Bi-Memory Model for Guiding Exploration by Pre-existing Knowledge

Kary Främling KARY.FRAMLING@HUT.FI
Helsinki University of Technology, P.O. Box 5500, FI-02015 TKK, Finland.

Abstract

Reinforcement learning agents explore their environment in order to collect reward that allows them to learn what actions are good or bad in what situations. The exploration is performed using a policy that has to keep a balance between getting more information about the environment and exploiting what is already known about it. This paper presents a method for guiding exploration by pre-existing knowledge expressed by e.g. heuristic rules. A dual memory model is used where the value function is stored in long-term memory while the heuristic rules for guiding exploration act on the weights in a short-term memory. Experimental results from two "toy domains" illustrate that exploration is significantly improved when guidance can be provided by pre-existing knowledge.

1. Introduction

In supervised learning a "teacher" provides a set of training samples that have usually been pre-processed in a way that simplifies learning. Reinforcement learning (RL) differs from supervised learning mainly because the RL agent has to explore its environment by itself and collect training samples. The agent takes actions following a policy and observes received reward that it attempts to maximize. Except for simple tasks, this exploration can be long or even infeasible without any guidance. Different ways of providing such guidance has been studied by several researchers, e.g. Schaal (1997), Millán et al. (2002) and Driessens & Džeroski (2004).

This paper presents a method for exploring the environment using pre-existing knowledge expressed e.g. by heuristic rules. It uses a dual memory model where a so-called long-term memory is used for action-value learning and a so-called short-term memory is used for guiding action selection by heuristic rules. Experimental results with simple heuristic rules illustrate that it is possible to converge to a good policy with less exploration than with some well-known methods. This paper concentrates on episodic tasks, i.e. tasks with a pre-defined terminal state, even though the methods are not limited to such tasks.

After this introduction, the most relevant RL methods to the scope of this paper are described in Section 2. Section 3 presents methods to shorten initial exploration and how to combine them with methods presented in Section 2. Section 4 shows comparative results for three different tasks, followed by conclusions.

2. Reinforcement learning principles

Most existing RL methods try to learn a *value function* that allows them to predict the sum of future reward from any state s when following a given action selection policy π. Value functions are either *state-values* (value of a state) or *action-values* (value of an action in a given state). Action-values are denoted $Q(s,a)$, where a is an action. The currently most popular RL methods are so-called *temporal difference* (TD) methods (Sutton, 1988). *Q-learning* (Watkins, 1989) is a TD control algorithm that updates action-values according to

$$\Delta Q(s_t,a_t) = \beta\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right] e_{t+1}(s,a) \quad (1)$$

where $Q(s_t,a_t)$ is the value of action a in state s at time t, β is a learning rate, r_{t+1} is the immediate reward and γ is a discount factor. The max-operator signifies the greatest action-value in state s_{t+1}. $e_{t+1}(s,a)$ is an eligibility trace that allows rewards to be propagated back to preceding states and actions. A replacing eligibility trace (Singh & Sutton, 1996) is calculated according to

$$e_{t+1}(s) = \begin{cases} \gamma\lambda e_t(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases}, \quad (2)$$

where λ is a *trace decay* parameter that together with the γ determines how quickly rewards are propagated backwards. Another common method for action-value learning is SARSA (Rummery & Niranjan, 1994):

$$\Delta Q(s_t,a_t) = \beta[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]\ e_{t+1}(s,a) \quad (3)$$

The most complete overviews available on methods for state-space exploration are (Thrun, 1992; Wiering, 1999).

Appearing in *Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning*, Bonn, Germany, 2005.

