Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2010 Special Issue

Online learning of shaping rewards in reinforcement learning

Marek Grześ*, Daniel Kudenko

Department of Computer Science, University of York, York, YO10 5DD, UK

ARTICLE INFO

Article history:
Received 8 February 2009
Received in revised form 14 July 2009
Accepted 2 January 2010

Keywords:
Potential-based reward shaping
Reinforcement learning
Learning heuristic

ABSTRACT

Potential-based reward shaping has been shown to be a powerful method to improve the convergence rate of reinforcement learning agents. It is a flexible technique to incorporate background knowledge into temporal-difference learning in a principled way. However, the question remains of how to compute the potential function which is used to shape the reward that is given to the learning agent. In this paper, we show how, in the absence of knowledge to define the potential function manually, this function can be learned online in parallel with the actual reinforcement learning process. Two cases are considered. The first solution which is based on the multi-grid discretisation is designed for model-free reinforcement learning. In the second case, the approach for the prototypical model-based R-max algorithm is proposed. It learns the potential function using the free space assumption about the transitions in the environment. Two novel algorithms are presented and evaluated.

1. Introduction

Reinforcement learning (RL) is a popular method to design autonomous agents that learn from interactions with the environment, or, in more mathematical terms, from repeated simulation (Bertsekas, 2007). In contrast to supervised learning (Mitchell, 1997), RL methods do not rely on instructive feedback; i.e., the agent is not informed as to what the best action is in a given situation. Instead, the agent is guided by the immediate numerical reward which defines the optimal behaviour for solving the task. This leads to the *temporal credit assignment* problem; i.e., the problem of determining which part of the behaviour deserves the reward (Sutton & Barto, 1998). To deal with this issue, conventional RL algorithms employ a delayed approach which is based on the back-propagation of the value function which is defined on the state space. However the back-propagation is time consuming, since it is an iterative approach.

To speed up the learning process, and to tackle the temporal credit assignment problem in a more efficient way, the concept of *reward shaping* has been considered in the field (Gullapalli & Barto, 1992; Konidaris & Barto, 2006; Mataric, 1994; Ng, Harada, & Russell, 1999; Randløv & Alstrom, 1998). The idea of reward shaping is to give additional (numerical) feedback to the agent in order to improve its convergence rate. Even though reward shaping has been powerful in many experiments, it quickly turned out that, used improperly, it can also be misleading (Randløv & Alstrom, 1998). To deal with such problems, potential-based reward shaping $F(s, s')$ was proposed (Ng et al., 1999; Wiewiora, 2003) as the difference of some potential function Φ defined over a source s and a destination state s':

$$F(s, s') = \gamma\Phi(s') - \Phi(s), \tag{1}$$

where γ is a discount factor. Ng et al. (1999) proved that reward shaping defined in this way leaves the optimal behaviour unchanged while the time for attempting suboptimal actions can be reduced. One problem with reward shaping is that often detailed knowledge of the potential function of states is not available, or very difficult to represent directly in the form of a shaped reward.

The main goal of this paper is to develop an algorithmic solution which would allow applying the potential-based reward shaping when the potential function cannot be defined manually. Generally a similar problem exists in informed heuristic search which also requires an admissible heuristic (Russell & Norvig, 2002). Specifically, two algorithms are proposed in this paper. The first one is designed for model-free RL. It is based on the multi-grid discretisation of the state space. The state space with higher generalisation is used to learn a high level value function which is treated as a potential function for reward shaping of the actual learning process. The second approach is for model-based RL, and applies the free space assumption to create and refine the model of environment dynamics for learning the potential function. These two approaches have one feature in common. They both assume that the value function at an abstract level is used as a potential function. In the first case it is a value function of the low resolution discretisation, and in the second case the model constructed according to the free space assumption is used to evaluate such a value function.

* Corresponding author.
E-mail addresses: grzes@cs.york.ac.uk (M. Grześ), kudenko@cs.york.ac.uk (D. Kudenko).

0893-6080/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2010.01.001

The remainder of this paper is organised as follows. Section 2 gives a brief overview of a Markov Decision Process. Reinforcement learning and reward shaping are discussed in Sections 3 and 4. The main contribution of this paper is divided into two main parts. In Section 5 our method to learn a potential function for reward shaping in model-free RL is introduced, and in Section 6 a corresponding algorithm for the model-based case is presented. Each of these major sections of the paper is additionally divided into a number of sub-sections which introduce a given approach, describe experimental settings and tested algorithms and parameters, and end with a presentation and discussion of obtained results. The paper ends with a review of related work in Section 7 and a conclusion in the final section.

This paper extends the work presented at ICANN'08 (Grzes & Kudenko, 2008a).

2. Markov decision processes

A Markov Decision Process (MDP) is a tuple (S, A, T, R), where $s \in S$ is the state space, $a \in A$ is the action space, $T(s, a, s')$ is the probability that action a when executed in state s will lead to state s', $R(s, a, s')$ is the immediate reward received when action a taken in state s results in a transition to state s' (Puterman, 1994). The problem of solving an MDP is to find a policy (i.e., a mapping from states to actions) which maximises the accumulated reward. When the environment dynamics (i.e., transition probabilities and a reward function) are available, this task becomes a planning problem (Ghallab, Nau, & Traverso, 2004) which can be solved using iterative approaches like policy and value iteration. Policy and value iteration belong to a large family of dynamic programming methods (Bertsekas, 2007).

3. Reinforcement learning

The policy and value iteration methods require access to an explicit, mathematical model of the environment, that is, transition probabilities, $T(s, a, s')$, and the reward function, $R(s, a, s')$. When such a model is not available, policy and value iteration cannot be applied. However the concept of an iterative approach in itself is the backbone of the majority of algorithms for learning a policy when the model is not available. Algorithms for learning in the absence of the model are known as reinforcement learning (RL) (Sutton & Barto, 1998) or neuro-dynamic programming (Bertsekas & Tsitsiklis, 1996). In many problems, even if such an explicit, mathematical model cannot be constructed, the system can be simulated either directly or via a generative model (it is often easier to build a generative model than an explicit MDP model of system dynamics; TD-gammon of Tesauro (1994) or Tetris of Böhm, Kókai, and Mandl (2005) show that it may be useful to learn the policy directly from simulation, that is, using a generative model of the environment, without considering the MDP model of system dynamics). This idea of simulation-based dynamic optimisation is known as *learning by reward and punishment* in the artificial intelligence literature (Sutton & Barto, 1998).

The first approach to learn from simulation is to estimate the missing model of the environment, i.e., $T(s, a, s')$ and/or $R(s, a, s')$, using, for example, statistical techniques. The repeated simulation is used to approximate or average the model. Once such an estimation of the model is available, standard techniques for solving MDPs, like policy and value iteration, are again applicable. This approach is known as model-based RL (Sutton, 1990).

An alternative approach can be of potential interest when the domain is huge and the approximate solution is satisfactory. The so called model-free RL algorithms do not attempt to estimate the model of the environment. Instead they directly estimate the value function or a policy (Ng & Jordan, 2000) from repeated simulation. In this way huge state spaces can be tackled, since there is no need to estimate the model for such a number of states (Bertsekas, 2007). These algorithms can be based on so called temporal difference updates to propagate information about values of states, $V(s)$, or state-action pairs, $Q(s, a)$. These updates are based on the difference of the two temporally different estimates of a particular state or state-action value. Model-free SARSA is such a method (Sutton & Barto, 1998). It updates state-action values by the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]. \tag{2}$$

It modifies the value of taking action a in state s, when after executing this action the environment returned reward r, moved to a new state s', and action a' was chosen in state s'.

