GUIDING EXPLORATION BY PRE-EXISTING KNOWLEDGE WITHOUT MODIFYING REWARD

Kary Främling

Helsinki University of Technology,
P.O. Box 5500, FIN-02015 HUT, Finland.

Tel.: +358-50-5980451

Telefax: +358-9-451 3665

Email: Kary.Framling@hut.fi.

Abstract

Reinforcement learning is based on exploration of the environment and receiving reward that indicates which actions taken by the agent are good and which ones are bad. In many applications receiving even the first reward may require long exploration, during which the agent has no information about its progress.

This paper presents an approach that makes it possible to use pre-existing knowledge about the task for guiding exploration through the state space. Concepts of short- and long-term memory combine guidance by pre-existing knowledge with reinforcement learning methods for value function estimation in order to make learning faster while allowing the agent to converge towards a good policy.

Keywords: reinforcement learning, exploration, pre-existing knowledge, short-term memory, action ranking

List of symbols

α	Learning rate for weight updates in STM
β	Learning rate for value function updates (action-values in this paper) in LTM
δ_n	Exploration bonus term (n-th term)
ε	Probability of random action selection when using ε-greedy exploration
γ	Discount factor
λ	Trace decay factor of eligibility trace
π	Action selection policy
τ	Temperature of Boltzmann action selection policy
A	Finite set of actions
a	Action belonging to A
a_j	Output value of neuron j
$a_{min}(s)$	Minimum output value for state s
$a_j'(s)$	Target value for action j in state s for STM weight updates
e(s,a)	Eligibility trace value of state s, action a
F_n	Reward shaping function (n-th function)
K_n	Weight of exploration bonus n
LTM	Long-term memory
$ltw_{i,j}$	LTM weight of state variable i and action j

M	Number of actions
margin	Used for ensuring that repeated "SLAP" operations end up by making the "SLAPed" action to get the lowest rank in state s
N	Number of state variables
$Q(s,a)$	Action-value of action a in state s
R	Reward function
r_{t+k+1}	Reward obtained when arriving into states s_{t+1}, s_{t+2} etc.
S	Finite set of states
s	State belonging to S, described by values of state variables s_i. In lookup-table tasks only one s_i has the value one while the others are zero.
s_i	State variable
STM	Short-term memory
$stw_{i,j}$	STM weight of state variable i and action j
T	Next-state transition probabilities
$V^{\pi}(s)$	State value that corresponds to the expected return when starting in s and following the action selection policy π thereafter
$w_{i,j}$	Weight of action neuron j for input s_i and output a_j

1 Introduction

When animals and humans cannot perform tasks based on pre-existing knowledge or by imitating others, then a trial and learning approach becomes necessary. The success or failure of a trial should help to modify behavior in the "right" direction. The scientific research area called reinforcement learning (RL) is one of those studying such behavior. RL methods have been successfully applied to many problems where the agent has to explore its environment and learn interactively. The agent explores the environment by taking *actions* as a function of the current *state* and according to a stochastic rule called *policy*. The environment gives *reward* to the agent according to a task-dependent *reward function*. From this interaction with the environment, the agent attempts to identify a *value function* that maps states to actions in a way that allows the agent to maximize the *expected return* (i.e. the long-term reward). When the agent always takes the action that maximizes the value function in the current state it is said to follow a *greedy policy*, i.e. it exploits what it has learned during exploration.

In tasks with delayed reward the agent may have to explore the environment without receiving reward for a long time, which makes it difficult for the agent to know whether it is making progress towards reward-giving states or not. Even in tasks where reward is more frequent, exploration may be long before the agent has learned enough about the environment to perform well. *Undirected exploration* methods use some degree of randomness in action selection in order to promote sufficient exploration of the state space. Undirected exploration tends to be long even with state spaces of moderate size. *Directed exploration* methods utilize some exploration-specific knowledge that increases the knowledge gain so that the value function is learned faster or more completely.

In this paper we present a technique for directing exploration more explicitly by pre-existing knowledge about the task at hand. Humans and animals utilize such pre-existing knowledge in many tasks, e.g. when navigating through a maze it is usually advisable not to go back the same way that we arrived from and not take the same direction as we did the previous time if we come back to a previously visited state. It would be dangerous to try to avoid such actions by giving negative reward because they might actually be the optimal ones in the long run. Pre-existing knowledge may also originate from advice provided by other agents,

e.g. feedback given by teachers to their pupils when they try to solve exercises in mathematics. Such pre-existing knowledge is not a source of reward as such, but it helps the agent explore the state space in a way that allows him to receive reward faster or to receive reward at all. Several authors have emphasized the utility of such guidance during learning, e.g. Kaelbling et al. (1996, p. 33), Schaal (1997) and Millán, Posenato, & Dedieu (2002).

Pre-existing knowledge can be expressed in many different forms such as numerical controllers, rules, decision trees etc. Such knowledge is often sufficient by itself in deterministic environments with complete information about the task at hand but may fail in other kinds of environments. In this paper we will use heuristic rules as pre-existing knowledge for guiding exploration in order to speed up RL learning. When the performance of the RL system becomes better than the performance of the heuristic rules, the RL controller should dominate action selection. For managing this balance between exploration and exploitation we propose using a bi-memory model where the value function is stored in *long-term memory* while exploration rules modify the contents of the so-called *short-term memory*.

Both memories can be implemented using any kind of function approximation model, such as neural networks. The training rules used are also suitable for continuous-valued domains, which is an advantage compared to well-known directed exploration methods such as counter-based or recency-based exploration. Being able to guide exploration in continuous-valued domains is usually even more important than in discrete domains due to the infinite number of possible states. Because many real-world tasks (e.g. control systems) are continuous, such new methods could make it possible to use RL learning in tasks where it is not conceivable for the moment.

After this introduction, the most relevant RL methods to the scope of this paper are described in Section 2. Section 3 presents the bi-memory model and associated learning rules. Section 4 shows experimental results for three different tasks, followed by conclusions.

2 Reinforcement learning principles

One of the main domains treated by RL is solving *Markov Decision Processes* (*MDP*s). A (finite) MDP is a tuple $M = (S,A,T,R)$, where: S is a finite set of states; $A = \{a_1, \ldots, a_k\}$ is a set of $k \geq 2$ actions; $T = [P_{sa}(\cdot) \mid s \in S, a \in A\}$ are the next-state transition probabilities, with $P_{sa}(s')$ giving the probability of transitioning to state s' upon taking action a in state s; and R is a reward function that specifies the reward values given in different states $s \in S$. Tasks with special *terminal states* (or *goal* states), e.g. games, mazes, are called *episodic tasks*. Of the many different RL methods that exist, we will here concentrate on the currently most used ones, i.e. Temporal Difference (TD), Q-learning and SARSA.