Commonly used *undirected exploration* methods that do not use any task-specific information are *ε-greedy exploration* and *Boltzmann* action selection. ε-greedy exploration selects the greedy action with probability (1-ε) and an arbitrary action with probability ε using a uniform probability distribution. *Directed exploration* methods use task-specific knowledge for guiding exploration in such a way that the state space would be explored more efficiently. The action to be taken is selected by maximizing an evaluation function that combines action-values with *exploration bonus*es δ_n weighted by factors K_0 to K_k (Ratitch & Precup, 2003):

$$N(s,a) = K_0 Q(s,a) + K_1 \delta_1(s,a) + ... + K_k \delta_k(s,a) \qquad (4)$$

Counter-based methods use an exploration bonus that directs exploration to less frequently visited states. *Recency-based exploration* prefers least recently visited states. Other directed exploration methods exist that use statistics on value function variance or other indicators. The same effect of preferring unexplored actions (and states) mainly in the beginning of exploration can be achieved by a technique called *optimistic initial values* that uses initial value function estimates that are bigger than the expected ones. A common implementation of this technique is to initialize action values to zero and give negative reward at every step (Thrun, 1992, p. 8). This means that unused actions have greater value estimates than used ones, so unused actions have a greater probability to be selected in the beginning of exploration.

3. SLAP reinforcement and the BIMM network

This section describes the use of a *long-term memory* (*LTM*) and *short-term memory* (*STM*)[1] model for combining action-value learning with heuristic rules that guide exploration (Främling, 2003). LTM is used for action-value learning while STM learning is used for guiding state-space exploration by the *SLAP* (Set Lower Action Priority) principle, described in sub-section 3.2. In sub-section 3.3 we study the effect of different learning parameters on the trade-off between rapid exploration and converging towards a "good" policy.

3.1 Bi-Memory Model (*BIMM*)

The bi-memory model uses a short-term memory for controlling exploration and a long-term memory for learning the value function. Both memories are here implemented as linear function approximators or Adalines (Widrow & Hoff, 1960), but any function approximator may be used for both STM and LTM. A linear function approximator calculates action values as the weighted sum of action neuron input values

$$a_j(s) = \sum_{i=1}^{N} s_i w_{i,j} \qquad (5)$$

where s_i is the value of state variable i, $w_{i,j}$ is the weight of action neuron j for input i, a_j is the output value of action neuron j and N is the number of state variables. Weights are typically stored in a two-dimensional matrix of size $M \times N$, where M is the number of actions. This representation is identical to the lookup-table representation usually used in discrete RL tasks but gives the advantage of being able to handle continuous-valued state variables directly. Adalines can be trained using the Widrow-Hoff training rule

$$w_{i,j}^{new} = w_{i,j} + \alpha(a_j' - a_j)s_i \qquad (6)$$

where a_j' is the "target" value used in supervised learning. α is a learning rate parameter that determines the step size of weight modifications. Widrow-Hoff learning is a gradient descent method that minimizes the root mean square error (RMSE) between a_j' and a_j samples. When using BIMM, Adaline outputs are calculated according to

$$a_j(s) = K_0 \sum_{i=1}^{N} ltw_{i,j} s_i + K_1 \sum_{i=1}^{N} stw_{i,j} s_i \qquad (7)$$

where K_0 and K_1 are positive constants that control the balance between exploration and exploitation. STM is actually an exploration bonus whose influence on action selection is determined by the value of K_1 as in equation (4). $ltw_{i,j}$ is the LTM weight and $stw_{i,j}$ is the STM weight for action neuron j and input i. $a_j(s)$ is the estimated action-value $N(s,a)$ in equation (4). Both Q-learning and SARSA can be used to update LTM weights by replacing Q with ltw in equations (1) and (3).

3.2 Guiding exploration

Exploration bonuses affect action selection by increasing or decreasing the probability of an action being selected in a given state. If an action does not seem to be useful for exploration in some state according to some pre-existing knowledge, then make it less likely for that action to be used in that state. Similarly, if an action seems to be good in some state according to current rules, then make it more likely to be used in that state. In the tests performed in this paper only the first case is used, i.e. action probabilities are only decreased by the set lower action priority (SLAP) principle, where STM weights are updated using the Widrow-Hoff update rule with the target value

$$a_j'(s) = a_{min}(s) - margin \qquad (8)$$

where $a_{min}(s)$ is the smallest $a_j(s)$ value in state s. The *margin* should have a "small" value (0.1 has been used in