4. Reward shaping

When the agent is learning from simulation, the immediate reward r which is in the update rule given by Eq. (2) represents the (only) feedback from the environment. The idea of reward shaping is to provide an additional reward which will improve the performance of the agent. This shaping reward does not come from the environment. It is extra information which is incorporated by the designer of the system and estimated on the basis of knowledge of the problem. The concept of reward shaping can be represented by the following formula for the SARSA algorithm:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + F(s, a, s') + \gamma Q(s', a') - Q(s, a)],$$

where $F(s, a, s')$ is the general form of the shaping reward which in our analysis is a function $F : S \times S \rightarrow \mathbb{R}$.

Depending on the quality of the shaping reward, it can decrease the time the algorithm spends attempting suboptimal actions. This decrease is the main aim of applying reward shaping. Ng et al. (1999) defined formal requirements on the shaping reward. In particular, the optimal behaviour of the (model-free) agent is left unchanged if and only if the shaping reward is defined as a difference of some potential function Φ of a source state s and a destination state s' (see Eq. (1)). This can be further clarified in the following way. When one has certain knowledge about the environment (knowledge which may help decrease the number of suboptimal actions the agent will attempt during learning), this knowledge can be used in different ways. In some cases the Q-table can be simply initialised based on this knowledge. The theoretical work of Ng et al. (1999) proved that if instead of initialising the Q-table, the same knowledge is used as a shaping reward, the final solution of the agent will not be changed. One of the most important implications of this fact, is that it allows for a straightforward use of background knowledge in RL with function approximation. It is not an obvious task of how to use existing heuristics to initialise the Q-table which is represented — for example, as a multi-layer neural network. The fact that reward shaping can be equivalent allows for a straightforward use of background knowledge in such cases. Heuristic knowledge can be easily given via reward shaping even when the function approximation with multi-layer neural networks is used for function approximation. In the case of neural networks with global basis functions (Bishop, 1996) the use of reward shaping instead of Q-table initialisation (assuming that such an initialisation could be done easily) would have additional advantages. The consistent reward shaping would be given all the time during the learning process, whereas initialised values would change rapidly during temporal-difference learning.

The work of Ng et al. (1999) and their requirement of potential-based shaping rewards applies to model-free algorithms like Q-learning or SARSA (Sutton & Barto, 1998). Recently Asmuth, Littman, and Zinkov (2008) gave theoretically grounded conditions on the potential function Φ for a prototypical model-based algorithm R-max (Brafman & Tennenholtz, 2002). In particular, Asmuth et al. (2008) proved that the R-max algorithm with potential-based reward shaping preserves its properties; i.e., is PAC-MDP (Strehl, Li, & Littman, 2006) if the shaping function is admissible. The notion of admissibility here is similar to the meaning of admissible heuristic functions in informed search (Russell & Norvig, 2002). The shaping function is said to be admissible in the context of the R-max algorithm if $\Phi(s) \geq \max_a Q(s, a)$, that is, the shaping reward never underestimates the reward (i.e., never overestimates the cost (Russell & Norvig, 2002)). The notion of admissibility naturally extends to the value function, for example, any arbitrary value function $V'(s)$ can be said to be admissible if $V'(s) \geq \max_a Q(s, a)$.

The theoretical work discussed in the two preceding paragraphs specifies how to apply reward shaping in both model-free and model-based settings. In particular, requirements on the potential function Φ are determined. But, this work does not specify how to obtain such knowledge and how to represent it as a heuristic function. The main focus of this paper is how to approximate the heuristic function online when it is not available or cannot be specified manually, and when (according to the theoretical requirements) reward shaping is defined as the difference of the potential function Φ of consecutive states s and s' (see Eq. (1)). This reduces to the problem of how to learn the potential function Φ. In particular two solutions are introduced in this paper. The first one is designed for model-free RL. It is based on the multigrid discretisation of the state space. The agent learns the target, ground value function with desired resolution, and at the same time it also learns another value function with lower resolution which is used as an estimate of the potential function for shaping the ground learning. It is worth noting that a particularly convenient potential function would be the one which is equal to the value function, that is, $\Phi(s) = V(s)$, which helps justify why roughly approximating the value function is a promising approach for reward shaping (Ng et al., 1999). The second approach is designed for model-based RL and is based on the free space assumption which is used in algorithms for learning heuristics in real time (LRTA*-type of algorithms Rayner, Davison, Bulitko, Anderson, & Lu, 2007). The free space assumption deals with initial uncertainty assuming that all actions in all states are unblocked. In our algorithm the statistical model of the underlying MDP, which is used for learning a potential function (heuristic), is initialised according to the free space assumption and is being further improved during repeated simulation. Before each replanning step of the R-max algorithm the potential function also undergoes replanning (value iteration) and after that it is used as a potential function for the R-max algorithm. Section 5 contains an introduction and the evaluation of the first approach and Section 6 treats the model-based case.

5. Learning a potential function with multigrid discretisation

In this section an algorithm to learn a potential function for reward shaping in model-free RL is presented. Firstly the state aggregation is explained. Next the discussion of the new algorithm is given followed by its empirical evaluation.

5.1. State aggregation

State aggregation represents one of the options for tackling the state space explosion which arises in sequential decision making, particularly when the mathematical model of the environment needs to have the Markov property (Boutilier, Dean, & Hanks, 1999). It is based on aggregating states of the original MDP into clusters, and treating these clusters as states of a new MDP. Such an MDP can be solved easier since its state space can be significantly reduced. States of the ground MDP are grouped according to a given notion of similarity, and, in the ideal case, states within one cluster should be indistinguishable with regard to the policy (e.g., all states which are in the same abstract state have the same optimal action). When states within aggregates are distinguishable in this sense, such aggregation is approximate. The fact, to what extent the states which belong to one cluster differ, determines the quality of the approximate representation. The trade-off between high quality approximation and tractability of solving an MDP needs to be considered. In this section we propose a RL architecture in which the original RL problem can be represented with the desired precision (i.e., the algorithm can reach the optimal solution) whereas the high level approximation (even though of a low quality) can improve the convergence of ground learning.

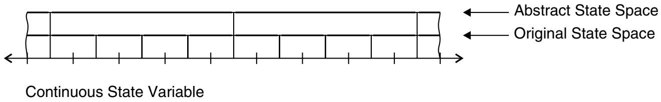

Fig. 1. Multigrid discretisation of the continuous domain variable.

In this section we investigate the particular case when the continuous state space is discretised to form the discrete state space for which the value function can be represented in a tabular form. The state aggregation for high level learning can be easily obtained by applying discretisation with bigger intervals (see Fig. 1). Low level discrete states are used to learn the actual solution to the original problem and learning on abstract states with a rough approximation provides useful guidance.

5.2. Learning a potential function for reward shaping

We propose a RL architecture with two different discretisations of the state space. The first one is to learn an approximation of the Q-function at the ground RL level. The second one which has lower resolution is to represent an abstract V-function which is used as the potential function to calculate the shaping reward (see Eq. (1)) for the ground level. The algorithm which is proposed here builds on two techniques existing in the field: (1) multigrid discretisation used with MDPs (Chow & Tsitsiklis, 1991) and (2) automatic shaping which was recently proposed (Marthi, 2007).

