Two Learning Approaches to Maze Exploration: Case Study with E-puck Mobile Robots

Roman Neruda, Stanislav Slušný and Petra Vidnerová[*]

Abstract—**An emergence of intelligent behavior within a simple robotic agent is studied in this paper. Two control mechanisms for an agent are considered: new direction of reinforcement learning called relational reinforcement learning, and a radial basis function neural network trained by evolutionary algorithm. Relational reinforcement learning is a new interdisciplinary approach combining logical programming with traditional reinforcement learning. Radial basis function networks offer wider interpretation possibilities than commonly used multilayer perceptrons. Results are discussed on the maze exploration problem.**

Keywords: Evolutionary robotics, Reinforcement learning, Maze exploration

1 Introduction

One of the key question of the Artificial Intelligence is how to design intelligent agents. Several approaches have been studied so far. In our previous work, we have been examining mainly Evolutionary robotics (ER).

The ER approach attacks the problem through a self-organization process based on artificial evo-lu-ti-on [13]. Robot control system is typically realized by a neural network, which provides direct mapping from robot's sensors to effectors. Most of the current applications use traditional multilayer perceptron networks. In our approach we utilize local unit network architecture called radial basis function (RBF) network, which has competitive performance, more learning options, and (due to its local nature) better interpretation possibilities [18, 19].

This article gives summary of our experiences and comparison to Reinforcement Learning (RL) - another widely studied approach in Artificial Intelligence. RL is focusing on agent, that is interacting with the environment by its sensors and effectors. This interaction process helps agent to learn effective behavior. These kinds of tasks are commonly studied on miniature mobile robots of type Khepera [2] and E-puck [1].

[*]Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod vodárenskou věží 2, Prague 8, Czech Republic, roman@cs.cas.cz. This work was supported by the Ministry of Education of the Czech Republic under the project Center of Applied Cybernetics No. 1M684077004 (1M0567).

2 Related work

The book [16] provides comprehensive introduction to the ER, with focus on robot systems. Recently, effort is made to study emergence of intelligent behavior within the group of the robots.

Pioneering work was done by Martinoli [14]. He solved the task, in which group of simulated Khepera robots were asked to find "food items" randomly distributed on an arena. The control system was developed by the artificial evolution. Our work with single robot and robot teams were published in [19, 18].

Reinforcement learning is gaining increasing attention in recent years. The basic overview of the field can be found in [20].

3 Evolutionary robotics

The evolutionary algorithms (EA) [13, 12] represent a stochastic search technique used to find approximate solutions to optimization and search problems. They use techniques inspired by evolutionary biology such as mutation, selection, and crossover. The EA typically works with a population of *individuals* representing abstract representations of feasible solutions. Each individual is assigned a *fitness* that is a measure of how good solution it represents. The better the solution is, the higher the fitness value it gets. The population evolves toward better solutions. The evolution starts from a population of completely random individuals and iterates in generations. In each generation, the fitness of each individual is evaluated. Individuals are stochastically selected from the current population (based on their fitness), and modified by means of operators *mutation* and *crossover* to form a new population. The new population is then used in the next iteration of the algorithm.

Feed forward neural used as robot controllers are encoded in order to use them the in the evolutionary algorithm. The encoded vector is represented as a floating-point encoded vector of real parameters determining the network weights.

Typical evolutionary operators for this case have been used, namely the uniform crossover and the mutation which performs a slight additive change in the parameter value. The rate of these operators is quite big, ensuring the exploration capabilities of the evolutionary learning. A standard roulette-wheel selection is used together with a small elitist rate parameter. Detailed discussions about the fitness function are presented in the next section.

4 Relational Reinforcement Learning

The lack of theoretical insight into EA is the most serious problem of the previous approach. The RL is based on dynamic programming [6], which has been studied more than 50 years already. It has solid theoretical backgrounds built around Markov chains and several proved fundamental results. On the other side, it is not possible usually to fulfill theoretical assumptions in the experiments.

