International Journal of Control and Automation
Vol. 2, No. 1, March, 2009

Performance Comparison of Two Reinforcement Learning Algorithms for Small Mobile Robots

Roman Neruda, Stanislav Slušný
Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, Prague 8, Czech Republic
roman@cs.cas.cz

Abstract

The design of intelligent agents by means of reinforcement learning is studied in this paper. A relational reinforcement learning algorithm is used to achieve a compact knowledge representation. Moreover, this approach allows to improve the learning performance by augmenting the algorithm with the so-called background knowledge. A case study on simulated physical robotic agents is performed and compared with our previous evolutionary robotics experiments in order to justify our approach.

1 Introduction

One of the key question of the artificial intelligence (AI) is the design intelligent agents. The majority of approaches so far utilizes one form or another of sophisticated hard-wired models created by humans. Self-adaptation and self-organization are important features of intelligent agents behaviour. These are typically achieved by employing some kind of machine learning algorithm to emerge a control mechanism based on agent interaction with the environment.

In this paper we employ the *reinforcement learning (RL)* algorithm [8] to extract a mapping from sensor space to effector space. reinforcement learning is a widely studied approach in AI used to design intelligent agents. In its classical form, this mapping is represented as a transition matrix which has several disadvantages such as a large size and low comprehensibility of the knowledge representation. Thus, alternative ways of knowledge representations have been sought in order to reduce the complexity and improve the human understanding of the model. Dzeroski [7] proposed an approach, called *relational reinforcement learning*, that combines reinforcement learning with inductive logical programming. In this article, we extend this work, we compare the performance of the algorithm on maze task, in which simulated e-puck robot has to explore the maze in an efficient manner.

2 Reinforcement Learning Problem

The basic setup of reinforcement learning problem is an agent that is situated in an environment, senses the environment, and acts upon it in time. The task of the agent is to optimize its behavior in the following sense. The agent-environment interaction is modeled through the notion of rewards.

Agent's task is to maximize rewards by choosing right actions. In order to distinguish situations that need different action responses, the agent has a set of states available.

In the following, we will work with embodied agent (small miniature robot), that can sense the world by its sensors and act in the world by using its effectors (motors). Further common assumptions about the basic RL setting are that of discrete time steps and Markov property of the environment [8].

Agent's life can be written as a sequence

$$o_0 a_0 r_0 s_1 a_1 r_1 ... \tag{1}$$

where $o_t \in S$ denotes observation, 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. We will assume full observability of the state. With that in mind, we can unify the states with observations ($s_i = o_i$).

The goal of the agent is to find optimal *strategy*. Strategy is a transcript that specifies which actions should agent choose in particular states. The mentioned Markov property of the environment states that the model of the environment (transition probabilities $P^a_{ss'}$ and expected rewards R^a_s) depend only on the last state and a chosen action (and not on the whole history of state-action pairs).

Formally, the *value function* V^π specifies how good the strategy π is. Agent's task is to find an optimal policy π^*, that maximizes the value V^*:

$$\pi^* = argmax_\pi V^\pi \tag{2}$$

The discounted cumulative reward is used as a value of the policy. In this settings, the optimal strategy is deterministic and stationary — it is a mapping from the state space to the action space ($\pi : S \rightarrow A$).

$$V^\pi(s_t) = E_\pi[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...] = E_\pi[\sum_{i=0} \gamma^i r_{t+1}] \tag{3}$$

Historically, the first solution to the above given problem is due to Bellman. He considered settings in which the model of the environment is known (this is called dynamic programming). On the contrary, in the RL, the model of the environment is not known in the advance — agent is just able to interact with it.

A major breakthrough in the RL area was the Q-learning algorithm [10, 2], which is guaranteed to compute optimal strategy under certain conditions (described later).

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

$$Q^\pi(s, a) = R^a_s + \gamma \sum_{s'} P^a_{ss'} Q^\pi(s', a), \tag{4}$$

In order to find the optimal strategy π^*, it is satisfactory to find the optimal Q-values Q^*. In a testing phase, an agent then chooses action that maximizes expected reward:

$$\pi^*(s) = argmax_a Q^*(s, a) \tag{5}$$

The Q-learning algorithm(figure 3) is proven to converge under the usual RL assumptions: every state-action pair is visited infinitely many times, and moreover, the stochastic-approximation conditions hold for step size parameter α. To continuously visit all state-action pairs, we have used

soft-max action selection, which decides (in the training phase), if the agent will take the best action so-far, or it will explore the environment by randomly selected action.