The multigrid discretisation in the MDP setting (Chow & Tsitsiklis, 1991) was used to solve MDPs in a coarse-to-fine manner. While this technique is well suited to dynamic programming methods, (a coarse problem can be solved and used as a starting point for the solution on a finer grid) there was no easy way to merge layers with a different resolution when applied to RL algorithms. First such attempts were made in (Anderson & Crawford-Hines, 1994) and this problem was evident there. The need for knowledge of the topology of the state space is necessary in their solution to define how multiple levels are related, but this fact made this approach infeasible for RL tasks. It used a multigrid as a way of obtaining knowledge, but the mechanism to use this knowledge at a ground RL level was missing. We propose potential-based reward shaping as a solution to these problems. The ground RL algorithm does not have to be modified and knowledge can be given in a transparent way via reward shaping.

In the automatic shaping approach (Marthi, 2007) an abstract MDP is formulated and solved. In the initial phase of learning, the abstract MDP model is built and, after a defined number of episodes, the abstract MDP is solved exactly and its value function used as the potential function for ground states. In this paper, we propose an algorithm which applies a multigrid strategy when a continuous state space is discretised to be represented in a tabular form. Instead of defining an abstract task as dynamic programming for solving an abstract MDP, we use RL to solve the high level task online. Because such an abstract RL does not need to learn the

Algorithm 1 SARSA-RS: Multigrid SARSA with potential-based reward shaping from the value function of aggregated states.

```
1:  $\forall s, aQ(s, a) = 0, \forall zV(z) = 0, t = 0$
2:  $z \leftarrow$ aggregating state for $s$
3:  $a \leftarrow$ random action in state $s$
4:  repeat
5:      $z' \leftarrow$ aggregating state for $s'$
6:      $t = t + 1$
7:      $r_v = reward_v(r)$
8:      if $r_v \neq 0$ or $z \neq z'$ then
9:          $V(z) = (1 - \alpha_v)V(z) + \alpha_v(r_v + \gamma_v^t V(z'))$;
10:         $t = 0$
11:     end if
12:     $F(s, s') = \gamma_v V(z') - V(z)$
13:     $z = z'$
14:     $a' \leftarrow$ best action in state $s'$
15:     with probability $\epsilon$: $a' \leftarrow$ random action in state $s'$
16:     $Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(r + F(s, s') + \gamma Q(s', a'))$
17:     $s = s'$; $a = a'$
18: until state $s$ is terminal
```

model, a shaping reward can be provided right from the start of learning. Additionally, the multigrid discretisation yields a natural translation between ground and abstract levels. Our method does not require any more knowledge about the environment than necessary to define discretisation at the ground level. We do not need methods to translate abstract to ground states or approximating environment dynamics (transition probabilities) at an abstract level. We propose an algorithm which, in contrast to the shaping approach of Marthi (2007), is entirely model-free, does not increase computational requirements, and is straightforward to apply with minimal knowledge about the domain.

Algorithm 1 summarises our approach. It follows the structure of SARSA from (Sutton & Barto, 1998). In our case learning at the ground level is the same as in the base-line. The modification is the point where basic SARSA is given the shaping reward $F(s, s')$ in Line 16 of Algorithm 1 where the temporal difference for ground states is computed. The way in which $F(s, s')$ is evaluated defines our extension.

The shaping reward $F(s, s')$ is computed in Line 12 as the difference of the value function of current and previous states visited by the agent. Thus $\Phi(s) = V(z)$ where $V(z)$ is the value function of the abstract level and state s is aggregated by given state z. The value $V(z)$ is learned using temporal difference updates (Line 9). All parameters that relate to this task have subscript v in the algorithm. The mapping from state s to a corresponding state z is done in a straightforward way without any special knowledge. Basically, abstract states z aggregate ground states s and subsumption can be easily determined. However with optional, additional knowledge about the problem, such a mapping can remove some state variables in the abstract representation and appropriately focus the high level learning.

High level RL is treated as a Semi-MDP (Sutton, Precup, & Singh, 1999) since, due to higher generalisation, an agent can be several time steps within one high level position. For this reason, time t is used when the temporal difference in Line 9 is computed.

The generic function $reward_v(r)$ shows that high level learning can receive an internally modified reward. The fact which may be considered when defining this function is that the shaping reward should not obscure the reward given by the environment because it may lead to suboptimal solutions when, for example, the potential function is based on non-admissible heuristic.

5.3. Experimental methodology

Algorithm 1 was evaluated empirically on the Mountain Car task with a continuous state space (Sutton & Barto, 1998). The following values of common RL parameters were used: $\alpha = 0.1$, $\alpha_v = 0.1$, $\gamma = 0.99$ and $\gamma_v = 0.99$. In all experiments ϵ-greedy exploration strategy was used with ϵ decreasing linearly from 0.3 in the first episode to 0.01 in the last episode. All runs were repeated 10 times and average results are presented in graphs. The x-axis represents the number of completed episodes, and the y-axis represents numerical rewards. Following the evaluation process from recent RL competitions, the accumulated reward (part b) of each figure) over all episodes was also used as a measure to compare results in a more readable way. Error bars of the standard error of the mean (SEM) are also presented in graphs for the accumulated reward.

To investigate how well the proposed algorithm scales to big problems, different discretisations for the low-level learning were applied. In this way, a range of problems with 231 to 10^4 states was obtained for the analysis. Such an empirical methodology was used in the related literature (e.g., by Wingate and Seppi (2005)) and allows for better understanding of properties of evaluated algorithms, especially scalability to bigger problems.

The experiments were performed on the Mountain Car task according to the description in (Sutton & Barto, 1998). The agent received a reward of 1 upon reaching the goal state on the right hill and -1 on the left hill. Function $reward_v(r)$ returned zero for negative r and an unchanged value for positive r. An experiment was terminated and the agent placed in a random position after reaching any of the two goal positions or after 10^3 episodes. Three different discretisations were used with 11×21 (these smallest values were taken from (Epshteyn & DeJong, 2006)), 40×40 and 100×100 intervals on correspondingly the position and velocity of the car. For these discretisations, high level states aggregated 4, 8 and 20 ground states accordingly.

5.4. Results

The results with three discretisations introduced in the previous section are in Figs. 2–4 respectively. As these problems gradually become more difficult, the number of episodes was increased in more difficult versions, leading to numbers 10^4, $2 \cdot 10^4$, and $5 \cdot 10^4$.

The most essential observation which supports the significance of Algorithm 1 is that the bigger the instance of the problem, the bigger the positive influence this algorithm has. Particular comparisons can be read from the graphs. On the easiest task the accumulated reward of SARSA is 1500 and SARSA-RS 2500 after the same number (10^4) of episodes (Fig. 2b). On the 40×40 version when SARSA obtains 1500, SARSA-RS has already scored 4500 after the same number (in this case $2 \cdot 10^4$) of episodes (Fig. 3b). The most difficult 100×100 version shows further improvement of reward shaping (Fig. 4b). When SARSA obtains 1500, SARSA-RS gains about 7200 after the same number (about $4.7 \cdot 10^4$) of episodes. This relationship can be also found, though in a less clear way, in graphs presenting average reward per episode in the (a) part of each figure. This observation clearly shows that the positive influence of our algorithm is more evident when the problem is more difficult (in terms of the number of states).

Learning becomes more difficult when the problem becomes bigger, but the advantage of SARSA-RS when compared with pure SARSA is significant in all cases. Error bars placed in graphs for the accumulated reward indicate that this difference is statistically significant.