The general model of agent-environment interaction is modeled through the notion of rewards. The essential assumption of RL states, that agent is able to sense rewards coming from the environment. Rewards evaluate taken actions, agent's task is to maximize them. The next assumption is that agent is working in discrete time steps. Symbol S will denote finite discrete set of states and symbol A set of actions. In each time step t, agent determines its actual state and chooses one action. Therefore, agent's life can be written as a sequence

$$o_0 a_0 r_0 s_1 a_1 r_1 ... \qquad (1)$$

where s_t denotes state, which is determined by processing sensors input, $a_t \in A$ action and finally symbol $r_t \in R$ represents *reward*, that was received at time t.

Formally, agent's task is to maximize

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{i=0} \gamma^i r_{t+1} \qquad (2)$$

where the quantity $V^{\pi}(s_t)$ is called discounted cumulative reward. It is telling us, what reward can be expected, if the agent starts in state s_t and follows policy π, $0 \leq \gamma < 1$ is a constant that determines the relative value of delayed versus immediate rewards.

The most serious assumption of RL algorithms is the *Markov property*, which states, that agent does not need history of previous states to make decision. The decision of the agent is based on the last state s_t only. When this property holds, we can use theory coming from the field of *Markov decision processes* (MDP).

The strategy π, which determines what action is chosen in particular state, can be defined as function $\pi : S \rightarrow A$, where $\pi(s_t) = a_t$. Now, the agent's task is to find optimal strategy π^*. Optimal strategy is the one, that maximalizes expected reward. In MDP, single optimal deterministic strategy always exists, no matter in what state has the agent started.

Optimal strategy π^* can now be defined as

$$\pi^* = argmax_{\pi} V^{\pi}(s), \forall s \in S \qquad (3)$$

To simplify the notation, let's write $V^*(s)$ instead of symbol V^{π^*}, value function corresponding to optimal strategy π^*.

$$V^*(s) = max_{\pi} V^{\pi}(s) \qquad (4)$$

The first breakthrough of RL was the Q-learning algorithm [21, 4], which computes optimal strategy in described conditions.

The key idea of the algorithm is to define the so-called *Q-values*. $Q^{\pi}(s, a)$ is the expected reward, if the agent takes action a in state s and then follows policy π.

$$Q^{\pi}(s, a) = r(s, a) + \gamma V^{\pi}(s'), \qquad (5)$$

where s' is the state, in which agent occurs taking action a in state s ($s' = \delta(s, a)$).

It is probably most commonly used algorithm of RL, mainly because of its simplicity. However, several improvements have been suggested to speed up the algorithm. In real life applications, state space is usually too big and convergence toward optimal strategy is slow. In recent years, there have been a lot of efforts devoted to rethinking idea of states by using function approximators [7], defining notion of options and hierarchical abstractions [5]. Relational reinforcement learning [11] is approach that combines RL with Inductive Logical Programming.

The distinction between classical RL and Relational Reinforcement Learning is the way how the Q-values are represented. In classical Q-learning algorithm are Q-values stored in the table. In relational version of the algorithm, they are stored in the structure called *Logical decision tree* [8]. In our experiments, we have used logical decision trees as implemented in the programs TILDE [8] from package ACE-ilProlog [9].

5 Evolutionary RBF Networks

Evolutionary robotics combines two AI approaches: neural networks and evolutionary algorithms. Neural network realizes a control system of the robot. It gets on input values from robot's sensors and its outputs control the wheels.

Evolutionary algorithms [13, 12] are then used to train such a network. It would be difficult to utilize the training by traditional supervised learning algorithms since they require instant feedback in each step. Here we typically can evaluate each run of a robot as a good or bad one, but it is impossible

```
• for each $s, a$ do
    – initialize the table entry $Q'(s, a) = 0$
    – $e = 0$
• do forever
    – $e = e + 1$
    – $i = 0$
    – generate a random state $s_0$
    – while not goal($s_i$) do
        ∗ select an action $a_i$ and execute it
        ∗ receive an immediate reward $r_i = r(s_i, a_i)$
        ∗ observe the new state $s_{i+1}$
        ∗ $i = i + 1$
    – endwhile
    – for $j = i - 1$ to 0 do
        ∗ update $Q'(s_j, a_j) = r_j + \gamma \max_{a'} Q'(s_{j+1}, a')$
```

Figure 1: Scheme of Q-learning algorithm, taken from [11].

to assess each one move as good or bad. Thus, the evolutionary algorithm represent one of the few possibilities, how to train the network.