The best action is chosen according to the Boltzmann distribution:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1} e^{Q_t(b)/\tau}} \tag{6}$$

where τ is a temperature parameter, that is decreasing during the training phase.

The Q-learning algorithm updates estimates of Q-values by using the so-called *2f rule* (see algorithm figure 3). As we have experimented only with episodic tasks (when the robot fulfilles the task, the task is repeated with robot moved from its initial position), updates are carried out on the end of episode.

3 Relational Reinforcement Learning

Several improvements have been suggested to speed up the basic RL algorithm. In real life applications, the state space is usually too large and the convergence toward the optimal strategy is slow. In recent years, there have been a lot of efforts devoted to rethinking the idea of states by using function approximators [4], decision trees, or defining the notion of options and hierarchical abstractions [3]. Relational reinforcement learning [7] is an approach that combines RL with inductive logical programming (ILP).

In the classical version of the Q-learning algorithm described above, the Q-values were stored in tables or decision trees. This was not only very space-consuming, but it also made it difficult to aggregate parts of state space, that are similar. In relational RL, the Q-values are stored in a more sophisticated way. We have used the *logical decision trees* as implemented in the programs TILDE [5] from the ACE-ilProlog package [6] in our experiments.

The major improvement in comparison with classical reinforcement learning is more compact representation of knowledge, that speeds up the convergence of the algorithm. More detailed description of our representation will be given in section describing experiments.

Classical decision trees (figure 3) employ propositional or attribute value representations. Recently, however, these representations have been upgraded towards the first order logic. The main difference is that logical decision trees employ Prolog-queries as tests in the internal nodes of the decision trees. The queries can contain variables, and the variables can be shared between nodes.

The key issue when using inductive logic programming is the so-called *background knowledge* which contains definitions of general predicates that can be used in the induced hypotheses. The background knowledge allows the user to influence the learning process and the results in a significant way.

Consider the trees from figure 3. The strategy represented by them is not to choose action forward, if the obstacle is in front of the agent and to go left, if obstacle is on the right (the former tree), and to go right, if obstacle is on the left (the latter tree). This strategy can be described by one logical decision tree on figure 2, if we give the agent the background knowledge about sensor's symmetry. The root of the tree can contain predicates, that help to resolve variables by unification algorithm.

Not only that the induced logical trees are typically smaller than classical decision trees, but also the state space, that has to be explored by the agent, is significantly reduced (due to state aggregation). Two trees in figure 3 correspond to two different situations, that are aggregated into one relational tree in 2. Using our induction strategy and provided background knowledge, an agent

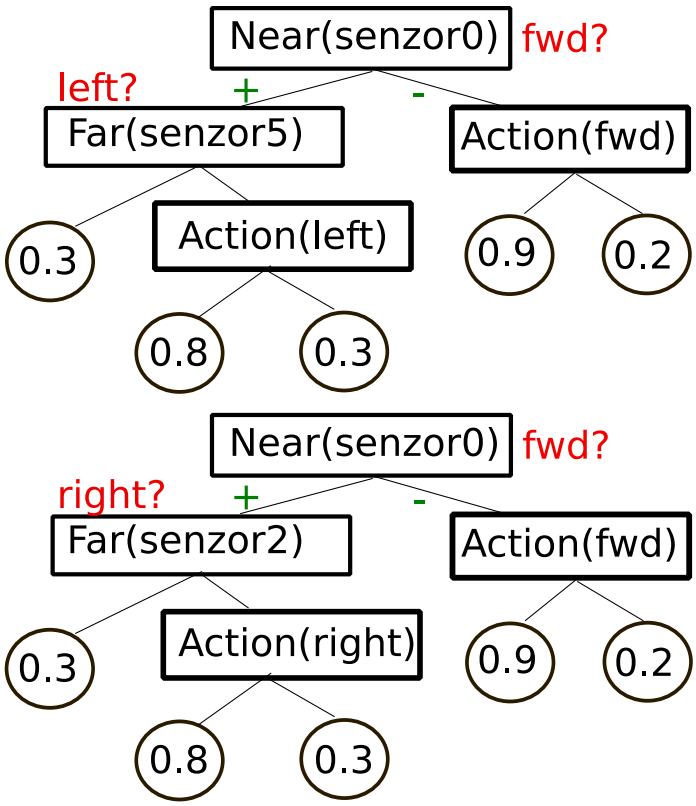


Figure 1. Representation of Q-values using classical decision trees. Q-values are stored in the leaves. The better the action, the higher the Q-value. The former tree stores the information that agent should choose action "LEFT", if the distance measured by sensor0 reads "NEAR", and the distance measured by sensor5 reads "FAR". The latter tree stores a corresponding case for a symmetric situation.

can guess action value without actually exploring the action. This usually leads to worse behavior in the beginning though (agent aggregates all states into one), until the tree is big enough.