[1] "Short-term memory" has been used in different contexts, e.g. for storing the eligibility trace; for memorizing parts of the state history in order to improve identification of "hidden states" (McCallum, 1995); and for storing contextual clues to be used in near-future states (Bakker 2002). These differ from the STM used here.

all tests reported in this paper), which ensures that an action that is repeatedly SLAPed will eventually have the lowest action value. Only STM weights are modified by the Widrow-Hoff rule, which becomes

$$stw_{i,j}^{new} = stw_{i,j} + \alpha(a_j' - a_j)s_i \tag{9}$$

The new activation value is then

$$\begin{aligned} a_j^{new}(s) &= K_0\sum_{i=1}^{N} ltw_{i,j}s_i + K_1\sum_{i=1}^{N} stw_{i,j}^{new}s_i = \\ K_0\sum_{i=1}^{N} ltw_{i,j}s_i &+ K_1\sum_{i=1}^{N} s_i\left(stw_{i,j} + \alpha(a_j' - a_j)s_i\right) = \\ K_0\sum_{i=1}^{N} ltw_{i,j}s_i &+ K_1\sum_{i=1}^{N} stw_{i,j}s_i + \alpha(a_j' - a_j)K_1\sum_{i=1}^{N} s_i^2 = \\ a_j &+ \alpha(a_j' - a_j)K_1\sum_{i=1}^{N} s_i^2 \end{aligned} \tag{10}$$

where we can see that setting α to $1/(K_1\Sigma s_i^2)$ guarantees that a_j^{new} will become a_j' in state s after SLAPing action j. Replacing α with $\alpha/(K_1\Sigma s_i^2)$ in equation (9) gives a generalization for BIMM of the well-known Normalized Least Mean Squares (NLMS) method

$$stw_{i,j}^{new} = stw_{i,j} + \alpha(a_j' - a_j)s_i \Big/ K_1\sum_{i=1}^{N} s_i^2 \tag{11}$$

that reduces the error $(a_j' - a_j)$ exactly by the ratio given by α, which makes it easier to select a good value for α. For instance, if $\alpha = 1$ in equation (11) the SLAPed action will directly have the lowest $a_j(s)$ value for state s so it will not be used again until all other possible actions have been tested in the same state. This is especially useful in deterministic tasks. In stochastic tasks α should be inferior to one because even the optimal action may not always be successful, so immediately making its action-value the lowest would not be a good idea. As long as the value of $a_{min}(s)$ doesn't change, the value $a_j'(s)$ remains the same for all j. This is true until $a_j^{new}(s)$ becomes lower than the current $a_{min}(s)$ for some j. Therefore, action values slowly go towards minus infinity when using SLAP an infinite number of times[2].

A general algorithm for using SLAP in a learning task is given in Figure 1. Especially when using SARSA or other on-policy methods for action-value updates, STM weights should be updated before updating the action-values, otherwise the changes in STM weights might modify the action selection for the next state. It is also worth pointing out that the SLAP update rules do not include a time variable t as for Q-learning and SARSA, so SLAP can be used asynchronously with action-value updates. In fact, SLAP can be used for any state-action pair at any time independently of the current state and the current action selected.

```
Initialize parameters
REPEAT (for each episode)
  s ← initial state of episode
  REPEAT (for each step in episode)
    a ← action given by π for s
    Take action a, observe next state s'
    SLAP "undesired" actions
    Update action-value function in LTM
    s ← s'
```

Figure 1. General algorithm for using SLAP in typical RL task.

3.3 Increasing exploration

As shown by the experimental tasks in section 4, it can be easy to identify heuristic rules that make exploration faster. However, if these rules do not also give sufficient exploration of the state space, then it has to be provided by other means. STM weights are reinitialized before starting a new episode and can have a great impact on the way in which the state space is explored. Initializing STM weights to random values and using a "high" value for K_1 is one way of increasing exploration. In all tests reported here, STM weights have been initialized to random values in the interval [0,1) while LTM weights are initialized to zero. Therefore actions will initially be selected in a random order independently of the value of K_1 in equation (7) as long as LTM weights remain zero. When LTM values become non-zero due to action-value learning, the amount of randomness in action selection depends both on STM and LTM weights.