2.1 Learning a value function

Most current RL methods are based on the notion of value functions. Value functions are either *state-values* (i.e. value of a state) or *action-values* (i.e. value of taking an action in a given state). The value of a state $s \in S$ can be defined formally as:

$$V^{\pi}(s) = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\}, \tag{1}$$

where $V^{\pi}(s)$ is the state value that corresponds to the expected return when starting in s and following the action selection policy π thereafter. The factor r_{t+k+1} is the reward obtained when arriving into states s_{t+1}, s_{t+2} etc. γ is a discounting factor that determines to what degree future rewards affect the value of state s. Action-value functions are denoted $Q(s,a)$, where $a \in A$.

Temporal difference (TD) learning is an example of so-called *bootstrapping* methods (Sutton, 1988). Bootstrapping signifies that value function updates occur at every state transition based not only on received reward, but also on the difference between the current state-value and the state-value of the next state. State-values are updated according to:

$$V(s_t) \leftarrow V(s_t) + \beta[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]e_{t+1}(s), \tag{2}$$

where β is a learning rate and $e_{t+1}(s)$ represents an *eligibility trace* (Klopf, 1972, as cited in Barto, Sutton, & Anderson, 1983). An eligibility trace memorizes at least a part of the state or state-action history of the current episode. The eligibility trace value is increased by one (*accumulating eligibility trace*) (Barto, Sutton, & Anderson, 1983) or set to one (*replacing eligibility trace*) (Singh & Sutton, 1996) every time the state is encountered during an episode. $0 \leq \lambda \leq 1$ is a *trace decay* parameter, which together with γ determines how quickly the trace decreases. For a replacing eligibility trace, a state's eligibility trace value $e_{t+1}(s)$ is calculated by:

$$e_{t+1}(s) = \begin{cases} \gamma\lambda e_t(s) & \text{if } s \neq s_t \\ 1 & \text{if } s = s_t \end{cases} \tag{3}$$

Q-learning (Watkins, 1989) is a TD control algorithm, which calculates action-values rather than state-values according to:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \beta\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right] e_{t+1}(s,a), \tag{4}$$

where $Q(s_t,a_t)$ is the value of action a in state s at time t. $e_{t+1}(s,a)$ is the eligibility trace for the state-action value $Q(s_{t+1},a_{t+1})$. The max-operator signifies the greatest action-value in state s_{t+1}. Q-learning is an off-policy method, i.e. Q-values converge to the true value function no matter what policy is used as long as the policy continues to update all state-action pairs infinitely.

SARSA is an on-policy TD control method, where action-values are updated by:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \beta[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] e_{t+1}(s,a), \tag{5}$$

where $Q(s_{t+1}, a_{t+1})$ is the action-value of the action selected by the policy in state s_{t+1}, so the action to be taken in the next state has to be determined before updating the action-value of the current state and action. This simplifies the use of eligibility traces compared to standard Q-learning, where taking non-greedy actions requires resetting the trace (Rummery & Niranjan, 1994).

2.2 Exploration/exploitation trade-off

Finding a good trade-off between exploration and exploitation is one of the most difficult tasks in RL for control (Singh et al., 1998). The policy π determines the balance between exploring the environment and exploiting already found, but possibly sub-optimal solutions. It is also possible to modify the reward function in such a way that it makes exploration more efficient.

2.2.1 Exploration policies that do not modify reward

Exploration policies that do not modify the reward function are of two types, i.e. *undirected exploration* methods that do not use any task-specific information and *directed exploration* methods that use some task-specific knowledge for guiding exploration. The most common undirected exploration methods are *ε-greedy* (also called *semi-uniform distributed exploration* in (Thrun, 1992)) and *Boltzmann* (also called *Softmax*) action selection. With ε-greedy action selection, a greedy action is selected with probability (1-ε) and an arbitrary action with probability ε using a uniform probability distribution. With Boltzmann action selection, actions are selected according to the probability

$$\text{Prob}(s_t, a_i) = \frac{e^{Q(s_t,a_i)/\tau}}{\sum_k e^{Q(s_t,a_k)/\tau}}, \tag{6}$$

where τ (called *temperature*) adjusts the randomness of action selection.

Many methods of directed exploration try to guide the exploration in such a way that the entire state space would be explored in order to learn the value function as well as possible. The action to be taken is usually selected by maximizing an evaluation function that combines action-values with some kind of *exploration bonus*, δ_n (Ratitch & Precup, 2003):

$$N(s,a) = K_0 Q(s,a) + K_1 \delta_1(s,a) + ... + K_k \delta_k(s,a) \tag{7}$$

The positive constants K_i control the exploration-exploitation balance by giving more or less influence on action selection to different exploration bonuses. Many different exploration bonus methods exist. *Counter-based* methods direct exploration towards states that were visited least frequently in the past, while *recency-based exploration* prefers states that were

visited least recently. Other methods use statistics about the underlying MDP in order to visit states whose value function estimates seem to be the least certain (Thrun, 1992) (Wiering, 1999) (Ratitch & Precup, 2003). All these methods require storing some supplementary information about the task in addition to the value function estimates. Counter-based and recency-based exploration methods only require storing one counter per state (or state-action pair in the case of action-value learning) while *model-based* methods attempt to construct a model of the environment (i.e. the underlying MDP) such as Dyna (Sutton, 1990), E^3 (Kearns & Singh, 1998) and R-Max (Brafman & Tennenholtz, 2002).

2.2.2 Reward shaping

Directed exploration methods only affect action selection without modifying the reward function R. Methods that modify R in order to guide the agent are called *reward shaping* methods. Reward shaping can make exploration significantly faster by modifying the reward given to the agent in a way that guides the agent directly towards a (hopefully) optimal policy (Gullapalli, 1992) (Mataric, 1994) (Randløv & Alstrøm, 1998) (Ng, Harada, & Russell, 1999) (Randløv, 2000). Reward shaping requires a priori knowledge about the environment, e.g. where the goal is located, the stochastic transition rate of the environment or the number of sub-goals to reach. In (Mataric, 1994), the reward function is decomposed into reward given for reaching a goal and *progress estimators* that guide exploration in a way that speeds up learning in a multi-robot task. The use of such progress estimators can be expressed in a similar way as for directed exploration (7):

$$R_M = K_0 R + K_1 F_1 + ... + K_k F_k \,, \tag{8}$$

where R is an "original" reward function and F_n are reward shaping functions. Badly selected shaping functions may mislead the agent into learning sub-optimal policies. The potential-based shaping functions introduced in (Ng, Harada, & Russell, 1999) define conditions under which the reward function R_M preserves the original optimal policy, where

$$F(s_t, a_t, s_{t+1}) = \gamma \Phi(s_{t+1}) - \Phi(s_t) \tag{9}$$

and Φ is a real-valued function. A potential-based shaping function guarantees that if the agent comes back to a previously visited state, then the sum of all discounted shaping

reward received during the cycle should be zero. If this is not the case, the optimal policy will be modified by the shaping function. For instance, if performing a cycle results in positive shaping reward, then the shaping function encourages the agent to perform such cycles (Ng, Harada, & Russell, 1999). Therefore modifying the actual reward function often leads to sub-optimal policies (Randløv & Alstrøm, 1998) (Ng, Harada, & Russell, 1999). This is why reward shaping is not used in the experiments reported in this paper.