6. Learning a potential function with the free space assumption

In this section an algorithm to learn a potential function for reward shaping in the model-based R-max algorithm is introduced. Since the method suggested in this section to learn the potential function is specific to particular properties of the R-max algorithm,

Please cite this article in press as: Grześ, M., & Kudenko, D. Online learning of shaping rewards in reinforcement learning. Neural Networks (2010), doi:10.1016/j.neunet.2010.01.001

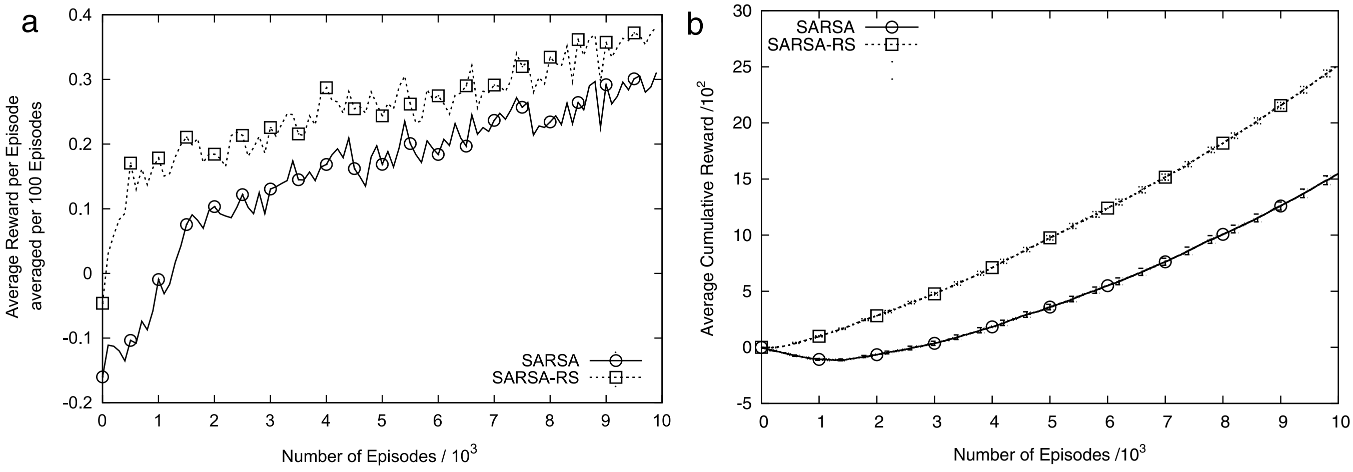

Fig. 2. The average reward per episode (a) and the average cumulative reward (b) on the Mountain Car task with discretisation 11 × 21 and aggregation of 4 states.

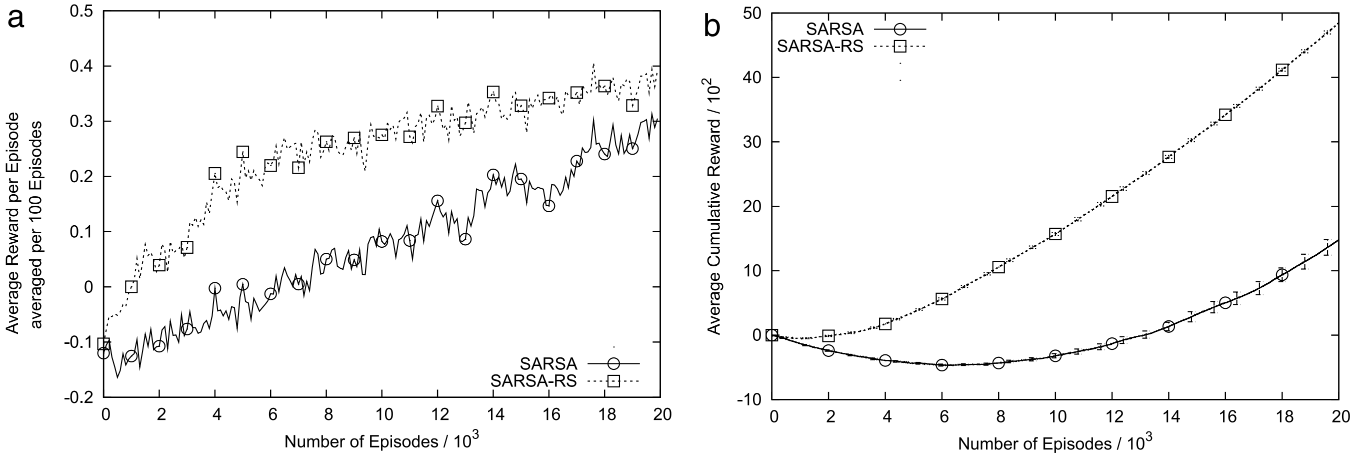

Fig. 3. The average reward per episode (a) and the average cumulative reward (b) on the Mountain Car task with discretisation 40 × 40 and aggregation of 8 states.

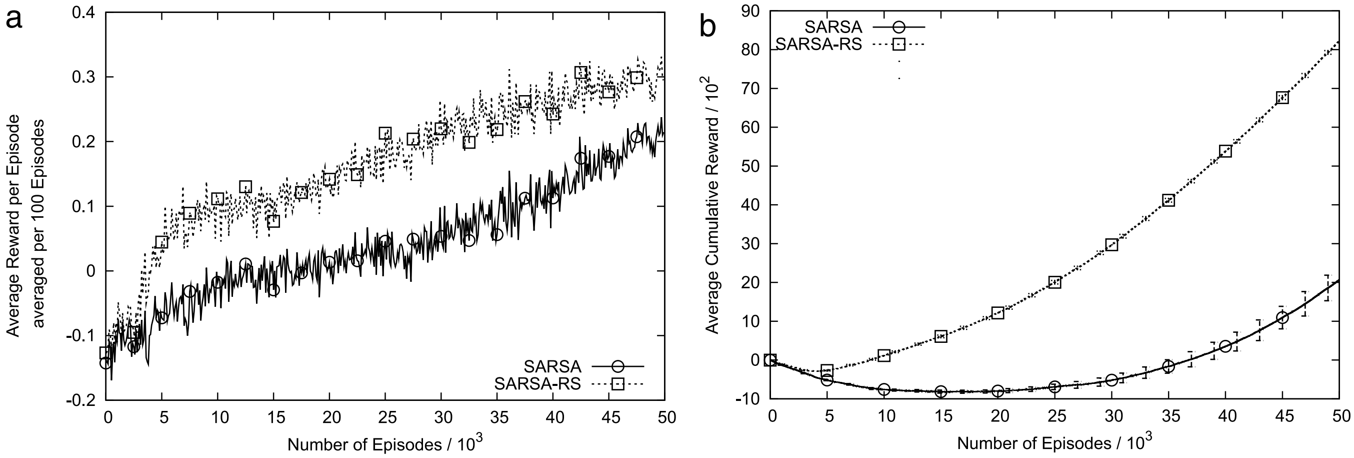

Fig. 4. The average reward per episode (a) and the average cumulative reward (b) on the Mountain Car task with discretisation 100 × 100 and aggregation of 20 states.

and also attempts to preserve the theoretical convergence properties of this algorithm, the introduction to our method starts with an explanation of how the R-max algorithm works. Next, the free space assumption is explained. After that, our approach to learn reward shaping for the R-max learning is discussed, and the empirical evaluation of the proposed method is presented in the final part of this section.