Undirected exploration methods (e.g. ε-greedy, Boltzmann) can also be used to increase exploration. Driessens and Džeroski (2004) alternated guided and unguided episodes for the same purpose, where hand-coded rules, human operators or other pre-existing knowledge provided guidance.

4. Experimental results

This section compares methods presented in the previous sections on two different tasks: 1) semi-MDP maze world and 2) mountain-car. Exploration methods compared are: **1) *Q/SARSA***: ε-greedy exploration; **2) *CTRB***: counter-based exploration **3) *OIV***: optimistic initial values and **4) *BIMM***. Q/SARSA, CTRB and BIMM use zero initial Q-values, $r = 1$ at terminal state and $r = 0$ for all other states. OIV used zero initial Q-values, $r = 0$ at goal and $r = -1$ for all other state transitions. Constant learning parameters are used in order to simplify the choice of parameter values and for reasons of comparability.

[2] Setting $a_j'(s) = a_{max}(s) + margin$ would increase weights in the same way that SLAP decreases them but this has not been useful for the experimental tasks in this paper.

4.1 Maze with transition delays

Semi-Markov Decision Processes (SMDP) may include continuous time (Bradtke and Duff, 1995). This signifies that the transition time from one state to another depends on a probability distribution F_{xy}. Here we introduce state transition time by the notion of "corridors" in a maze, i.e. states with only two actions that represent opposite directions. When in a corridor, the agent continues forward until it reaches a non-corridor state. The maze world is of size 20x20 (Figure 2), where 10 doors have been opened in addition to the initial unique solution. Both deterministic and stochastic state transitions are used. The stochastic state transition rates used are 0.2 and 0.5, which indicate the probability of another direction being taken than the intended one.

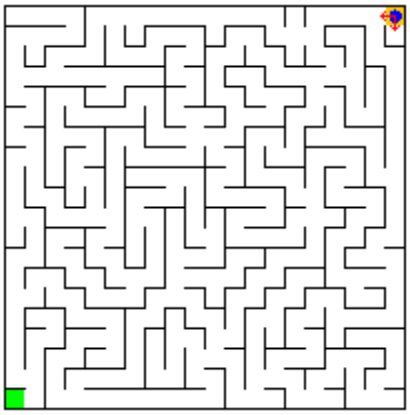

Figure 2. Maze with 10 supplementary "doors" opened in the walls in addition to the initial unique route. Agent in start position, goal position in lower left corner.

Q-learning without eligibility trace is used for action-value learning. Learning parameters are indicated in Table 1. The counter-based exploration bonus in equation (4) is implemented as

$$\delta_1(s,a) = \begin{cases} 1 - \dfrac{cnt(s,a) - cnt(s)_{\min}}{cnt(s)_{\max} - cnt(s)_{\min}} & \text{if } cnt(s)_{\max} - cnt(s)_{\min} > 0 \\ 0 & \text{if } cnt(s)_{\max} - cnt(s)_{\min} = 0 \end{cases} \quad (12)$$

where $cnt(s,a)$ is the counter for action a in state s and $cnt(s)_{min}$ and $cnt(s)_{max}$ are the smallest and greatest counter values for actions in state s. Counter values are reset after every episode. For BIMM agents, SLAP was used according to the following rules when entering a new state and before performing the next action: 1) SLAP the "inverse" action and 2) if the new state is already visited during the episode, SLAP action with the biggest value $a_j(s)$ in equation (7) for the new state. The rules are applied in the order indicated, so with $\alpha = 1$ the action taken at the previous visit becomes the last one in the action ranking given by equation (7).

All agents performed 250 episodes. Actions were selected greedily after 200 episodes in order to compare how well the value function was learned by all methods on an equal basis. This signifies that ε was set to zero for all agents as well as K_1 for BIMM and CTRB. Figure 3 shows that BIMM converges towards a good policy after much less exploration than Q- and CTRB-agents. The BIMM graph is close to the graph of the OIV agent but OIV converges very slowly. The greater the stochastic state transition rate, the slower the OIV agent converges. This is probably due to the cycles that occur with stochastic state transitions, which cause negative reward to be given even to the optimal action.