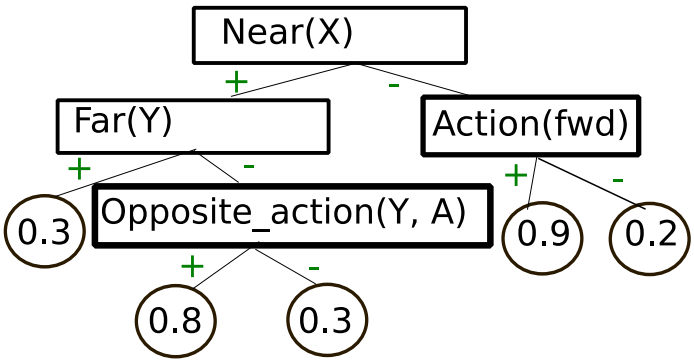
4 Experiments

4.1 Basic setting

In order to compare the performance and properties of the described algorithms, we conducted a set of simulated experiments in which miniature e-puck [1] robots were trained to explore the environment and avoid walls in a maze.

The e-puck is a mobile robot with a diameter of 70 mm and a weight of 50 g. The sensory system employs eight active infrared light sensors distributed around the body, six on one side and two on other side (cf. Fig. 4). The robot sensors can detect a white paper at a maximum distance of approximately 8 cm. The sensors return discrete values from the $[0, 4095]$ intervals, while the effectors accept values from the $[-1000, 1000]$ interval. The higher the absolute value, the faster the motor is moving.

Without any further preprocessing of sensor's and effector's values, the state space would be too



Background knowledge:
OPPOSITE_ACTION_TO_SENSOR(S, A)
:-ISLEFTSENSOR(S), A == RIGHT.
OPPOSITE_ACTION_TO_SENSOR(S, A)
:-ISRIGHTSENSOR(S), A == LEFT.
ISLEFTSENZOR(SENSOR5).
ISLEFTSENZOR(SENSOR6).
ISRIGHTSENZOR(SENSOR2).
ISRIGHTSENZOR(SENSOR3).
Root:
ACTION(A)
Example:
NEAR(SENZOR0), FAR(SENZOR2),
ACTION(RIGHT).

Figure 2. Logical decision tree, created by unifying trees from figure 3. This is the part of the tree, that was built in the early phase of the described experiment.

big. Therefore, instead of raw sensor values, the learning algorithms works with "perceptions". Instead of 4095 raw sensor values, we used only 5 perceptions. 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 100 x 60 cm and tested in more complex environment of size 110 x 100 cm (cf. Fig. 5). The Webots [11] simulation software have been used to carry out the experiments. 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.

4.2 The reward function

The reward function which is the most important part of the task definition has been set to correspond to the fitness function of our previously carried out experiments with evolutionary trained radial basis function networks reported in [9]. The reward function contains several terms which stimulate distinct desired behaviors of the robot.

To stimulate maze exploration, the agent is rewarded when it passes through the zone. The zone is a randomly located area which can not be sensed by an agent. Therefore, Δ_j is 1, if agent passed

1. for each s, a do
   (a) initialize the table entry $Q'(s, a) = 0$
   (b) $e = 0$
2. do forever
   (a) $e = e + 1$
   (b) $i = 0$
   (c) generate a random state s_0
   (d) while not goal(s_i) do
      i. select an action a_i and execute it
      ii. receive an immediate reward $r_i = r(s_i, a_i)$
      iii. observe the new state s_{i+1}
      iv. $i = i + 1$
   (e) endwhile
   (f) for $j = i - 1$ to 0 do
      i. update
      $Q'(s_j, a_j) = r_j + \alpha_k \max_{a'} Q'(s_{j+1}, a')$

Figure 3. Scheme of the Q-learning algorithm [7].