6.1. R-max and reward shaping

The R-max algorithm is a prototypical model-based algorithm which is PAC-MDP (Strehl et al., 2006), that is, the number of suboptimal steps is bounded polynomially by relevant parameters. The theoretical properties of this algorithm, the fact that it is becoming popular in the reinforcement learning literature, and

Please cite this article in press as: Grześ, M., & Kudenko, D. Online learning of shaping rewards in reinforcement learning. Neural Networks (2010), doi:10.1016/j.neunet.2010.01.001

also that it has been recently analysed with regard to reward shaping (Asmuth et al., 2008), makes it a good candidate for our analysis. In particular, our approach proposed in this paper is in accordance with the theoretical assumptions of R-max learning. The R-max algorithm applies the approach of optimism under uncertainty. Specifically, it initially assumes that all state-action pairs lead deterministically to the goal state, and the maximum value of the reward, R-max (i.e., R-max $= \max_{(s,a,s')} r(s,a,s')$ when $\gamma = 1$, or R-max $= \max_{(s,a,s')} r(s,a,s')/(1-\gamma)$ when $\gamma < 1$), is given after each transition in this model. For dynamics defined in this way, the value function is computed using value iteration:

$$Q(s,a) \leftarrow \sum_{s'} R(s,a,s') + \gamma \sum_{s'} [T(s,a,s') \max_{a'} Q(s',a')]. \qquad (3)$$

After that, the agent acts in the environment (real or simulated) following the policy defined by the current value function. When samples are drawn from the environment, the agent improves its estimation of the model. Specifically, for each tuple $\langle s, a, s', r\rangle$ obtained from the environment, the agent increases counters $c(s,a,s') \leftarrow c(s,a,s') + 1$ and $t(s,a,s') \leftarrow t(s,a,s') + r$. The precision of the R-max algorithm is regulated through the parameter m which depends on specific figures which define the precision of the method. The higher the value of m the more precise the model is and the more accurate value function calculation according to this model (Brafman & Tennenholtz, 2002). If, for a given state-action pair $\sum_{s'} c(s,a,s') \geq m$, the initial model for particular state-action pair is replaced by $T(s,a,s') = c(s,a,s')/\sum_{s''} c(s,a,s'')$ and $R(s,a,s') = t(s,a,s')/c(s,a,s')$. The value iteration (Eq. (3)) is performed every time a new state-action pair becomes known, that is, when its $\sum_{s'} c(s,a,s')$ reaches the value of m. When no new state-action pairs become known, the algorithm follows a near-optimal policy which is represented by the current value function obtained from the last planning step (Eq. (3)).

The potential-based reward shaping for both model-free and model-based RL was introduced in Section 4. The planning step of the R-max algorithm can have the following form when this type of reward shaping is used:

$$Q(s,a) \leftarrow \sum_{s'} R(s,a,s') + F(s,s') \\ \gamma \sum_{s'} [T(s,a,s') \max_{a'} Q(s',a')], \qquad (4)$$

where $F(s,s')$ is computed according to Eq. (1), that is, it is a difference of the value of the potential function Φ of states s' and s. Thus, in this section (analogically to the previous section) our aim is to learn the potential function Φ. According to the introduction in Section 4, we have to take into account the fact that the potential function has to be admissible if we want to preserve convergence properties of the R-max algorithm. The following subsection aims at introducing the algorithm to learn this function online, that is, at the same time, as the actual R-max learning takes place. The free space assumption (Rayner et al., 2007) is used in this algorithm.

6.2. Free space assumption and learning a potential function

The free space assumption (FSA) is an approach to define an initial model of the environment which assumes that all transitions in the environment are possible (in navigation robotic environments it would assume that there are no walls between all adjacent states), and that all actions are deterministic and always lead to the expected state, that is, a state which has the highest probability in the actual stochastic model of the environment. Thus, this approach assumes that all actions always lead to their expected effects (Rayner et al., 2007). In the hypothetical robotic environment, it would mean that an action move forward, always moves the robot from a given state to the state in front of the robot, ignoring any existing walls and probabilistic effects of actions like, for example, slippery surface which would slow down the robot's movement or change the direction of its motion. A part of the state space with stochastic actions, and blocked transitions between neighbouring states is shown in Fig. 5.

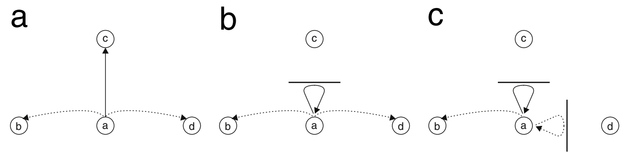

Fig. 5. Probabilistic actions in a stochastic environment. Solid lines show expected effects of actions, and dashed lines show low-probability unexpected failures. Solid lines between circular states reflect no connectivity (walls) between those states.

An important property of the model defined according to the free state assumption is that it leads to an admissible value function when compared to the value function which corresponds to the true model and when the same reward is used in both cases. Since the probabilistic effects and obstacles are ignored, the value function computed with this model is always higher than the true value. Thus, the value function of the initial FSA model can be seen as a first option to provide an admissible potential function to the R-max algorithm with reward shaping. In domains where the admissible heuristic cannot be (easily) defined manually, the use of the free space assumption yields a potential solution to the problem of determining the admissible heuristic.

The initial FSA model can be defined before the agent starts its interaction with the environment. The key idea of our algorithm which is presented in the next subsection is that this initial FSA model can be further revised, using the same experience (i.e., tuples $\langle s, a, s', r\rangle$ obtained from the environment) as the basic R-max algorithm. The potential problem of this idea is that the straightforward incremental modification of the FSA model may lead to changes which will make this model non-admissible, that is, the value function computed according to this model would not be admissible. A more detailed discussion of this issue together with a solution is in the next subsection where the full algorithm is presented.

6.3. Learning a potential function for R-max

In this subsection a new approach to R-max learning with reward shaping is proposed. The main aim is to obtain the R-max algorithm with reward shaping which can be used when the admissible heuristic cannot be specified manually. The approach based on the free space assumption is presented in Algorithm 2. It is a standard R-max solution enhanced by the mechanism to learn a potential function Φ.

In the first instance, the models of the environment are initialised. In Line 1 the R-max model is initialised according to the description of the standard R-max algorithm in Section 6.1, and in Line 2 the initial FSA-model is created according to Section 6.2. The next two lines use these models to compute the potential function Φ and the value function of the R-max algorithm. The VI method implements standard value iteration (Eq. (3)) and VI-RS is with reward shaping according to Eq. (4). Since VI-RS requires the potential function Φ to compute $F(s,s')$ in Eq. (4), Φ has to be computed before VI-RS is executed. In Line 6 the current state s is initialised to the start state, and the algorithm enters the main loop after that.

The first step of each iteration of the main loop is to decide on the action for the current state. The current value function is used to make this decision, that is, an action with the highest value is always chosen (ties are broken randomly). In Line 9 the

Algorithm 2 RS-FSA: The R-max algorithm with online learning of potential-based reward shaping via the free space assumption.

```
1: R-model ← initialise the R-max model
2: FSA-model ← initialise the FSA model
3: $\Phi \leftarrow VI(FSA\text{-}model)$
4: R-value-function ← VI-RS(R-model, $\Phi$)
5:
6: $s \leftarrow$ the start state
7: repeat
8:     $a \leftarrow$ choose an action using R-value-function
9:     $(s', r) \leftarrow$ execute $a$ in state $s$
10:    $known \leftarrow$ R-model.observe($s, a, s', r$)
11:    FSA-model.observe($s, a, s', r$)
12:    if $known$ then {a new state became known in the R-max model}
13:        $\Phi \leftarrow VI(FSA\text{-}model)$
14:        R-value-function ← VI-RS(R-model, $\Phi$)
15:    end if
16:    if for the first time $\neg known$ in at least $k$ episodes then
17:        FSA-model ← increase FSA transitions
18:        $\Phi \leftarrow VI(FSA\text{-}model)$
19:        R-value-function ← VI-RS(R-model, $\Phi$)
20:    end if
21:    if $s'$ is terminal then
22:        $s \leftarrow$ the start state
23:    else
24:        $s \leftarrow s'$
25:    end if
26: until terminal condition
```

sampling from the environment takes place, and the next state and the corresponding reward are returned by the environment. This experience tuple is used for updating models (the R-model in Line 10 and the FSA-model in Line 11). The update of the R-model is according to the description in Section 6.1, and the method *observe* of this model returns *true* when the executed update made the state-action pair (s,a) known to this model, that is, $\sum_{s'} c(s, a, s')$ reached the value of m for the first time. Updates of the FSA-model increase corresponding counters.