2.2.3 Optimistic initial values

A technique called *optimistic initial values* (also called "optimism in the face of uncertainty" in Kaelbling, Littman, & Moore, 1996) gives a similar effect of preferring unexplored actions (and states) as for many directed exploration methods (Thrun, 1992, p.8). It is implemented by using initial value function estimates that are bigger than the expected ones. Therefore, unused actions have greater value estimates than used ones, so unused actions tend to be selected. When all actions have been used a sufficient number of times, the true value function overrides the initial value function estimates. An advantage of optimistic initial values compared to directed exploration methods is that no supplementary memory is needed.

A common implementation of optimistic initial values is to initialize action-values to zero and giving negative reward for every action performed (Sutton & Barto, 1998, p. 39). In addition to encouraging initial exploration, optimistic initial values with negative step reward perform a kind of reward shaping by continually driving the agent away from wherever it has been (Sutton & Barto, 1998, p. 215). This means that the agent will have a tendency of going away from the start state, which usually makes the agent reach a terminal state faster. The negative step reward can be expressed as a reward shaping function as in (8), where

$$F(s_t, a_t, s_{t+1}) = \begin{cases} -1 & \text{if } s_{t+1} \text{ is not a terminal state} \\ 0 & \text{if } s_{t+1} \text{ is a terminal state} \end{cases} \tag{10}$$

RL tasks that use equation (10) as their reward function R are called *stochastic shortest path problems* in (Bertsekas & Tsitsiklis, 1996).

3 Guiding exploration by pre-existing knowledge

This section describes the use of a *long-term memory* (*LTM*) and *short-term memory*[1] (*STM*) model for combining action-value learning with pre-existing knowledge that guides exploration, e.g. heuristic rules. LTM is used for action-value learning while STM is used for guiding state-space exploration by the *SLAP* (Set Lower Action Priority) principle, described in section 3.2. In the 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 (***BIMM***) (Främling, 2003) is a memory model that separates a short-term memory used for controlling exploration from the long-term memory used 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:

$$a_j(x) = \sum_{i=1}^{N} x_i w_{i,j} , \tag{11}$$

where x_i is the value of input variable i, $w_{i,j}$ is the weight of action neuron j for input i, a_j is the output value of 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 when allowing only one state variable to be one and forcing the others to be zero.

[1] The concept of "short-term memory" has been used in many different contexts and with varying signification. In many papers, e.g. McCallum (1995), short-term memory memorizes at least parts of the state history of one episode in order to improve identification of states in "hidden state" problems. Bakker (2002) uses a short-term memory for storing contextual clues to be used in near-future states. Barto et al (1983) also speak about using short-term memory for storing the eligibility trace, where they call it episodic memory. All these uses differ from the notion of short-term memory used here.

However, one major advantage of linear function approximators over lookup-tables is that they can handle continuous-valued state variables, where the state-space is infinite. Adalines can be trained using the Widrow-Hoff training rule (Widrow & Hoff, 1960):

$$w_{i,j}^{new} = w_{i,j} + \alpha(a_j' - a_j)x_i\,, \tag{12}$$

where a_j' is the "correct" or "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 all a_j' and a_j values. The RMSE error function has only one minimal solution for Adalines, so gradient descent is guaranteed to find optimal weights if α is small enough (this is also shown by Doya (1996) for TD learning). The generalized Widrow-Hoff rule allows for gradient descent also in multi-layer ANNs, but then training is usually much longer and the risk of local minima in the error function means that finding optimal weights is not guaranteed. The use of multi-layer ANNs in RL tasks is discussed for instance in Barto et al. (1990), where s_i is replaced by the feature value ϕ_i that may be calculated by the hidden layer of a multi-layer ANN, be values from measuring instruments or some other mechanism that estimates the system's state.

When using BIMM in a RL setting where the inputs x_i correspond to state variables s_i, the combined output of two Adalines is calculated according to:

$$a_j(s) = K_0 \sum_{i=1}^{N} ltw_{i,j} s_i + K_1 \sum_{i=1}^{N} stw_{i,j} s_i\,, \tag{13}$$

where s_i is the value of state variable i, $a_j(s)$ is the sum of the estimated action-value and the exploration bonus value of the STM. $a_j(s)$ thereby corresponds to $N(s,a)$ in equation (7) while K_0 and K_1 are positive constants that control the balance between exploration and exploitation as in (7). $ltw_{i,j}$ is the LTM weight and $stw_{i,j}$ is the STM weight for action neuron j and input i.. Both Q-learning and SARSA can be used to update LTM weights. Replacing $Q(s,a)$ with $ltw_{a,s}$ in equation (4) gives:

$$ltw_{a,s} \leftarrow ltw_{a,s} + \beta\left[r_{t+1} + \gamma \max_a ltw_{a,s(t+1)} - ltw_{a,s} \right] e_{t+1}(s,a) \tag{14}$$

The SARSA update rule (5) can be used in the same way.