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.

through the zone in j-th trial and 0 otherwise. The fitness value is then computed as

$$Fitness = \sum_{j=1}^{4} (S_j + \Delta_j), \tag{7}$$

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

$$S_j = \sum_{k=1}^{800} \frac{T_{k,j}}{800}. \tag{8}$$

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

$$T_{k,j} = V_{k,j}(1 - \sqrt{\Delta V_{k,j}})(1 - i_{k,j}) \tag{9}$$

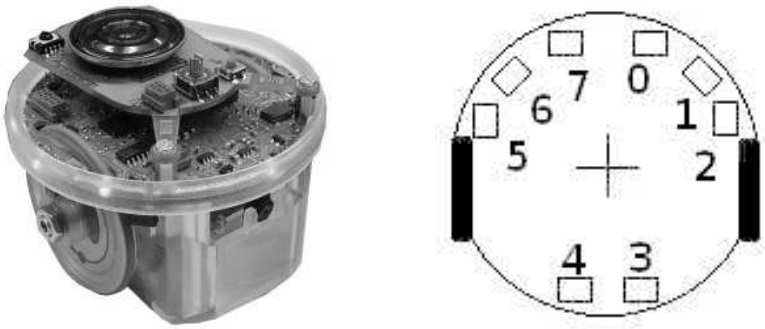


Figure 4. Miniature e-puck robot has eight infrared sensors and two motors.

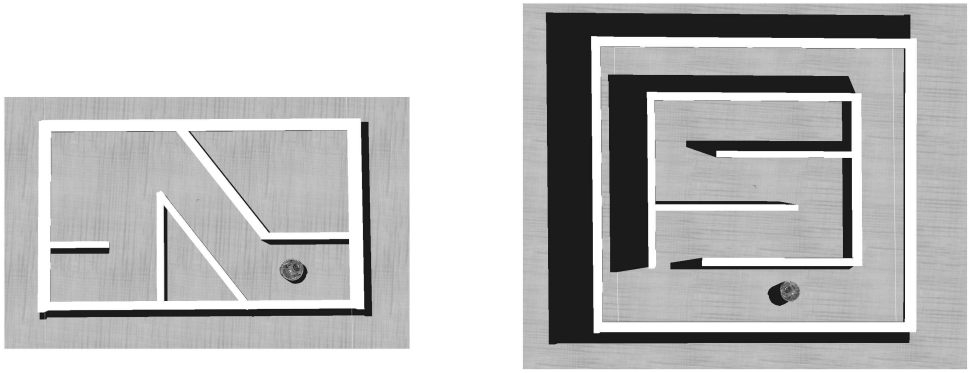


Figure 5. a) The agent was trained in the simulated environment of size 100 x 60 cm. b) The simulated testing environment was of size 110 x 100 cm.

The 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.

4.3 Results

The performance of the reinforcement learning agents is shown on figure 6. 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 evolutionary algorithm in [9], where 200 populations of 50 individuals also result in 10000 simulations. The fitness of the solution found by RL is slightly better than the solution found by EA, on the other hand the inner representation of the neural network is much more compact then the Q-values table of the RL.

Usage of the logical trees in the relational RL algorithm has a great impact on performance, as shown on Fig. 6. In the relational case of algorithm, it's satisfactory for the agent to learn knowledge about left sensors. The rest can be derived from the background knowledge. Therefore,

the convergence is faster in this case.