The *RBF network* [17, 15, 10], used in this work, is a feed-forward neural network with one hidden layer of *RBF units* and linear output layer. The network function is given by Eq. (7).

$$y(\vec{x}) = \varphi\left(\frac{\| \vec{x} - \vec{c} \|}{b}\right) \tag{6}$$

$$f_s(\vec{x}) = \sum_{j=1}^{h} w_{js}\varphi\left(\frac{\| \vec{x} - \vec{c}_j \|}{b_j}\right), \tag{7}$$

where f_s is the output of the s-th output unit, y is the output of a hidden unit, φ is an activation function, typically Gaussian function $\varphi(s) = e^{-s^2}$.

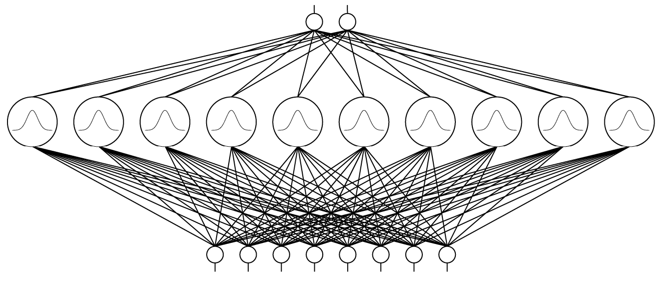

Figure 2: A scheme of a Radial Basis Function Network.

The evolutionary algorithm is summarised in Fig. 3. It works with a population of *individuals* representing abstract representations of feasible solutions. Each individual is assigned a *fitness* that is a measure of how good solution it represents. The evolution starts from a population of completely random individuals and iterates in generations. Individuals are stochastically selected from the current population (based on their fitness), and modified by means of genetic operators *mutation* to form a new generation.

```
1. START: Create population $P(0) = \{I_1, \cdots, I_N\}$.
2. FITNESS EVALUATION: For each individual evaluate fitness function.
3. TEST: If the stop criterion is satisfied, return the solution.
4. NEW GENERATION: Create empty population
   $P(i+1)$ and repeat the following procedure until
   $P(i+1)$ has $N$ individuals.
     i) Selection: Select two individuals from $P(i)$ :
        $I_1 \leftarrow selection(P_i)$,
        $I_2 \leftarrow selection(P_i)$.
    ii) Crossover: With probability $p_c$:
        $(I_1, I_2) \leftarrow crossover(I_1, I_2)$
   iii) Mutation: With probability $p_m$:
        $I_k \leftarrow mutate(I_k), k = 1, 2$
    iv) Insert: Insert $I_1, I_2$ into $P_{i+1}$
5. LOOP: Go to step 2.
```

Figure 3: Scheme of an evolutionary algorithm.

In case of RBF networks learning, each individual encodes one RBF network. The individual consists of h blocks:

$$I_{RBF} = \{B_1, \ldots, B_h\}, \tag{8}$$

where h is a number of hidden units. Each of the blocks contains parameter values of one RBF units:

$$B_k = \{c_{k1}, \ldots, c_{kn}, b_k, w_{k1}, \ldots, w_{km}\}, \tag{9}$$

where n is the number of inputs, m is the number of outputs, $\vec{c_k} = \{c_{k1}, \ldots, c_{kn}\}$ is the k-th unit's centre, b_k the width and $\vec{w}_k = \{w_{k1}, \ldots, w_{km}\}$ the weights connecting k-th hidden unit with the output layer. The parameter values are encoded using direct floating-point encoding.

We use standard *tournament selection*, *1-point crossover* and *additive mutation*. Additive mutation changes the values in the individual by adding small value randomly drawn from $\langle -\epsilon, \epsilon \rangle$.