Table 1. Parameter values used in grid world tests. For BIMM $K_1 = 0.1$ in deterministic task and $K_1 = 10^{-6}$ in stochastic tasks. $\gamma = 0.95$ for all except OIV. Not indicated parameters are zero.

Agent	***Q***		***CTRB***		***OIV***		***BIMM***	
Grid world	β	ε	β	K_1	β	γ	α	β
Deterministic	1	0.1	1	0.1	1	1	1	1
Stoch. 0.2	0.1	0.1	0.5	0.01	0.5	0.95	0.2	0.5
Stoch. 0.5	0.1	0.1	0.5	0.001	0.5	0.95	0.1	0.5

Table 2 gives numeric comparisons for the performance of the four methods. BIMM agents achieve a better stable policy than the others in all tests as indicated by the third column. The total number of steps is also clearly lower than for the other agents. Since the training parameters often represent a compromise between how good the converged policy is and how much exploration is needed, the fact that BIMM has the best performance in both indicate its superiority in this task. Also, even though the advantage of BIMM in the beginning of exploration decreases with an increasing stochastic state transition rate (second column of Table 2), this is compensated by an improved "converged" policy compared with the other agents.

Table 2. Results for 20x20 maze with ten extra doors in the order Q/CTRB/OIV/BIMM. The third column indicates the average number of steps for the "stable" policy as average value of episodes 241-250.

Stoch. trans. rate	**Steps on first episode**	**Steps with converged policy**	**Total number of steps**
Deterministic	7450/4650/1760/**502**	49.0/**48.0**/**48.0**/**48.0**	134000/62800/27800/**21400**
0.2	7300/6200/2100/**1420**	64.4/69.1/69.2/**60.3**	145000/96000/57400/**41600**
0.5	9980/7840/**3900**/4170	116/196/132/**102**	152000/168000/105000/**103000**

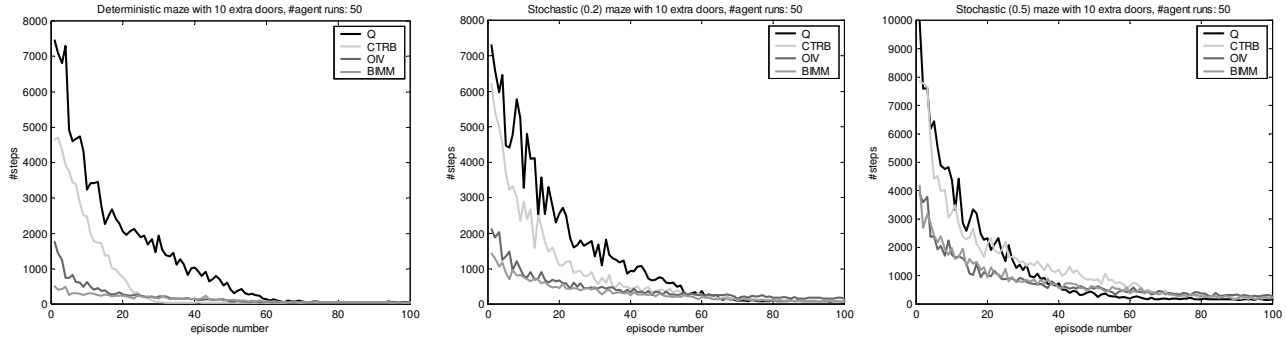

Figure 3. Maze results as average number of steps per episode. Grid size, stochastic transition rate and the number of agent runs used for calculating the average number of steps is indicated at the top of each graph.

