successful robot in executing particular task.

Input: number of individuals in population N, number of elits E, maximum number of populations G_{max}.
Output: the best found solution of a problem.

1. Start: Create initial population of N individuals $P(0) = \{I_1, ..., I_N\}$, $i = 0$.
2. Evaluation of individuals: To compute fitness function for every individual I, build ANN corresponding to I and initialize simulated environment. Let robot (controlled by ANN) freely carry actions for fixed time period. Evaluate robot by real value, depending how well it was doing.
3. Stop-condition: If $i = G_{max}$, finish and return solution represented by individual with minimal value of fitness function.
   (a) Creation of new population $P(i + 1)$ from population $P(i)$: Create empty population $P(i + 1)$.
   (b) Selection: Choose E best individuals from population $P(i)$ and move them to the population $P(i+1)$. Apply selection operator, choose $N - E$ individuals and insert them into the population $P'(i)$.
   (c) Reproduction: Apply crossover and mutation operators on population $P'(i)$, resulting population is $P(i + 1)$.
      i. Crossover step: If population $P'(i)$ contains odd number of individuals, insert chromosome of the first individual from population $P'(i)$ into population $P''(i)$. From population $P'(i)$ choose pairs of chromosomes C and D and apply on them crossover operator, insert new chromosomes C' and D' into population $P''(i)$.
      ii. Mutation step: On every chromosome from population $P''(i)$ apply mutation operator, insert new chromosome into population $P(i + 1)$.
4. New generation: $i = i + 1$, Continue by step 2.

Algorithm 1: Skeleton of the evolutionary algorithm.

Chapter 3

Khepera robots

3.1 Physical robot

Khepera is a miniature mobile robot with a diameter of 70 mm and a weight of 80 g. The robot is supported by two lateral wheels that can rotate in both directions and two rigid pivots in the front and in the back. The sensory system employs eight "active infrared light" sensors distributed around the body, six on one side and two on other side. In "passive mode", they measure the amount of infrared light in the environment, which is roughly proportional to the amount of visible light. In "active mode" these sensors emit a ray of infrared light and measure the amount of reflected light. The closer they are to a surface, the higher is the amount of infrared light measured. The Khepera sensors can detect a white paper at a maximum distance of approximately 5 cm. The robot is equipped with a Motorola MC68331 CPU with 512 Kbytes of EEPROM and 512 Kbytes of static RAM.

Robot miniaturization has several advantages[4]. It makes possible to build complex environments on a limited surface, fundamental laws of physics give higher mechanical robustness to a miniature robot and while the miniature robot will resist the collision, the other robot would probably report serious damages (this is in particular important when evolving real robots without simulation).

3.2 Simulation environments

Although evolution on real robots is feasible, serial evaluation of individuals on a single physical robot might require quite a long time. As stated in [3], the experiment on evolution of a homing navigation performed by Floreano and Mondada took 10 days when carried out entirely on physical robots.

One way to avoid the problem of time is to evolve robots in simulation and then run the most successful individuals on the physical robot. Simulated robots might run faster and the power of parallel computers can be easily exploited for running more individuals at the same time. However, simulation present other problems. There is a real danger that programs which run well on simulated robots will completely fail on real robots because of the differences in real and simulated world sensing. Several evolutionary experiments recently conducted in simulations and successfully validated on real robots demonstrated that, at least for a restricted class of robot-environment interactions, it is possible to develop robots in simulation.

Figure 3.1: Khepera robot built by K-Team SA (www.k-team.com).