The fitness function should reflect how good the robot is in given tasks and so it is always problem dependent. Detailed description of the fitness function is included in the experiment section.

Figure 4: Miniature mobile e-puck robot.

Sensor value	Meaning
0-50	NOWHERE
51-300	FEEL
301-500	VERYFAR
501-1000	FAR
1001-2000	NEAR
2001-3000	VERYNEAR
3001-4095	CRASHED

Table 1: Sensor values and their meaning.

6 Experiments

In order to compare performance and properties of described algorithms, we conducted simulated experiment. Miniature robot of type e-puck[1] was trained to explore the environment and avoid walls. E-puck is a mobile robot with a diameter of 70 mm and a weight of 50 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 e-puck sensors can detect a white paper at a maximum distance of approximately 8 cm. Sensors return values from interval $[0, 4095]$. Effectors accept values from interval $[-1000, 1000]$. The higher value, the faster the motor is moving.

Without any further preprocessing of sensor’s and effector’s values, the state space would be too big. Therefore, instead of raw sensor values, learning algorithms worked with “perceptions”. Instead of 4095 raw sensor values, we used only 5 perceptions(table 1). Effector’s values were processed in similar way: instead of 2000 values, learning algorithm chosen from values $[-500, -100, 200, 300, 500]$. To reduce the state space even more, we grouped pairs of sensors together and back sensors were not used at all.

The agent was trained in the simulated environment of size

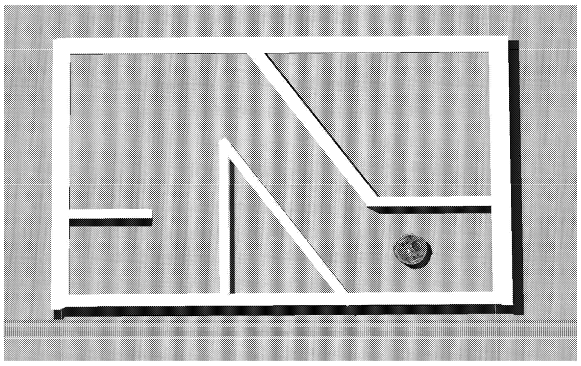
Figure 5: Agent was trained in the simulated environment of size 100 x 60 cm.

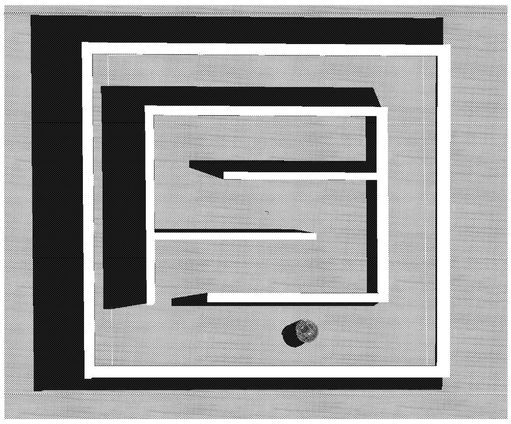
Figure 6: Simulated testing environment of size 110 x 100 cm.

100 x 60 cm and tested in more complex environment of size 110 x 100 cm. We used Webots [3] simulation software. Simulation process consisted of predefined number of steps. In each simulation step agent processed sensor values and set speed to the left and right motor. One simulation step took 32 ms.

6.1 Evolutionary RBF Networks

The evolutionary RBF networks were applied to the maze exploration task. The network input and output values are preprocessed in the same way as for the reinforcement learning.

To stimulate maze exploration, agent is rewarded, when it passes through the zone. The zone is randomly located area, which can not be sensed by an agent. Therefore, Δ_j is 1, if agent passed through the zone in j-th trial and 0 otherwise. The fitness value is then computed as

$$Fitness = \sum_{j=1}^{4}(S_j + \Delta_j), \qquad (10)$$

where quantity S_j is computed by summing normalized trial gains $T_{k,j}$ in each simulation step k and trial j.

$$S_j = \sum_{k=1}^{800} \frac{T_{k,j}}{800}. \tag{11}$$

The three component $T_{k,j}$ motivates agent to move and avoid obstacles.