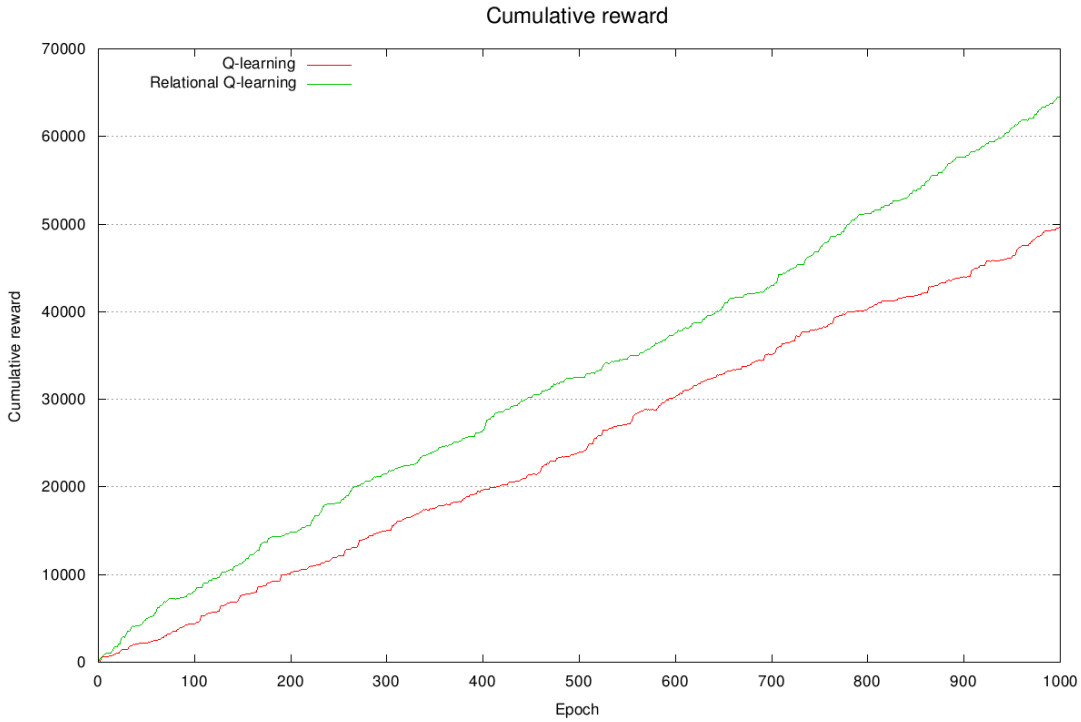


Figure 6. Comparison of Q-learning and Relational Q-learning algorithm. After the training, which took 10000 epochs, agent was tested for another 1000 epochs. Average of 10 runs.

Sensors			
Left	Front	Right	Motors
VERY NEAR	NEAR	VERY FAR	RIGHT
FEEL	NOWHERE	NOWHERE	SLOWLY RIGHT
NEAR	NEAR	NOWHERE	RIGHT
FEEL	NOWHERE	NEAR	FORWARD
VERY FAR	NOWHERE	NOWHERE	FORWARD

Table 2. Rules learned by the RL algorithm.

Tables 2 and 3 show symbolic representation of the rules induced from the relational RL and RBF network models (the RBF network has five hidden units in this case). For the sake of clarity, all the parameters listed are discretized. We can understand them as rules providing mapping from input sensor space to motor control. However, in the RBF network case, these 'rules' act in accord, since the whole network computes linear sum of the five corresponding gaussians which can result in more complicated behavior.

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

Sensors			
Left	Front	Right	Motors
VERY NEAR	NEAR	VERY FAR	RIGHT
FEEL	NOWHERE	NOWHERE	LEFT
NEAR	NEAR	NOWHERE	RIGHT
FEEL	NOWHERE	NEAR	RIGHT
VERY FAR	NOWHERE	NOWHERE	FORWARD

Table 3. Rules learned by the Radial basis function networks.

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.

5 Conclusion

In this article, we have shown, how appropriate knowledge representation can speed up learning process. We have proposed relational version of Q-learning algorithm, that combines reactive decision making with background planning and brings logic programming into behavior-based robotics approach. The performance of the algorithm has been evaluated on the simple maze task. In our previous work [9], we used evolutionary algorithms with RBF neural networks to solve the same task.

The experiments show 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 the basic RL algorithm 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 or more sophisticated flavours of the RL.

The comparison of the three approaches — classical RL, relational RL, and RBF networks — indicated that although they were all able to learn the given task, it is advisable to use as much of a background knowledge, as possible in order to speed up the learning and achieve higher scalability of the approaches. The hybrid approaches combining formally sound knowledge representation with strong heuristics seem the most promising direction of further work.

The natural representation of knowledge makes it possible to simply transfer knowledge to another robot. It's very simple to add prior knowledge to the task and the learned knowledge can be easily extended to the another domains. In the future, we would like to investigate knowledge transfer across domains and agents in multi-agent systems.

Acknowledgements

This work has been supported by the Ministry of Education of the Czech Republic under the project Center of Applied Cybernetics No. 1M684077004 (1M0567). Stanislav Slusny has been partially supported by project no. GAUK 7637/2007 of Charles University Grant Agency.

References

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

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

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

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

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

[6] 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.

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

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

[9] Stanislav Slušný, Roman Neruda, and Petra Vidnerová. Comparison of RBF network learning and reinforcement learning on the maze exploration problem. *Artificial Neural Networks - ICANN 2008*, pages 720–729, 2008.

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

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