3.2 Guiding exploration

Exploration bonuses can 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, 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. STM weights are modified using the Widrow-Hoff rule (12) in order to change action probabilities. In the tests performed in this paper, action probabilities are only reduced according to the *set lower action priority* (SLAP) principle, where the target value $a_j'(s)$ is smaller than the current action value $a_j(s)$. However, it is not obvious how to select the target value $a_j'(s)$. A good guess would be to use the minimal action value $a_{min}(s)$ for state s as the target value. The problem with this is that after SLAPing different actions they would eventually end up having the same action value, which is not desirable as explained in detail in section 3.3. For the moment the best choice for the target value to use remains an open issue. In this paper we have selected to use $a_{min}(s)$ minus a constant "*margin*" as the target value:

$$a_j'(s) = a_{min}(s) - margin \tag{15}$$

The *margin* should have a "small" value (0.1 has been used in all tests reported in this paper), which ensures that an action that is repeatedly SLAPed will eventually have the lowest action value in state s. If no *margin* would be used, all $a_j(s)$ values would eventually converge to the same target value if all actions were SLAPed repeatedly. As explained in section 3.3, this could lead to losing an action ranking that is useful for systematic exploration. It might also be useful to adjust the action values of other actions than the SLAPed one. Increasing an action's probability to be selected works in a similar way by using the Widrow-Hoff update rule with the target value:

$$a_j'(s) = a_{max}(s) + margin \tag{16}$$

SLAP only modifies STM weights by the Widrow-Hoff rule, which becomes:

$$stw_{i,j}^{new} = stw_{i,j} + \alpha(a_j' - a_j)s_i \tag{17}$$

When inserting equation (17) into equation (13), we obtain the new activation value

$$\begin{aligned} a_j^{new}(s) &= K_0\sum_{i=1}^{N} ltw_{i,j}s_i + K_1\sum_{i=1}^{N} stw_{i,j}^{new}s_i = K_0\sum_{i=1}^{N} ltw_{i,j}s_i + K_1\sum_{i=1}^{N} s_i\left(stw_{i,j} + \alpha(a_j' - a_j)s_i\right) = \\ K_0\sum_{i=1}^{N} ltw_{i,j}s_i &+ K_1\sum_{i=1}^{N} stw_{i,j}s_i + K_1\sum_{i=1}^{N} s_i^2\alpha(a_j' - a_j) = a_j + \alpha K_1\sum_{i=1}^{N} s_i^2(a_j' - a_j) \end{aligned} \tag{18}$$

where we can see that setting α to $1/(\Sigma s_i^2 K_1)$ 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 (18) gives a generalization for BIMM of the well-known Normalized Least Mean Squares (NLMS) method that is commonly used in signal processing instead of the basic Widrow-Hoff rule:

$$stw_{i,j}^{new} = stw_{i,j} + \alpha(a_j' - a_j)s_i \Big/ K_1\sum_{i=1}^{N} s_i^2 \tag{19}$$

The NLMS method 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 (Främling, 2004). This is why NLMS (19) is used in all experiments reported in section 4. For instance, if $\alpha = 1.0$ in equation (19) 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. The value of α could also be modified depending on the degree of certainty of the heuristic rule that is used, thereby performing stronger or weaker SLAPs.

As long as the value of $a_{min}(s)$ doesn't change, the target 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, $a_j(s)$ values go slowly towards minus infinity when using SLAP an infinite number of times.

Error! Not a valid bookmark self-reference. gives a general procedure for using SLAP in a typical action-value learning task. Especially when using SARSA or other on-policy methods for action-value updates, it is important to update STM weights before updating the action-values, otherwise the changes in STM weights might modify the action selection for

the next state. It is also important to notice 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. SLAP can even be used for any state-action pair at any time independently of the current state and the current action selected. This is a major difference compared with most existing methods of directed exploration, e.g. counter- and recency-based exploration whose values are updated synchronously with action-value updates. Such directed exploration methods also assume discrete state-spaces, where the current state is uniquely identified by binary features as for lookup-table representations of the action-value function. Especially with continuous-valued representations, e.g. Radial Basis Function (RBF) networks, they cannot be used. SLAP uses well-known learning rules for continuous-valued representations so it can be used with such representations by definition.

Algorithm 1. General algorithm for using SLAP in episodic RL task.

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'

3.3 Increasing exploration

As shown by the experimental tasks in section 4, it is often easy to identify heuristic rules that make exploration faster by avoiding taking actions that do not seem to be useful for some reason. However, these rules also need to ensure sufficient exploration of the state space. This is an explicit goal of most directed exploration methods. It is also obtained by using optimistic initial values.

If the heuristic rules do not generate sufficient exploration of the state space, then it has to be provided by other means. STM values are reinitialized before starting a new episode and can have a great impact on the way in which the state space is explored. If we 1) set $\alpha = 1.0$, 2) initialize STM weights to random values; 3) use greedy action selection according to equation (13); 4) always SLAP the greedy action after using it and 5) use a $margin > 0$, then actions will be selected in a cyclic order. If STM weights are initialized to zero or if $margin = 0.0$, actions will often have equal values for $a_j(s)$ in equation (13) so they will be selected randomly rather than in a cyclic order.

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, so actions will be selected in a random (but cyclic) order independently of the value of K_1 in equation (13) 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 and their relative importance as determined by the parameters K_0 and K_1 in equation (7). Setting K_1 to some very small value will give no or very little randomness (10^{-6} has been used in the tests of section 4) while a greater value will give more randomness. Unfortunately, at least in discounted reward tasks this method does not give uniform exploration over the whole state space because LTM values (and their differences) tend to be greater close to reward-giving states than far from them, so the randomness of STM weights tends to generate more exploration far from reward-giving states than close to them. To avoid this, an undirected exploration method such as ε-greedy exploration can be used instead of always taking the greedy action indicated by $a_j(s)$ in equation (13).

Results for three different initialization/exploration methods are plotted in Figure 1 for a deterministic 20x20 grid world with start and terminal states in opposite corners. The heuristic rules applied when entering a new state are 1) SLAP action that would take agent back to previous state and 2) SLAP the current greedy action. The following observations can be made:

1. Using zero initial STM weights (A) is slower than the others. Total number of steps for episodes 1-250: A) 36900; B) 29600; C) 32005.

2. Random initial STM weights and K_1 = 0.1 (B) gives the fastest exploration during the 50 first episodes. Number of steps in first episode: A) 1180; B) 810; C) 1220.

3. ε-greedy exploration (C) gives the best exploitation results, obtained by setting K_1 and ε to zero after 200 episodes, i.e. using greedy exploration by the action-values in LTM. Average number of steps for episodes 241-250: A) 39.6; B) 39.3; C) 38.0 (optimal).