4.2 Mountain-Car

The description of this task is similar to that in (Singh & Sutton, 1996) and (Randløv, 2000). The mountain-car task has two continuous state variables: the position x_t and the velocity v_t. At the beginning of each trial these are initialized randomly, uniformly from the range $x \in$ [-1.2,0.5] and $v \in$ [-0.07,0.07]. The altitude is $\sin(3x)$. The agent chooses from actions $a_t \in \{+1,0,-1\}$ that correspond to forward thrust, no thrust and reverse. The physics of the task are:

$$v_{t+1} = \text{bound}\left(v_t + 0.001a_t + g\cos(3x_t)\right)$$

and

$$x_{t+1} = \max\{x_t + v_{t+1}, -1.2\}$$

where g = 0.0025 is the force of gravity and the bound operation places the variable within its allowed range. If x_{t+1} is clipped by the max-operator, then v_{t+1} is reset to 0. The terminal state is any position with $x_{t+1} > 0.5$. The continuous state space is discretized by 8 non-overlapping intervals for each variable, which gives a total of 64 states. Episodes were limited to 1 000 000 steps.

As in most previous work on this task, the SARSA(λ) learning algorithm with replacing eligibility traces was used for action-value learning. Parameter values are: **SARSA**: learning rate $\beta = 0.1$, discount rate $\gamma = 0.9$, $\lambda = 0.95$, $\varepsilon = 0.1$; **OIV**: learning rate $\beta = 0.1$, discount rate $\gamma = 1.0$, $\lambda = 0.9$; and **BIMM**: learning rate $\beta = 0.1$, discount rate $\gamma = 0.9$, $\lambda = 0.95$, $K_1 = 0.1$, $\alpha = 1.0$. The counter-based method was not used in this task. With the continuous-valued state-variables it often takes several steps before changing state in the discretized state space, so an ordinary counter-based exploration changes the used action too rapidly to allow the agent to explore efficiently. This is also true for the heuristic rules used by BIMM in the maze task. It turns out that it is difficult to find good heuristics for the mountain-car task, as noticed by Randløv (2000) who tried to find a good reward shaping function.

The heuristic rules for using SLAP are 1) SLAP if sign of velocity is different from the sign of the action's thrust and 2) SLAP if velocity is positive and action is zero thrust. The second rule makes exploration slightly faster, but is not of much practical significance. These rules actually implement a controller that is at least nearly optimal for the task at hand, so one could ask what is the point of using a learning controller if we already have a good one? First of all, the initial controller is a static closed-loop controller while learning will give an adaptive open-loop controller that may be able to compensate for errors or calibration drift in real-world applications (see e.g. Främling (2004), for such an adaptive controller and Millán et al. (2002), where rules are used together with a learning controller). The initial controller may also be sub-optimal and incomplete, i.e. not cover the whole state space. Finally, the goal of this paper is to show that the BIMM and SLAP can be used for guiding exploration rather than evaluate the goodness of the heuristic rules.

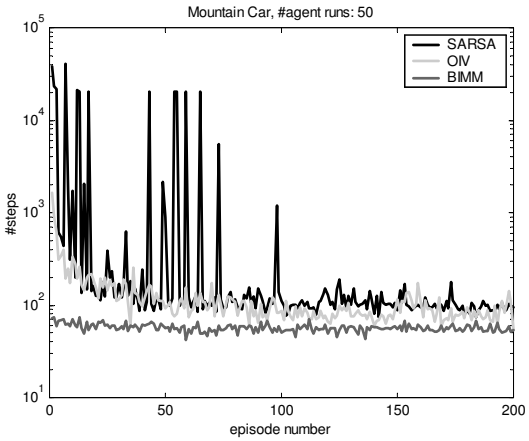

Figure 4. Average number of steps as a function of episode from 50 runs. The peaks are due to infinite episodes, limited to 1 000 000 steps. Note the logarithmical scale on the y-axis.

The results in Figure 4 are consistent with those in (Singh & Sutton, 1996) and (Randløv, 2000). SARSA uses an average of 38000 steps for the first episode, OIV uses 1700 steps and BIMM uses 73 steps. The simulations were run for 10000 episodes with greedy exploration from 9000 episodes onwards. The average numbers of steps for

the last 100 episodes (9901-10000) were 81.2 for SARSA, 78.6 for OIV and 55.6 for BIMM. The total numbers of steps for episodes 1 - 10000 are 1 140 000 for SARSA, 830 000 for OIV and 554 000 for BIMM so the BIMM agent clearly learned the action-value function better and with less exploration.