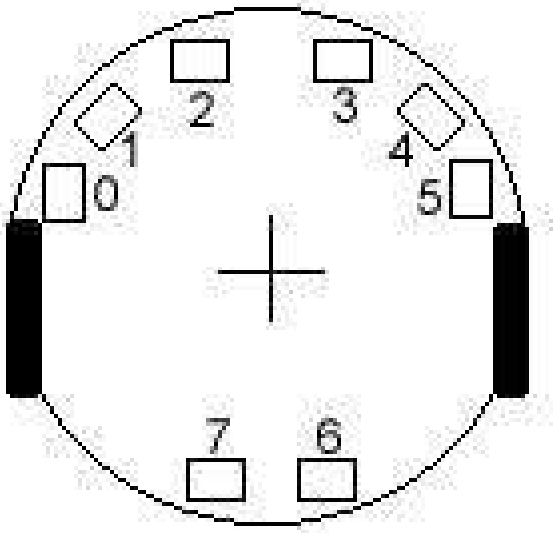

Figure 3.2: Schematic drawing of the robot with infrared promixity sensors, left and right wheel (controlled by independent motors).

One of the widely used simulation software (for Khepera robots) is Yaks simulator, which is freely available. Yaks can also control real robot through serial cable connection, so evolved neural network can be directly tested on real robot. Simulation consists of pre-defined number of discrete steps, each single step corresponds to 100 ms. Yaks allows the user to customize and design world with walls, obstacles and specify start locations. In our experiments, we introduce the concept of *zones*, which are circular areas that provide the ability to reference to that specic location on the map. Zones incorporate features such as the number of times a robot has visited it.

Chapter 4

Experiments

We have conducted two experiments. First experiment is actually navigation with wall avoidance, classic task that most people working in mobile robotics have some experiences with. A robot is put in an environment with some corridors and is required to cover longest possible distance without hitting the objects. In second experiment, the task of robot is to learn effective maze exploration strategy.

Experiments were carried out in simulated environment. Parameters of evolutionary algorithm are shown in table 4.1. Evaluation of robot took at most 200 steps, and was interrupted when the robot hit the wall. Neural networks had one hidden layer with 4 neurons, eight input neurons (corresponding to sensors) and two output neurons (corresponding to motors). Each simulation step, signal values measured by sensors were passed to neural network and outputs from neural networks were sent to motors. Values from sensors are in range $[0, 1]$, the higher value is, the closer obstacle is in front of sensor. Motor speeds have to be in range $[-0.5, 0.5]$, where 0 means motor turned off, 0.5 corresponds to fastest forward speed and -0.5 corresponds to fastest backward speed.

Population size	150
Crossover probability	0.7
Mutation probability	0.7
Number of generations	150

Table 4.1: Parameters of evolutionary algorithm.

The experiments differ only in the way of evaluating individuals by fitness function. Evaluation is started by environment initialization and fitness value of the individual is cleared. Starting position of the robot is chosen randomly from six predefined positions. Each simulation step, fitness value ϕ of an individual is increased by fitness gain $\Delta\phi$.

4.1 Avoiding walls

The goal was to evolve a control system capable of maximizing speed while avoiding walls. The external size of a "maze" was approximately 80x67 cm (figure 4.2).

Similarly to [3], fitness gain in k-th step $\Delta\phi^k$ is composed of three components.