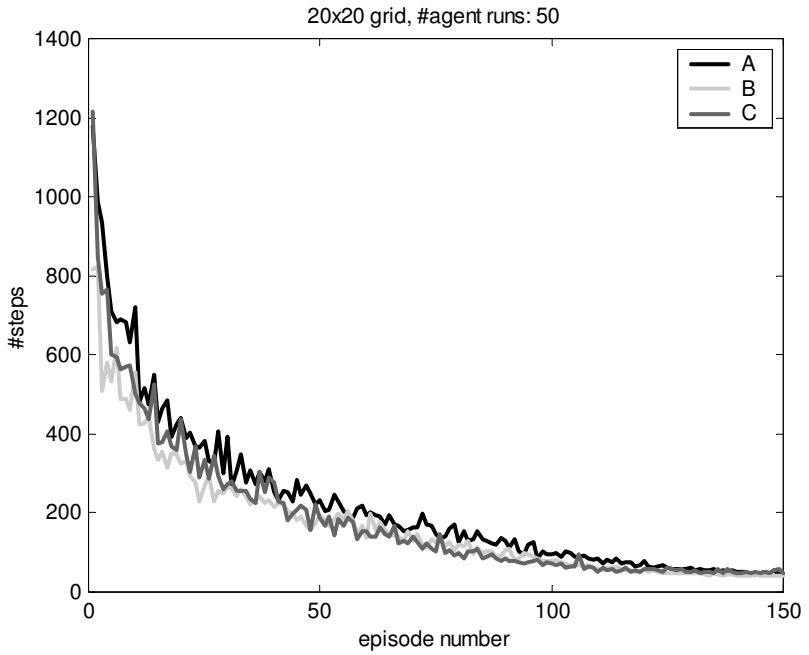

Figure 1. Comparison of different STM initialization and action selection methods for a deterministic 20x20 grid. A) zero STM initial weight, K_1 *= 1.0; B) random STM initial weights in [0,1),* K_1 *= 0.1 and C) random STM initial weights in [0,1) , K1 =* 10^{-6} *and* ε*-greedy exploration with* ε *= 0.1.* α *= 1.0 for A and B,* α *= 0.9 for C.*

Even in deterministic tasks α should be smaller than 1.0 when using ε-greedy exploration. This is because circuits can occur due to the taking of non-greedy actions. In that case it would be unwise to directly make the current greedy action the last one in the action ranking of the current state. The same applies to tasks with stochastic state transitions treated in the next section.

The extent to which STM weight values $stw_{i,j}$ are reduced due to SLAPing was studied using agent B. In practice, they do not become too small. The smallest $stw_{i,j}$ value that occurred during the first episodes of ten runs was zero. This is probably because actions are SLAPed

only about 1.5 times the number of steps taken during the first episode so the STM weights are not decreased very frequently. When the action-values have been learned, actions are SLAPed one time less than the number of steps taken during the episode, which signifies that only the reversing action is SLAPed but no more circuits occur.

4 Experimental results

This section compares results of different methods presented in the previous sections on three different kinds of tasks:

1. Grid worlds with deterministic and stochastic state transitions (classical MDP).
2. Maze world with transition delay, deterministic and stochastic state transitions.
3. Mountain-Car task with continuous-valued state variables, discretized by an 8x8 grid.

In order to simplify the choice of learning parameters and for reasons of comparability, constant learning parameters are used for all methods even though perfectly tuned decaying learning parameters could give better results (Singh et al., 1998).

BIMM/SLAP is a model-free exploration method in the sense that it does not attempt to learn a model of the underlying MDP for the task at hand. Instead it uses pre-existing knowledge acquired during earlier experiences with similar tasks, provided by a trainer or other means. Therefore the performance of BIMM/SLAP is here compared only against other model-free exploration methods. The exploration methods that are compared are:

1. ε-greedy exploration: zero initial Q-values, r = 1.0 at terminal state ("goal") and zero for all other state transitions, identified as **Q** or **SARSA** as appropriate.
2. Optimistic initial values: zero initial Q-values, r = 0.0 at goal and r = -1.0 for all other state transitions, identified as **OIV**.
3. BIMM/SLAP with heuristic rules: zero initial Q-values (LTM weights), r = 1.0 at goal and zero for all other state transitions, identified as **BIMM**.
4. Counter-based exploration: zero initial Q-values, r = 1.0 at goal and zero for all other state transitions, identified as **CTRB**. Not used in Mountain-Car task.

The counter-based exploration bonus in (7) is here implemented as:

$$\delta_1(s,a) = \begin{cases} 1.0 - \dfrac{cnt(s,a) - cnt(s)_{\min}}{cnt(s)_{\max} - cnt(s)_{\min}} & \text{if } \left(cnt(s)_{\max} - cnt(s)_{\min}\right) > 0 \\ 0.0 & \text{if } \left(cnt(s)_{\max} - cnt(s)_{\min}\right) = 0 \end{cases} \quad (20)$$

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 all actions in state s. Counter values are reset after every episode.

4.1 Grid world

Grid worlds are examples of Stochastic Shortest Path Problems (Bertsekas & Tsitsiklis, 1996) that are often used for comparing different RL methods (Sutton, 1990, 1991), (Moore & Atkeson, 1993), (Kaelbling, Littman, & Moore, 1996), (Ng, Harada, & Russell, 1999). There are also some limitations to grid worlds. States are uniquely and deterministically identified and there is a known "inverse" action to every possible action, i.e. an action that normally takes the agent back to the previous state. There are four possible actions that correspond to the compass directions east, west, north and south. Taking forbidden actions, i.e. actions making the agent hit a wall, is not counted here. Both deterministic and stochastic state transitions are tested. In the deterministic case, the agent always moves in the intended direction. In the stochastic case, two stochastic state transition rates were tested (0.2 and 0.5), which indicate the probability of some other of the possible directions to be taken than the intended one. Two different sizes of grid worlds are used here, i.e. 20x20 = 400 states and 40x40 = 1600 states, where the start state and the terminal state are in opposite corners of the grid world.

Q-learning (equation (4)) without eligibility trace is used by all methods for action-value learning. Learning parameters for the agents Q, OIV and CTRB are indicated in Table 1. In deterministic tasks $\beta = 1.0$ is always the best learning rate for Q-learning (Kaelbling, Littman, & Moore, 1996).

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 highest value $a_j(s)$ in equation (13) for the new state. The rules are applied in the order indicated, so at least with

$\alpha = 1.0$ the action taken at the previous visit becomes the last one in the action ranking given by equation (13). BIMM parameter values are listed in Table 1. In the deterministic case, ε-greedy exploration converged to a better policy than when using a "high" K_1-value (e.g. 0.1). In the stochastic tasks such supplementary exploration was not needed.

Table 1. Parameter values of methods Q, OIV, CTRB in grid world tasks. All parameters not indicated here were set to zero.

Agent	***Q*, γ=0.95**		***OIV***		***CTRB*, γ=0.95**		
Grid world	β	ε	β	γ	β	K_1 20x20	K_1 40x40
Deterministic	1	0.1	1	1	1	0.1	0.01
Stochastic, 0.2	0.1	0.1	0.5	0.95	0.5	0.01	0.01
Stochastic, 0.5	0.1	0.1	0.5	0.95	0.5	0.01	0.001

Table 2. BIMM parameter values in grid world tasks. Discount rate was γ=0.95 and $K_1 = 10^{-6}$ in all tasks.