5. Conclusions

The results show that applying pre-existing knowledge through heuristic rules and the SLAP and BIMM mechanisms can make exploration more efficient both in deterministic and stochastic tasks, as well as in tasks involving continuous-valued state variables. For the heuristic rules used here, it is also apparent that benefits in exploration do not reduce the probability of learning a good value function. One big difference between BIMM and existing methods for improving exploration is that BIMM agents only use their own internal information about the task at hand. This makes them interesting compared with methods like reward shaping, which usually require some a priori knowledge about the environment, such as where the goal is located, the stochastic level of the environment or the number of sub-goals to reach.

Even though only "toy tasks" are used in this paper, it should be possible to generalize the results to many other RL tasks. This should be the case especially for tasks where state generalization is necessary due the number of states. Since SLAP and BIMM use standard ANN structures and learning rules, they are also applicable to tasks involving continuous-valued state variables and state classifiers. Such tasks are a subject of current and future research, where explosion of the state-space size due to state variable discretization is a problem.

References

Bakker, B. (2002). Reinforcement Learning with Long Short-Term Memory. In T. G. Dietterich, S. Becker, and Z. Ghahramani (eds.), *Advances in Neural Information Processing Systems 14*, MIT Press, Cambridge, MA. 1475-1482.

Bradtke, S.J., Duff, M.O. (1995). Reinforcement Learning Methods for Continuous-Time Markov Decision Problems. In G. Tesauro, D. Touretzky, T. Leen, (eds.), *Advances in Neural Information Processing Systems 7*, Morgan-Kaufmann. 393-400.

Driessens, K., Džeroski, S. (2004). Integrating Guidance into Relational Reinforcement Learning. *Machine Learning*, Vol. 57. 271–304.

Främling, K. (2003). *Guiding Initial State-space Exploration by Action Ranking and Episodic Memory*. Laboratory of Information Processing Science Series B, TKO-B 152/03, Helsinki University of Technology. http://www.cs.hut.fi/Publications/Reports/B152.pdf.

Främling, K. (2004). Scaled gradient descent learning rate - Reinforcement learning with light-seeking robot. *Proceedings of ICINCO'2004 conference*, 25-28 August 2004, Setúbal, Spain. 3-11.

McCallum, A. R. (1995). Instance-Based State Identification for Reinforcement Learning. In G. Tesauro, D. Touretzky, T. Leen (eds.) *Advances in Neural Information Processing Systems 7*, MIT Press, 1995. 377-384.

Millán, J.R., Posenato, D., Dedieu, E. (2002). Continuous-Action Q-Learning. *Machine Learning*, Vol. 49. 247-265.

Randløv, J. (2000). Shaping in Reinforcement Learning by Changing the Physics of the Problem. In *Proc. of ICML-2000 conference*. 767-774.

Ratitch, B., Precup, D. (2003). Using MDP Characteristics to Guide Exploration in Reinforcement Learning. In: *Lecture Notes in Computer Science*, Vol. 2837 (Proceedings of ECML-2003 Conference), Springer-Verlag, Heidelberg. 313-324.

Rummery, G. A., Niranjan, M. (1994). *On-Line Q-Learning Using Connectionist Systems*. Tech. Rep. CUED/F-INFENG/TR 166, Cambridge Univ. Engineering Department. 20 p.

Schaal, S. (1997). Learning from demonstration. In M. Mozer, M. Jordan, T. Petsche (eds), *Advances in Neural Information Processing Systems 9*, MIT Press. 1040-1046.

Singh, S.P., Sutton, R.S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, Vol. 22. 123-158.

Sutton, R.S. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, Vol. 3. 9-44.

Thrun, S.B. (1992). The role of exploration in learning control. In DA White & DA Sofge, (eds.), *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold, New York.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. thesis, Cambridge University.

Widrow, B., Hoff, M.E. (1960). Adaptive switching circuits. *1960 WESCON Convention record Part IV*, Institute of Radio Engineers, New York. 96-104.

Wiering, M. (1999). *Explorations in Efficient Reinforcement Learning*. Ph.D. thesis, University of Amsterdam. 218 p.