$$\Delta\phi^k = V^k(1 - \sqrt{\Delta v^k})(1 - i^k) \tag{4.1}$$

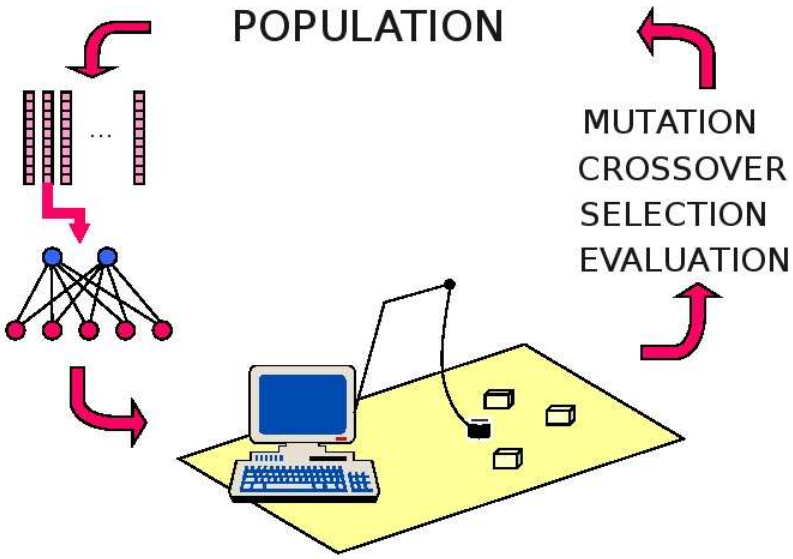

Figure 4.1: Artificial evolution with a single robot. The population of evolving individuals is stored on the computer memory, each individual can be tested in simulated or real environment (reprinted with small modifications from [3]).

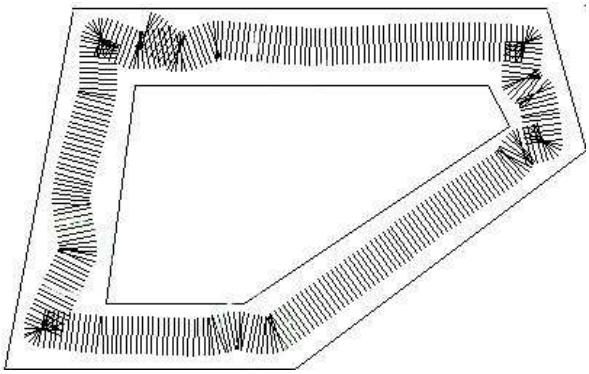

Figure 4.2: Trajectory of the best individual from the last generation in “avoid walls” experiment. Robot is able to move without hitting the walls.

$$V_L^k, V_R^k \in [-0.5, 0.5], V^k = |V_L^k| + |V_R^k| \tag{4.2}$$

$$\Delta v^k = |V_L^k - V_R^k|, \Delta v^k \in [0, 1] \tag{4.3}$$

$$\phi = \sum_{k=1}^{n} \frac{\Delta\phi^k}{n} \tag{4.4}$$

V^k is computed by taking sum of absolute values of left wheel speed V_L^k and right wheel speed V_R^k (thus $V^k \in [0, 1]$). This component is maximized by high rotation speed of the wheels, without regard to direction. Second component $1 - \sqrt{\Delta v^k}$ motivates wheels to rotate in the same direction. Last component is responsible for wall avoidance. The value i^k of the most active sensor in k-th step provides conservative measure of how close the robot is to an object, independently of its relative position. The closer it is to an object, the higher the measured value in a range from 0 to 1.

At the end of evaluation (either after 200 simulation steps or when robot hits the wall), fitness value is divided by the number of steps n, thus finally fitness evaluation is actually an average fitness gain.

Picture 4.2 shows trajectory of the best individual from last generation. Theoretically, if robot would be moving at full speed with no sensors activated, it could obtain maximal fitness value 1. In fact, corridors were quite narrow, so sensors were activated almost all the time. No robot from the last generation was moving at full speed.

As shown in picture 4.2, robot learned to effectively move without hitting the walls. Best individuals from earlier generations were quite successful too, but path was not so smooth yet.

4.2 Maze

In this experiment, robot should learn effective maze exploration strategy. Maze (figure 4.3) has size 100x100 cm. There are two ”zones” randomly located in the maze (green circles in figure 4.3). Robot is rewarded, when passes through these zones and zones are relocated. Zones can not be even sensed by the robot, so robot should learn effective strategy to seek through all corridors in a complex maze.