As was discussed in Section 6.1, the planning steps of the R-max algorithm are executed only when a new state-action pair becomes known. For this reason this fact is checked in Line 12, and re-planning takes place in Lines 12–15. After that the current state can be updated in Lines 21–25, and the algorithm continues to the next iteration of the loop.

The piece of code in Lines 16–20 has been omitted on purpose in the previous discussion since it requires a more detailed explanation. The problem which should be discussed here stems from the fact that the way the FSA-model is updated in Line 11 – that is, by increasing counters for the corresponding observation tuple – may lead to the value of Φ which is not admissible. The probability of this situation is very low (because of the prior for FSA-transitions specified according to the free space assumption — for details see Section 6.2), however it can be the case that the model which is obtained is not admissible. It may again become admissible once more observations will be sampled and used to update the model. To deal with this problem the code in Lines 16–20 was added. The following reasoning supports this. The problem which is tackled here stems from the fact that the estimate of dynamics for a particular state-action pair may be not admissible when the number of observed tuples is small. However, the theoretical properties of the R-max algorithm allow one to accept the model for a particular state-action pair when it was experienced at least m times. If the value of m is sufficiently high (according to relevant parameters which define the precision of the final result) the estimate of transition dynamics are considered to be sufficiently accurate. The algorithmic trick which we propose here is to re-initialise the counters of the FSA-model when the algorithm stabilises. More precisely, during the initial phase of learning, the FSA-model is being updated just by increasing corresponding counters. When the algorithm stabilises – that is, there are no more planning steps over a specified number of episodes k – the FSA-model is re-initialised in a special way. The re-initialisation takes place only for those state-action pairs for which $\sum_{s'} c(s, a, s') < m$, that is, for those pairs which have not been experienced well enough to be considered as known in the sense of the R-max m parameter. If such pairs are found in the FSA-model, then counters of those transitions which follow the free space assumption (i.e., those which were initially initialised to 1 in Line 2) are increased by the value of $m - \sum_{s'} c(s, a, s')$. In this way they become known in the sense of the R-max m parameter, and furthermore they will also lead to an admissible potential function Φ when compared with the value function of the R-max algorithm. The potential function Φ is estimated again after this step (Line 17), and the value function computed again using Φ for reward shaping (Line 18). If the previous potential function was not admissible, the new value function will lead to additional exploration of the state space, since the new potential function Φ is surely admissible in the sense of the value m of the R-max algorithm.

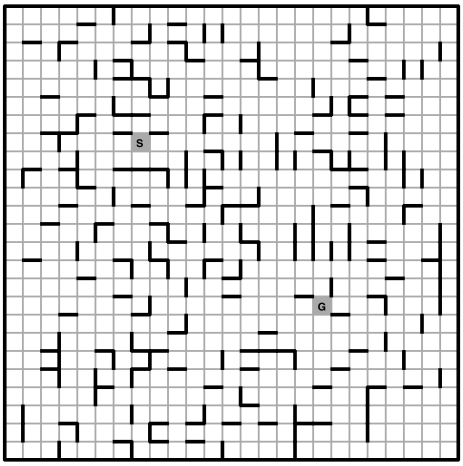

Fig. 6. The stochastic navigation maze domain.

6.4. Experimental methodology

Algorithm 2 was evaluated empirically on the navigation maze task that is shown in Fig. 6. It is a stochastic domain. Each action can result in its expected outcome with probability 0.8, and slip into one of two perpendicular directions with probability 0.1 for each of these directions (see Fig. 5a). The reward of −0.01 is given for the execution of each action. For the transition to the goal state G, the agent receives additional reward of the value of 1. The start state is marked with S. Blocked transitions (walls) between states are marked as solid lines between corresponding states. The learning problem in this environment is formulated as follows. The RL agent has to learn the highest reward path from the start state S to the goal state G without knowing in advance transition probabilities of the environment. Without much loss of generality the reward model is assumed be known to the agent, what is commonly assumed in the relevant literature (Asmuth et al., 2008).

The following values of parameters were used: $m = 5$, and $k = 100$. The value of 100 for parameter k was to ensure that the re-initialisation of the FSA-model is done when the exploration of the algorithm stabilises. The value of 1 was assigned to the MDP discount factor γ. Experiments were conducted on a number of algorithms:

1. R-max - the standard version of the R-max algorithm.

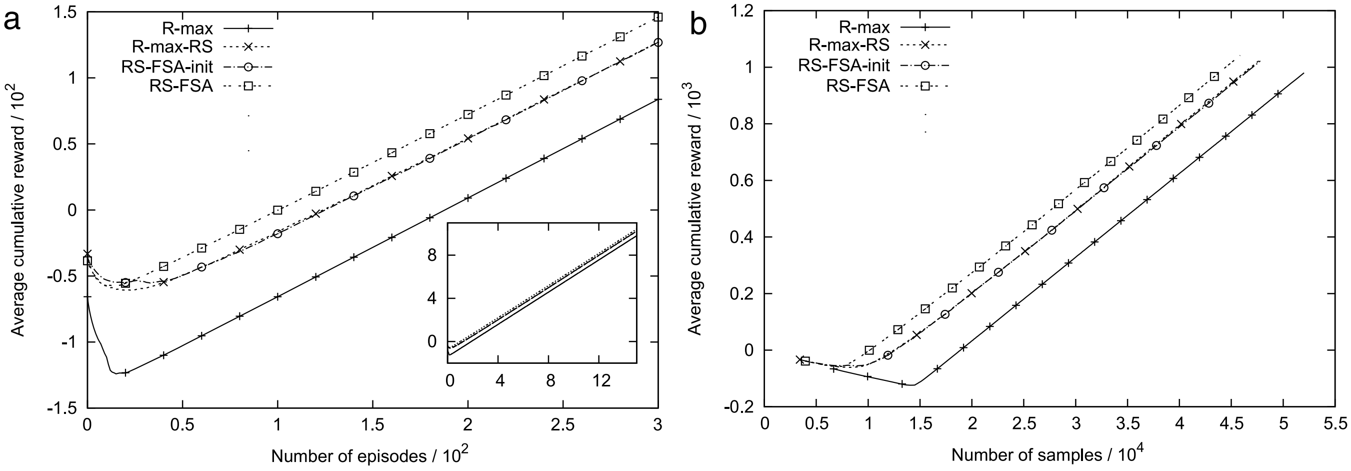

Fig. 7. The cumulative reward plotted as a function of: (a) the number of learning episodes, and (b) the overall number of samples.