$$T_{k,j} = V_{k,j}(1 - \sqrt{\Delta V_{k,j}})(1 - i_{k,j}) \tag{12}$$

First component $V_{k,j}$ is computed by summing absolute values of motor speed in k-th simulation step and j-th trial, generating value between 0 and 1. The second component $(1-\sqrt{\Delta V_{k,j}})$ encourages the two wheels to rotate in the same direction. The last component $(1 - i_{k,j})$ supports agent's ability to avoid obstacles. The value $i_{k,j}$ of the most active sensor in k-th simulation step and j-th trial provides a conservative measure of how close the robot is to an object. The closer it is to an object, the higher the measured value in range from 0 to 1. Thus, $T_{k,j}$ is in range from 0 to 1, too.

The experiment was repeated 10 times, each run lasted 200 generations (each generation corresponding to 800 simulation steps). In all cases the successful behavior was found, i.e. the evolved robot was able to explore the whole maze without crashing to the walls. See Fig. 7 for the mean, minimal and maximal fitness over 10 runs.

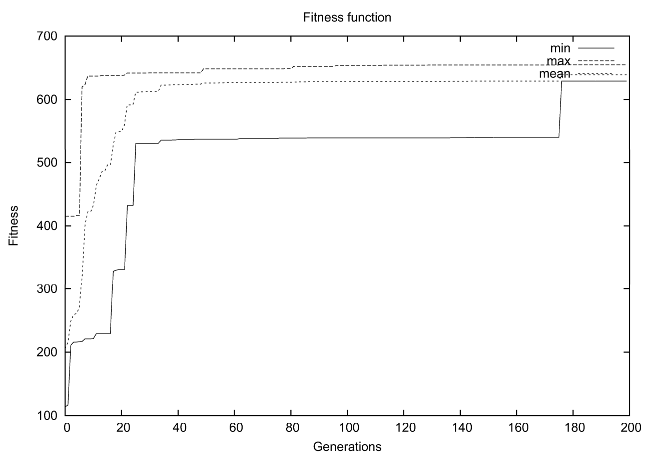

Figure 7: The mean, minimal and maximal fitness function over 10 runs of evolution. Fitness is scaled in a way that successful walk through the whole maze corresponds to the fitness 600 and higher.

Table 2 and Figure 8 show parameters of an evolved network with five RBF units. For the sake of clarity, the parameters listed are also discretized. We can understand them as rules providing mapping from input sensor space to motor control. However, these 'rules' act in accord, since the whole network computes linear sum of the five corresponding gaussians.

	Sensor		**Width**	**Motor**	
left	**front**	**right**		**left**	**right**
VERYNEAR	NEAR	VERYFAR	1.56	500	-100
FEEL	NOWHERE	NOWHERE	1.93	-500	500
NEAR	NEAR	NOWHERE	0.75	500	-500
FEEL	NOWHERE	NEAR	0.29	500	-500
VERYFAR	NOWHERE	NOWHERE	0.16	500	500

Table 2: Rules represented by RBF units (listed values are original RBF network parameters after discretization).

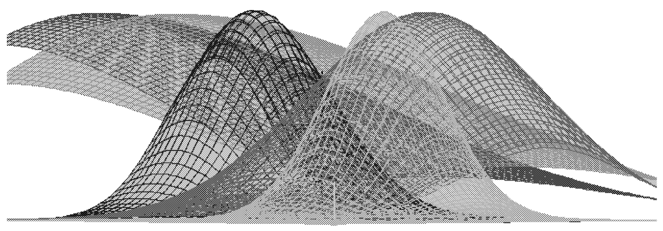

Figure 8: The evolved RBF network (see also Tab. 2). Local units responses plotted in 2D input space corresponding to left and right sensory inputs.

6.2 Reinforcement learning

The same experiment has been performed by means of relational reinforcement learning algorithm described above under the same simulated environment and identical conditions. The performance of the Reinforcement learning agent is shown on figure 9. The graph shows average number of steps from each learning episode. It can be seen that after 10000 episodes, the agent has learned the successful behavior. This number roughly corresponds to the time complexity of the GA, where 200 populations of 50 individuals also result in 10000 simulations. The fitness of the solution found by RL is slightly better than the GA-found solution, on the other hand the inner representation of the neural network is much more compact.