Similarly to previous experiment, fitness value is increased each simulation step by fitness gain. This time, however, we left two components and fitness gain is increased only for ability to avoid walls. Robot moving is stimulated in another way - at the end of simulation, fitness value is increased by number of visited zones Z.

$$\Delta\phi^k = (1 - i^k) \tag{4.5}$$

$$\phi = \sum_{k=1}^{n} \frac{\Delta\phi^k}{n} + Z \tag{4.6}$$

As shown in picture 4.3, evolved strategy consists of following right wall, what guarantees complete maze exploration. Even trajectory of best individuals is not very smooth, as robot is penalized for being too close to the wall, it is trying to move away from the wall

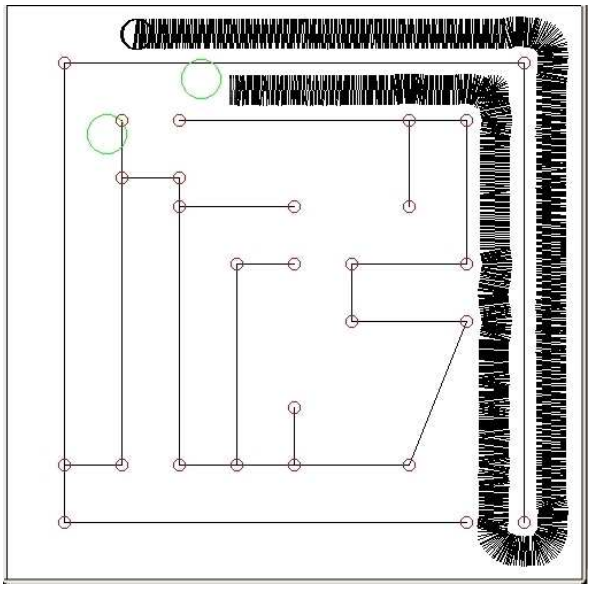

Figure 4.3: Khepera (represented by black circle) in a maze environment. Robot should look for green circles, called “zones”, although it was not able to sense them. When the zone was hit by robot, it was randomly relocated. Evolved strategy, which consisted of right wall following, guarantees complete maze exploration.

but not to loose contact with the wall on its right side. Due to the fact, that robot is not rewarded for the fast movement, robot is moving at significantly slower speed than in previous experiment.

Chapter 5

Conclusions

In the previous section, we presented two experiments, in which self-organization process based on evolutionary algorithm should have developed behaviours, that solve relative simple tasks. In both experiments, feed forward neural network was evolved, that controlled agent's behaviour. Although experiments were conducted in simulation only, neural networks were later transfered to the real Khepera and results were surprisingly good. No learning on real robots was needed. This proves potential of this approach.

Our experiments were concerned one embodied agent only. In recent years, the environmental complexity is scaled up by introducing other agents, and cognitive complexity is scaled up by introducing learning capabilities into each of the agents. Similarly to synthesizing behaviours, we should assume that evolutionary approach can be successfully applied also to synthesize agents able to display collective behaviours. In this case, evolving individual might exploit not only the properties that emerge from the interactions among the constituent elements of the agent and between the agent and the environment but also the interactions among different individual agents.

In our future work, we would like investigate the feasibility of applying the evolutionary method to the synthesis of the control systems of a group of robots able display collective behaviour. Research will also investigate which type of strategies emerge during evolution: these strategies might shed new light on behaviours of animals moving in groups (heards, flocks, ecc.).

Bibliography

[1] J. Šíma, R. Neruda. Teoretické otázky neuronových sítí. Matfyzpress, Praha, 1996.

[2] J. Holland. Adaptation In Natural and Artificial Systems. The University of Michigan Press, Ann Arbour, 1975.

[3] S. Nolfi, D. Floreano. Evolutionary Robotics - The Biology, Intelligence and Techology of Self-Organizing Machines. The MIT Press, 2000.

[4] Mondada, F., E. Franzi and P. Ienne. Miniaturisation: A tool for investigation in control algorithms. In T. Yoshikawa and F. Miyazaki, eds., Proceedings of the Third In ternational Symposium on Experimental Robotics. Springer-Verlag, Berlin, (1993), 501513.