2. R-max-RS - the R-max algorithm with reward shaping according to Asmuth et al. (2008), the potential function was evaluated as $\Phi(s) = r_s \times d(s)/\varrho + r_g$, where $r_s \leq 0$ is the step reward, $d(s)$ is the shortest straight-line distance from state s to the goal G, ϱ the probability of the expected outcome of the action, and r_g the reward given when the goal state is reached.
3. RS-FSA-init is Algorithm 2 in which the initial FSA-model is used during the entire learning process.
4. RS-FSA stands for the full version of the Algorithm 2, that is, with refinement of the FSA-model, recalculation of the potential function Φ and final re-initialisation to ensure an admissible potential function with regard to the parameter m.

In all graphs in this section all evaluations were computed for 30 runs of all algorithms. Similarly to the previous section we plotted the cumulative score of each algorithm as a function of the number of episodes. Additionally, in this analysis the cumulative reward is drawn also as a function of the overall number of samples. This is important here, since the R-max type of learning is aimed at sample complexity reduction (Kearns & Singh, 2002). The error bars of the standard error of the mean (SEM) (Cohen, 1995) were also calculated and plotted. However the error range was very small in this experiment, and thus each interval resulted in a single point on our graphs, which means that presented curves have very low variance and indicates that differences in obtained results are statistically significant. For this reason error bars are not present in the final version in Fig. 7.

6.5. Results

Fig. 7 shows the cumulative reward obtained by the four learning algorithms. The number of learning episodes was 1500. For better readability of results, Fig. 7a shows the cumulative reward in the first 300 episodes, the full range of episodes is additionally placed in the bottom-right corner of this figure.

The R-max algorithm without reward shaping has the lowest learning performance. R-max-RS which uses the hand-coded heuristic based on the straight line distance to the goal shows significant improvement. This kind of improvement is generally expected, and this kind of an admissible heuristic can be designed manually by a human. Our goal in this paper was to tackle the problem of reward shaping when such a heuristic cannot be easily designed, and thus our goal was to perform better then pure R-max but not necessarily better than a good, hand-coded, admissible heuristic. It turned out that already the RS-FSA-init version of our algorithm performed as well as R-max-RS. Curves for these two algorithms overlap after around 50 episodes of learning. The further refinement of the FSA-model in the RS-FSA algorithm and the use of this model to compute the new potential function Φ online resulted in further improvement. The full range of episodes in the bottom-right corner of Fig. 7a shows additionally that all lines are ideally parallel in the final period of learning, which means that all algorithms reach the same asymptotic convergence.

The R-max algorithm is intended to reduce the sample complexity (this can be a critical issue of applying a RL solution when sampling from the real environment is costly). For this reason in Fig. 7b the cumulative reward is also presented as a function of the overall number of samples. When the full range of episodes is concerned, the advantage of reward shaping solutions is more evident in Fig. 7b than in Fig. 7a. It means that even though reward shaping leads to a more rapid improvement in terms of the number of learning episodes, the relative difference is more significant in terms of the number of samples. This relative difference can be identified when the full experiment is compared, that is, Fig. 7b is compared with the bottom-right part of Fig. 7a. Curves are much further apart in Fig. 7b which shows a bigger difference.

Overall, the approach proposed in this section to learn the potential function for reward shaping online was shown to be successful, and can be considered when the heuristic function cannot be designed manually. Furthermore, the proposed approach is also competitive to hand-coded heuristics and, in the face of obtained results, can also be considered when such a heuristic is available.

7. Related work

In Algorithms 1 and 2 learning takes place at two levels of abstraction/model, and so it is worth relating this approach to the general concept of hierarchical machine learning. Stone and Veloso (2000) proposed the universal idea of layered learning where the search space of hypotheses can be reduced by a bottom-up, hierarchical task decomposition into independent subtasks. Each local task is solved separately, and tasks are solved in a bottom-up order. The distinguishing feature of this paradigm is that the learning processes at different layers do not interact with each other and different machine learning algorithms can be used at different layers. In particular, RL was applied to learn in this architecture (Stone & Veloso, 2000); i.e., to learn at a particular layer. Because tasks are solved independently using results from learning at lower layers, the algorithms proposed in our paper can be seen as a potential choice for selected subtasks.

When relating Algorithms 1 and 2 to hierarchical reinforcement learning it is worth taking note of how the hierarchy interacts with reinforcement learning in such algorithms. Regardless of the type of abstraction used to create a hierarchy (e.g., state abstraction, hierarchical distance to the goal (Kaelbling, 1993; Moore, Baird, & Kaelbling, 1999), feudal reinforcement learning (Dayan & Hinton, 1993), temporal abstraction (Parr & Russell, 1997; Sutton et al., 1999), or both state and temporal abstractions (Dietterich, 2000)), the hierarchy exists in the final representation of the solution; i.e., the policy is defined on this hierarchy, and learning may take place at all levels of the hierarchy simultaneously. The value function is a function of not only the ground states and actions but also some elements determined by the hierarchy (e.g., in Parr & Russell, 1997) HAMQ-learning maintains an extended Q-table Q([s,m],a) indexed by the pair of states, which includes environment state s, and machine state m, and an action a at a choice point. In Algorithms 1 and 2 the actual RL is not modified and the high level learning provides feedback which is given in a transparent way via reward shaping. There is also no need for knowledge about the hierarchal task decomposition, as in the basic case the knowledge which is used to design the state representation is sufficient to deploy Algorithm 1, and knowledge which allows defining the free space transitions to apply Algorithm 2. In particular they can be applied when the hierarchical task decomposition is not possible or difficult to distinguish or incorporate into the solution.

A related approach to learning the potential function for reward shaping was investigated by the authors of this paper in Grzes and Kudenko (2008b). In this case, symbolic knowledge which is represented in the form of STRIPS operators is used to create a high level symbolic plan (Ghallab et al., 2004). The symbolic plan is then translated into the potential function for reinforcement learning. This approach is effective in providing useful guidance to the learning agent when the high level symbolic representation can be identified. Such a representation should allow creating STRIPS operators and defining the planning problem. The work presented in the current paper does not have this requirement and can be applied to a broader range of scenarios.

8. Conclusion

Reward shaping is a powerful technique to incorporate background knowledge into RL agents. One problem with this approach is that often detailed knowledge of the potential function of states is not available, or very difficult to represent directly in the form of a shaped reward. For this reason, this paper discusses solutions which allow applying the potential-based reward shaping when the potential function cannot be defined manually.

Two algorithms are proposed in this paper. The first one is to learn the potential function in model-free RL, and the second one in the model-based RL. Both these algorithms apply the same paradigm of estimating the value function at an abstract level and the use of this value function as a potential function for the actual learning task.

The first algorithm, which is designed for model-free RL and is described in Section 5, is based on the multi-grid discretisation of the state space. The state space with higher generalisation is used to learn a high level value function which is treated as a potential function for reward shaping of the actual learning process. The theoretical and empirical analysis of this approach can be summarised as follows:

- simultaneous learning of the potential function and the actual policy in the model-free setting can converge to a stable solution,
- the algorithm proposed in this paper may yield significant improvement of the convergence rate, when the bigger instance of the problem is considered with a fine grained policy (i.e., based on a more expressive representation) to be learned,
- improved convergence speed is achieved at a low computational cost, because there is at most one backup of the V-function for each SARSA backup,
- a separate and external representation of knowledge is obtained and this high level knowledge may be useful for knowledge transfer (Taylor & Stone, 2005),
- no need for explicit domain knowledge; in the basic form the high level learning can be defined using the same knowledge which is used to design the state representation at the ground level,
- in general, the application of this algorithm can be considered whenever state aggregation or, in other words, grouping of original states, can be applied,
- in domains with structural dependencies between groups of state variables, the high level representation may additionally ignore some state variables which could lead to tremendous improvement of reward shaping in this algorithm.