Grid world	α	β	ε
Deterministic	0.9	1	0.1
Stochastic, 0.2	0.2	0.5	0.0
Stochastic, 0.5	0.1	0.5	0.0

With the 20x20 grid all agents performed 250 episodes. After 200 episodes actions were selected greedily using the learned action-values 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. The OIV agent is always using a greedy policy, so no modification of exploration parameters is needed. With the 40x40 grid all agents performed 1000 episodes, using greedy exploration after 900 episodes.

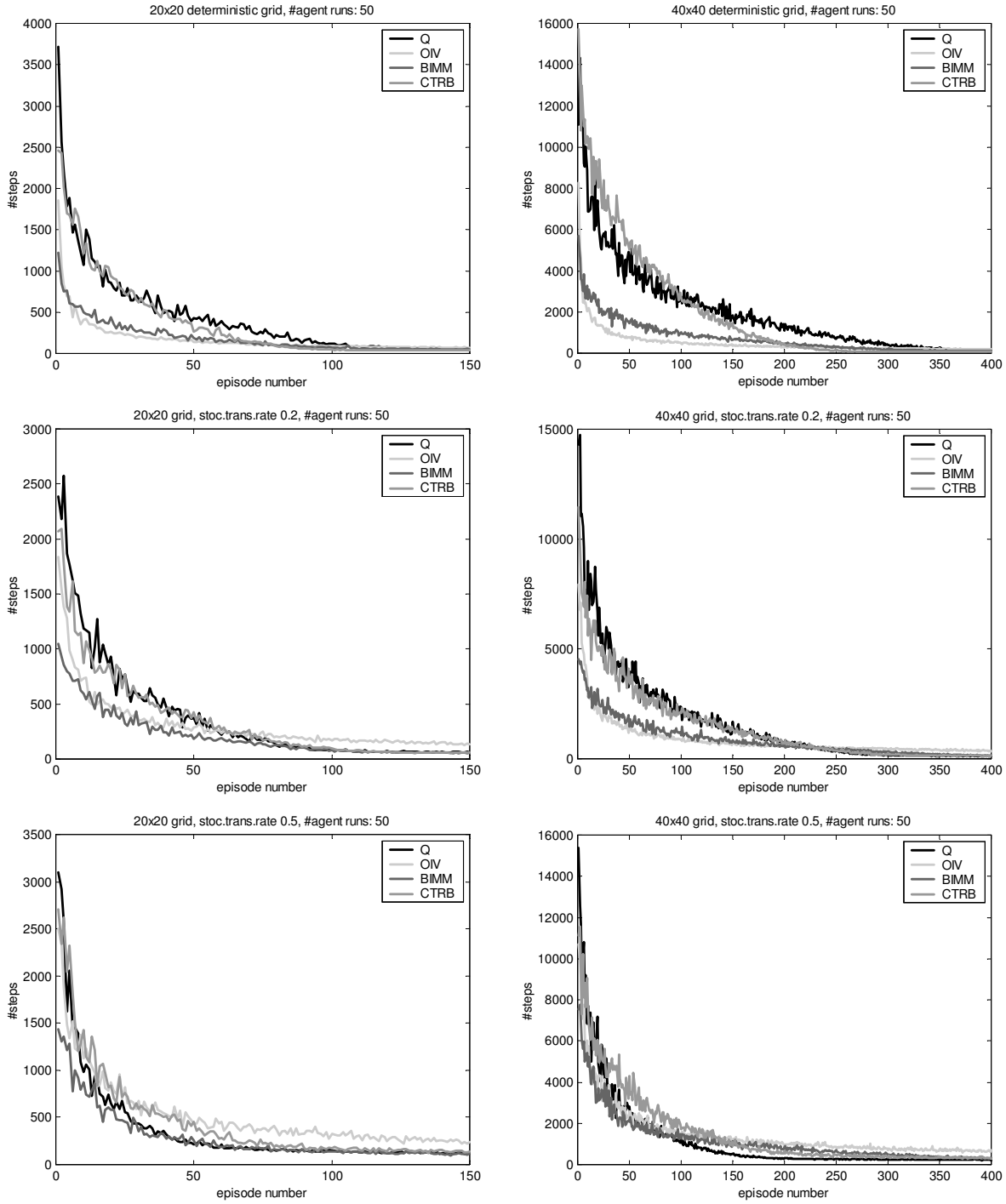

Figure 2. Grid world results as average number of steps per episode. Grid size, stochastic transition rate and the number of agent runs used for calculating the average number of steps are indicated at the top of each graph.

The graphs in Figure 2 show the average number of steps as a function of the episode. From the figures it seems clear that BIMM converges towards a good policy after much less exploration than Q- and CTRB-agents. In most figures the BIMM graph is constantly below the graph of the OIV agent and, because OIV converges slowly, BIMM always goes below OIV at some moment. For the deterministic 20x20 grid OIV converged to a stationary policy after about 600 episodes and for the deterministic 40x40 grid after about 3000 episodes, which is much slower than for the other methods. The converged policy[2] also becomes worse with a greater stochastic state transition rate. This is because a greater stochastic state transition rate increases the probability of cycles during exploration. When giving negative reward after every state transition as in equation (10), even an optimal action is punished when such cycles occur. Therefore the reward function in equation (10) is not a potential-based shaping function as defined in (Ng, Harada, & Russell, 1999). A necessary condition for a potential-based shaping function is that when coming back to a previously visited state, the sum of all rewards given by the potential-based shaping function between the two visits should be zero. This could be avoided by using a first-visit OIV agent instead of the every-visit one but it would also modify the exploration behavior (Bertsekas & Tsitsiklis, 1996) (Singh & Sutton, 1996).

Table 3. Grid world results. Numbers are indicated as Q/OIV/BIMM/CTRB. Steps with "converged" policy are averages of episodes 241-250 for 20x20 grids and episodes 951-1000 for 40x40 grids.

Grid size	**Steps on first episode**	**Steps with converged policy**	**Total number of steps**
20x20 det.	3720/1850/**1220**/2460	38.3/52.8/**38**/38.9	68000/**31500**/32000/58400
40x40 det.	11100/8290/**5710**/15700	78.0/116/**78.0**/80.2	838000/**284000**/343000/812000
20x20 stoc.0.2	2380/1840/**1040**/2070	52.9/99.3/**52.5**/54.9	61800/57000/**36600**/57700
40x40 stoc.0.2	14300/7950/**4550**/11500	108/208/**104**/109	724000/499000/**415000**/651000
20x20 stoc.0.5	3100/2500/**1430**/2700	101/180/**88.0**/91.9	64100/96800/**52300**/76700
40x40 stoc.0.5	15400/10700/**7520**/11100	187/385/**177**/196	**589000**/877000/590000/759000

[2] *Converged policy* here signifies a policy for which no significant improvement is observed during continued learning, but which is not necessarily a *stationary policy*.