7 Discussion

This article presented survey of popular approaches in mobile robotics used to robot behavior synthesis. In our future work, we would like to design hybrid intelligent system, combining the advantages of these approaches. This way, agent would benefit from using three widely studied fields: Inductive Logic Programming, Neural Networks and Reinforcement Learning. The Reinforcement Learning has strong mathematical background. On the other side, in real experiments, some of the assumptions are not realistic. Neural networks are very popular in robotics, because they provide straightforward mapping from input signals to output signals, several levels of adaptation and are robust to noise. Inductive logic programming allows agent to reason about states, thus concentrating attention on the most promising parts of the state space.

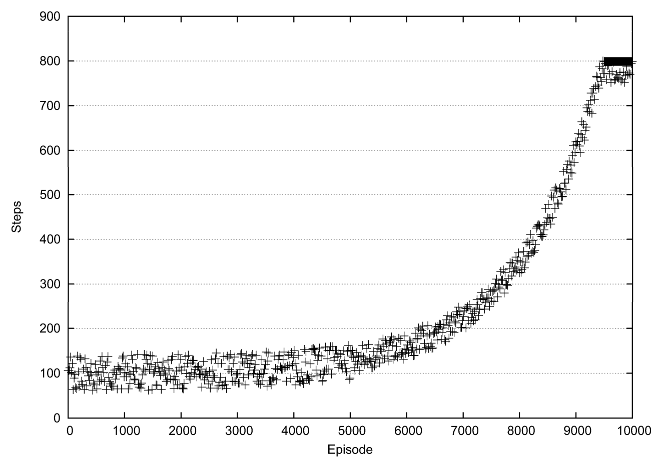

Figure 9: Learning curve for Reinforcement Learning agent averaged on 10 runs.

The experiments showed that a preprocessing plays rather important role in the case of robotic agent control. In our approach we have chosen a rather strong processing of inputs and outputs, which is suitable for RL algorithms mainly. In our future work we would like to study control with less preprocessed inputs/outputs which can be used mainly for the neural network controller. Also, another immediate work is to extract the most frequently used state transitions from the RL algorithm and interpret them as rules in a similar fashion we did with the RBF network.

References

[1] E-puck, online documentation. http://www.e-puck.org.

[2] Khepera II documentation. http://k-team.com.

[3] Webots simulator. http://www.cyberbotics.com/.

[4] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, pages 81–138.

[5] A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. 13:341–379.

[6] R. E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[7] D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic programming*. Ahtena Scientific, 1996.

[8] H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101:285–297, 1998.

[9] H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166.

[10] D.S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[11] S. Dzeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning 43*, pages 7–52, 2001.

[12] D. B. Fogel. *Evolutionary Computation: The Fossil Record.* MIT-IEEE Press, 1998.

[13] J. Holland. *Adaptation In Natural and Artificial Systems*. MIT Press, reprinted edition, 1992.

[14] A. Martinoli. *Swarm intelligence in autonomous Collective robotics: from tools to the analysis and synthesis of distributed control strategies.* Lausanne: Computer Science Department, EPFL, 1999.

[15] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1:289–303, 1989.

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

[17] T. Poggio and F. Girosi. A theory of networks for approximation and learning. Technical report, Cambridge, MA, USA, 1989. A. I. Memo No. 1140, C.B.I.P. Paper No. 31.

[18] S. Slušný and R. Neruda. Evolving homing behaviour for team of robots. *Computational Intelligence, Robotics and Autonomous Systems. Palmerston North : Massey University*, 2007.

[19] S. Slušný, R. Neruda, and P. Vidnerová. Evolution of simple behavior patterns for autonomous robotic agent. *System Science and Simulation in Engineering. - : WSEAS Press*, pages 411–417, 2007.

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[21] C. J. Watkings. *Learning from Delayed Rewards.* PhD thesis, Cambridge University, 1989.