The second approach, which is for model-based RL, applies the free space assumption to create and refine the model of environment dynamics for learning the potential function. The theoretical and empirical analysis of this approach can be summarised as follows:

- the theoretical properties of the underlying R-max algorithm are entirely preserved by the proposed solution,
- the algorithm is specifically useful when the admissible heuristic cannot be easily created and the initial model which is based on the free space transitions can be identified,
- according to the obtained results, the use of this algorithm can be considered even when a good heuristic can be designed manually, as our approach can further improve learning with reward shaping,
- our algorithm leads to the reduction of the sample complexity which follows the main practical aim of the model-based R-max algorithm, that is, the aim of maximal sample complexity reduction of the learning algorithm (Brafman & Tennenholtz, 2002; Kearns & Singh, 2002).

The final discussion of the paper aims at analysing potential problems of the proposed approaches and helps the reader to judge their application potential in particular domains.

The technique proposed in Section 5 is applicable to the general range of RL problems with a very limited background knowledge. The algorithm attempts to learn the heuristic function from a reduced representation. Therefore, even though this approach is flexible, it is strongly dependent on the design of the high level representation. The process of designing a suitable abstraction for learning of reward shaping is a mixture of art and engineering. Both approaches presented in this paper aim at rapid learning of the heuristic function which can guide the actual learning process. The first question which should be answered at the beginning is what kind of a heuristic function is to be learned at the abstract level, i.e., what kind of information should it provide to the learning algorithm. When an answer to this question is available, the abstraction should contain those state features which will be sufficient to represent the required heuristic function. Failure in identifying all features which are correlated with the required behaviour or inclusion of irrelevant features will result in learning wrong information or a very slow learning of heuristics. The overall success depends on particular engineering design. However, in the case of complex RL domains with many sets of correlated features, more than one heuristic function could be learned and used for reward shaping.

Reinforcement learning with reward shaping could be named *informed reinforcement learning* in a similar manner as informed search is distinguished from uninformed search (Russell & Norvig, 2002). Informed approaches which are based on heuristics are designed to perform more directed search or learning. The overall improvement of the R-max algorithm with reward shaping can be therefore inferred in the same way as the improvement of the best-first search algorithm is predicted over the Dijkstra's algorithm which is a special case of the best-first search algorithm when the heuristic function is always zero. Overall, we can expect that R-max with reward shaping will work better in similar situations where best-first search works better than Dijsktra's algorithm, and this fact makes the general idea of reward shaping for R-max rather strong. Thus the main shortcoming of our approach to learning the shaping reward is the fact that knowledge about the goal state and the free space assumption is required. But, this knowledge is usually available when the heuristic function is not. With this knowledge, one can not only deal with situations when the heuristic function is not available but also gain further improvement over manually designed heuristics if they exist.

Acknowledgement

This research was sponsored by the United Kingdom Ministry of Defence Research Programme.

References

Anderson, C., & Crawford-Hines, S. (1994). Multigrid Q-learning. Technical report CS-94-121, Colorado State University.

Asmuth, J., Littman, M. L., & Zinkov, R. (2008). Potential-based shaping in model-based reinforcement learning. In *Proceedings of AAAI conference on artificial intelligence*.

Bertsekas, D. P. (2007). *Dynamic programming and optimal control (2 Vol Set)* (3rd ed.). Athena Scientific.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Athena Scientific.

Bishop, C. M. (1996). *Neural networks for pattern recognition*. Oxford University Press.

Böhm, N., Kókai, G., & Mandl, S. (2005). An evolutionary approach to tetris. In *Proceedings of the sixth metaheuristics international conference*.

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *11*, 1–94.

Brafman, R. I., & Tennenholtz, M. (2002). R-max — A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 213–231.

Chow, C. S., & Tsitsiklis, J. N. (1991). An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control*, *36*(8), 898–914.

Cohen, P. R. (1995). *Empirical methods for artificial intelligence*. MIT Press.

Dayan, P., & Hinton, G. E. (1993). Feudal reinforcement learning. In *Proceedings of advances in neural information processing systems*.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Epshteyn, A., & DeJong, G. (2006). Qualitative reinforcement learning. In *Proceedings of the 23rd international conference on machine learning* (pp. 305–312).

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated planning, theory and practice*. Elsevier, Morgan Kaufmann Publishers.

Grzes, M., & Kudenko, D. (2008a). Multigrid reinforcement learning with reward shaping. In *LNCS*, *Proceedings of the 18th international conference on artificial neural networks*. Springer-Verlag.

Grzes, M., & Kudenko, D. (2008b). Plan-based reward shaping for reinforcement learning. In *Proceedings of the 4th IEEE international conference on intelligent systems* (pp. 22–29). IEEE.

Gullapalli, V., & Barto, A. G. (1992). Shaping as a method for accelerating reinforcement learning. In *Proceedings of the 1992 IEEE international symposium on intelligent control* (pp. 554–559).

Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of international conference on machine learning* (pp. 167–173).

Kearns, M., & Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 209–232.

Konidaris, G., & Barto, A. (2006). Autonomous shaping: Knowledge transfer in reinforcement learning. In *The 23th international conference on machine learning*.

Marthi, B. (2007). Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th international conference on machine learning* (pp. 601–608).

Mataric, M. J. (1994). Reward functions for accelerated learning. In *Proceedings of the 11th international conference on machine learning* (pp. 181–189).

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Moore, A., Baird, L., & Kaelbling, L. P. (1999). Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs. In *Proceedings of the international joint conference on artificial intelligence* (pp. 1316–1323).

Ng, A. Y., Harada, D., & Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th international conference on machine learning* (pp. 278–287).

Ng, A. Y., & Jordan, M. (2000). PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of uncertainty in artificial intelligence* (pp. 406–415).

Parr, R., & Russell, S. (1997). Reinforcement learning with hierarchies of machines. In *Proceedings of advances in neural information processing systems*. Vol. 10.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York, NY, USA: John Wiley & Sons, Inc.

Randløv, J., & Alstrom, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th international conference on machine learning* (pp. 463–471).

Rayner, D. C., Davison, K., Bulitko, V., Anderson, K., & Lu, J. (2007). Real-time heuristic search with a priority queue. In *Proceedings of the 2007 international joint conference on artificial intelligence* (pp. 2372–2377).

Russell, S. J., & Norvig, P. (2002). *Artificial intelligence: A modern approach* (2nd ed.). Prentice Hall.

Stone, P., & Veloso, M. (2000). Layered learning. In *Proceedings of the 11th European conference on machine learning*.

Strehl, A. L., Li, L., & Littman, M. L. (2006). Pac reinforcement learning bounds for rtdp and rand-rtdp. In *Proceedings of AAAI workshop on learning for search*.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th international conference on machine learning* (pp. 216–224).

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

Sutton, R. S., Precup, D., & Singh, S. P. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1–2), 181–211.

Taylor, M. E., & Stone, P. (2005). Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the 4th international joint conference on autonomous agents and multiagent systems* (pp. 53–59).

Tesauro, G. J. (1994). TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, *6*(2), 215–219.

Wiewiora, E. (2003). Potential-based shaping and q-value initialisation are equivalent. *Journal of Artificial Intelligence Research*, *19*, 205–208.

Wingate, D., & Seppi, K. D. (2005). Prioritization methods for accelerating MDP solvers. *Journal of Machine Learning Research*, *6*, 851–881.