Table 3 gives numeric comparisons for the performance of the four methods. BIMM agents converge to a better policy than the others in all tests as indicated by the third column. The total number of steps is also clearly lower than the others in three cases and very close to the lowest for the three others. Because the converged policy of BIMM is better than for the others, continuing the simulation would make the total number of steps the lowest for BIMM also for these three tasks. As 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 (as shown by second column of Table 3 and Figure 2), this is compensated by an even better converged policy compared with the other agents.

4.2 Maze with transition delays

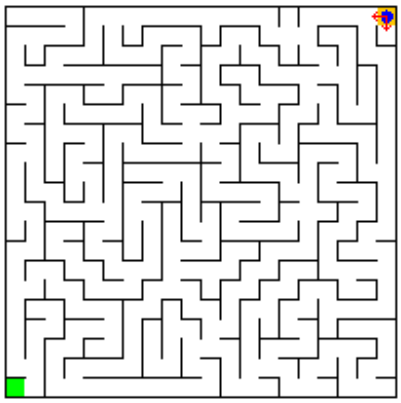

Figure 3. Maze with 10 supplementary "doors" opened in the walls in addition to the initial unique route. Agent is located in start state, terminal state in lower left corner.

Semi-Markov Decision Processes (SMDP) differ from MDP's by taking into account state transition time that depends on a probability distribution F_{xy} (Boyan & Littman, 1994) (Bradtke & Duff, 1995). State transition time is here introduced by the notion of "corridors", i.e, states with only two possible actions that represent opposite directions. When in a corridor, the agent deterministically continues forward until the end of it. The transition time here introduces an "extra penalty" for bad decisions like walking around in circles or running into dead-ends. The maze world is of size 20x20, where 10 supplementary doors have been opened in addition to the initial unique solution, also making circuits

possible (Figure 3). Learning parameters were identical to those indicated in Table 1 and Table 2 with two exceptions. The CTRB agent used the same parameter values as for the 20x20 grid, but for stochastic transition rate 0.5 it used $K_1 = 0.001$ instead of 0.01. For the deterministic maze, the BIMM agent used $K_1 = 0.1$, $\alpha = 1.0$ and $\varepsilon = 0.0$, which gave slightly better results than using ε-greedy exploration in this task.

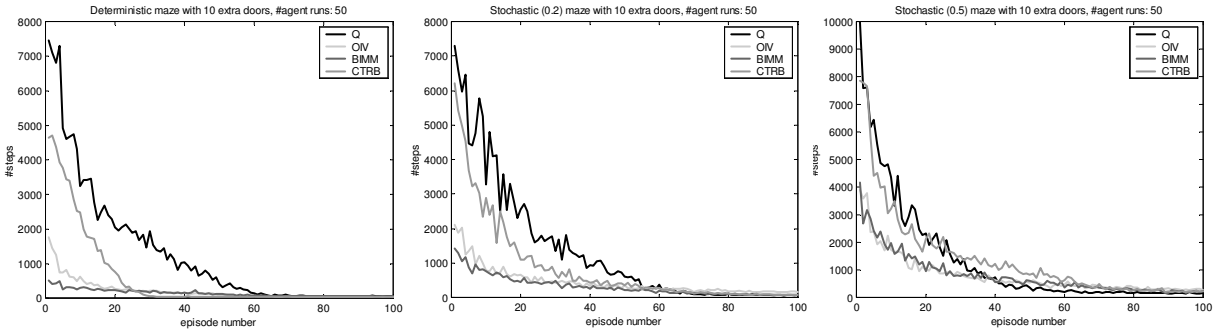

Figure 4. 20x20 maze results as average number of steps per episode from 50 agent runs.

As for the grid worlds, a greedy policy was used for all agents after 200 episodes. The results shown in Figure 4 and Table 4 confirm the results from the grid worlds in the previous section, but the advantage of BIMM compared to the other agents is even clearer in this task.

Table 4. Results for 20x20 maze with ten extra doors. Numbers are indicated as Q/OIV/BIMM/CTRB. The third column indicates the number of steps for the converged policy as average value of episodes 241-250.

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

4.3 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 three actions

$a_t \in \{+1,0,-1\}$ corresponding 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 of which only one has a value of one while the others are zero. Episodes were limited to 1 000 000 steps because some agents sometimes ran into infinite episodes.

As in most previous work on this task, the SARSA(λ) learning algorithm with replacing eligibility traces was used for action-value learning. The parameter values are:

- **SARSA**: $\beta = 0.1$, $\gamma = 0.9$, $\lambda = 0.95$, $\varepsilon = 0.1$.
- **OIV**: $\beta = 0.1$, $\gamma = 1.0$, $\lambda = 0.9$.
- **BIMM**: $\beta = 0.1$, $\gamma = 0.9$, $\lambda = 0.95$, $K_1 = 0.1$, $\alpha = 1.0$.

The counter-based method was not used in this task. Since the state-variables are continuous, it is often necessary to take several steps before changing state in the discretized state space. In that situation, ordinary counter-based exploration tends to change the used action at every step in the beginning of learning, which easily leads to infinite episodes and no learning . The same happens if the heuristic rules used by BIMM in the grid and maze tasks are applied to the Mountain-Car task.

Randløv (2000) tried to find a good reward shaping function for the Mountain-Car task but it turned out to be difficult to find a shaping function that would not lead to sub-optimal solutions. When using BIMM and SLAP the reward function does not need to be modified so it becomes easier to find usable heuristic rules. 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, 2005, 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 BIMM and SLAP methods can be used successfully for guiding exploration with pre-existing knowledge, rather than evaluate the goodness of the pre-existing knowledge itself.

The results shown in Figure 5 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. Therefore the BIMM agent clearly learned the action-value function better and with less exploration; the total numbers of steps for episodes 1-10000 are 1 140 000 for SARSA, 830 000 for OIV and 554 000 for BIMM. SLAP was applied an average of 14 times during the first episode. After about 100 episodes SLAP was applied less than once per episode as an average and continues decreasing until insignificant for the performance of the agent.

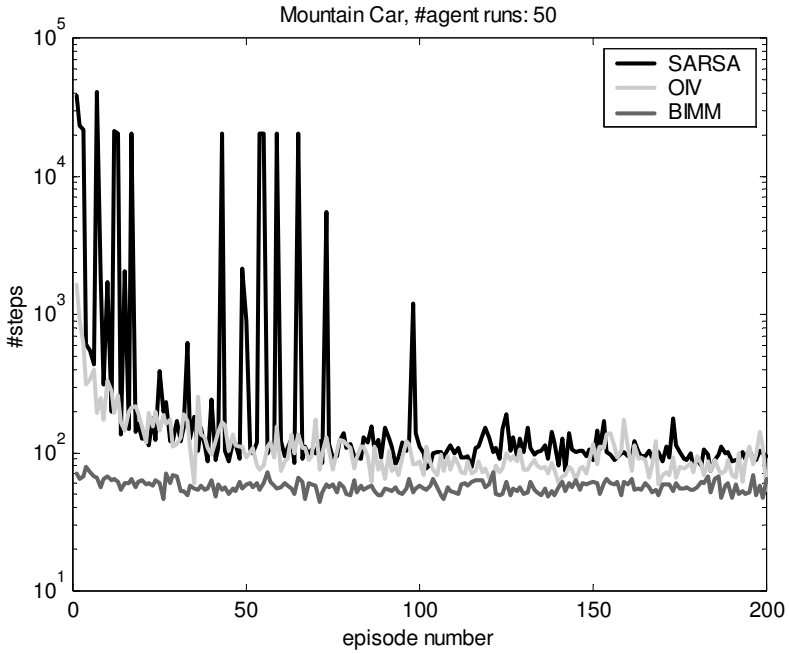

Figure 5. Average number of steps as a function of episode from 50 runs. The peaks are due to infinite episodes, limited to 1 000 000 steps. Note the logarithmical scale on the y-axis.

5 Conclusions

This paper presents new methods for guiding exploration in reinforcement learning tasks by pre-existing knowledge, expressed by heuristic rules in the experiments reported in this paper. The results show that with appropriate (but not necessarily optimal or complete) rules the length of exploration can be significantly reduced both in deterministic and stochastic environments. It is also shown that the reduced exploration efforts do not reduce the probability of converging to a good policy.

Even though only "toy tasks" were used here, it should be possible to generalize the results to other RL tasks. Since BIMM and SLAP use standard ANN structures and learning rules, they are also applicable to tasks where explosion of the state-space size due to state variable discretization is a problem. Experiments not reported here show that BIMM and SLAP perform as well or better with a fuzzy classifier instead of a discrete one in the Mountain-Car task. However, especially in tasks involving discounted reward it is difficult to perform action-value learning using classical methods such as Q-learning or SARSA with general

function approximators such as ANNs (Baird, 1995) (Doya, 1996, 2000). Even though it is possible to learn successfully, the results tend to be sensitive to the function approximator used and its learning parameters. Solving this issue is a subject of current and future research.

References

Baird, L. (1995). Residual Algorithms: Reinforcement Learning with Function Approximation. In *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 30-37). San Francisco, CA: Morgan Kaufman Publishers.

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

Barto, A.G., Sutton, R.S., & Anderson, C.W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics,* **13:5**, 835-846.

Barto, Andrew G., Sutton, R.S., & Watkins C.J.C.H (1990). Learning and Sequential Decision Making. In M. Gabriel & J. Moore (Eds.), *Learning and computational neuroscience : foundations of adaptive networks*. Cambridge, MA: M.I.T. Press.

Bertsekas, D. P. & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific.

Boyan, J. A. & Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. In J. D. Cowan, G. Tesauro, & J. Alspector (Eds.), *Advances in Neural Information Processing Systems 6* (pp. 671-678). Morgan Kaufmann.

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* (pp. 393-400). Morgan-Kaufmann.

Brafman, R.I. & Tennenholtz, M. (2002). R-max - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, **3**, 213-231.

Doya, K. (1996). Temporal Difference Learning in Continuous Time and Space. In D. Touretzky, M. Mozer, M. Hasselmo (Eds), *Advances in Neural Information Processing Systems 8* (pp. 1073-1080). MIT Press.

Doya, K. (2000). Reinforcement Learning in Continuous Time and Space. *Neural Computation*, **12**, 219-245.

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

Främling, K. (2004). Scaled gradient descent learning rate - Reinforcement learning with light-seeking robot. In *Proceedings of International Conference on Informatics in Control, Automation and Robotics (ICINCO)* (pp. 3-11). Setubal, Spain, 25-28 August 2004.

Främling, K. (2005). Adaptive robot learning in a non-stationary environment. In *Proceedings of European Symposium on Artificial Neural Networks (ESANN)* (pp. 381-386). Bruges, Belgium, 27-29 May 2005.

Gullapalli, V. (1992). *Reinforcement Learning and its Application to Control*. Univ. of Massachusetts, Amherst: Ph.D. Thesis, COINS Technical Report 92-10.

Kaelbling, L.P., Littman, M.L., & Moore, A.W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, **4**, 237-285.

Kearns, M. & Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. In *Proceedings of 15th International Conference on Machine Learning* (pp. 260-268). San Francisco, CA: Morgan Kaufmann.

Klopf, A.H. (1972). *Brain function and adaptive systems – A heterostatic theory*. Bedford, MA: Air Force Cambridge Res. Lab. Res. Rep. AFCRL-72-0164.

Mataric, M.J. (1994). Reward Functions for Accelerated Learning. In W. W. Cohen & H. Hirsch (Eds.) *Machine Learning: Proceedings of the Eleventh International Conference* (pp. 181-189). Morgan-Kaufmann.

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* (pp. 377-384). MIT Press.

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

Moore, A.W. & Atkeson, C.G. (1993). Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning*, **13**, 103-130.

Ng, A.Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of 16th International Conference on Machine Learning* (pp. 278-287). San Francisco, CA: Morgan Kaufmann.

Randløv, J. & Alstrøm, P. (1998). Learning to Drive a Bicycle using Reinforcement Learning and Shaping. In *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 463-471). Morgan-Kaufmann.

Randløv, J. (2000). Shaping in Reinforcement Learning by Changing the Physics of the Problem. In *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 767-774). San Francisco, CA: Morgan Kaufmann.

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)* (pp. 313-324). Heidelberg: Springer-Verlag.

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

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

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

Singh, S., Jaakkola, T., Littman, M.L., & Szepesvari, C. (1998). Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Machine Learning*, **38**, 287-308.

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

Sutton, R.S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning* (pp. 216-224). Morgan Kaufmann.

Sutton, R.S. (1991). Integrated Modeling and Control Based on Reinforcement Learning and Dynamic Programming. In R. Lippmann, J. Moody, & D. Touretzky (Eds.) *Advances in Neural Information Processing Systems 3* (pp. 471-478). Morgan-Kaufmann.

Sutton, R.S. & Barto, A.G. (1998). *Reinforcement Learning*. Cambridge, MA: MIT Press.

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*. New York, NY: Van Nostrand Reinhold.

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

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

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