Learning Shaping Rewards in Model-based Reinforcement Learning

Marek Grzes and Daniel Kudenko
Department of Computer Science
University of York
York, YO10 5DD, UK
{grzes, kudenko}@cs.york.ac.uk

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 how to compute the potential 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 potential manually, the potential function can be learned online in parallel with the actual reinforcement learning process. 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. The novel algorithm is presented and evaluated empirically and theoretically. Specifically, the proposed algorithm is shown to learn an admissible potential which is required by the R-max algorithm with potential-based reward shaping.

Keywords

potential-based reward shaping, learning heuristic, reinforcement learning

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 what the best action in a given situation is. 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 and 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 and Barto, 1992, Konidaris and Barto, 2006, Mataric, 1994, Ng et al., 1999, Randløv and 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 be also misleading [Randløv and 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 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 and Norvig, 2002]. An algorithm to learn shaping rewards online is proposed in this paper. This algorithm is designed for model-based RL, and applies the free space assumption to create and refine the model of environment dynamics for learning the potential function. It specifies an abstract level value function which is used as a potential function for reward shaping. The model constructed according to the free space assumption is used to compute this value function online.

The remainder of this paper is organised as follows. Section 2 gives a brief overview of Markov Decision Process. Reinforcement learning and reward shaping are discussed in Sections 3 and 4. The main contribution of the paper is described in Sections 5, 6 and 7. The empirical evaluation is in Sections 8 and 9. The paper ends with a conclusion in

the final section.

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., 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 et al., 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 and the reward function. When such a model is not available, policy and value iteration can not 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 and Barto, 1998] or neuro-dynamic programming [Bertsekas and Tsitsiklis, 1996]. In many problems, even if such an explicit, mathematical model can not be constructed, the system can be simulated either directly or via a generative model (it is often easier to build a generative mathematical model than an explicit model of system dynamics [Tesauro, 1994]). This idea of simulation-based dynamic optimisation is known as *learning by reward and punishment* in the artificial intelligence literature [Sutton and Barto, 1998].

The first approach to learn from simulation is to estimate the missing model of the environment 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 does not attempt to estimate the model of the environment, and is called model-free RL. Algorithms of this type directly estimate the value function or a policy [Ng and Jordan, 2000] from repeated simulation. 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 and Barto, 1998]. It updates state-action values by the formula:

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

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 Equation 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 Equation 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 and Barto, 1998]. Recently Asmuth et al. [2008] gave theoretically grounded conditions on the potential function Φ for a prototypical model-based algorithm R-max [Brafman and 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 et al., 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 and 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 and 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 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 can not be specified manually, and when (according to the theoretical requirements) reward shaping is defined as the difference of potentials Φ of consecutive states s and s' (see Equation 1). This reduces to the problem of how to learn the potential $\Phi(s)$.

A novel algorithm is introduced in this paper. It 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 et al., 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 underlying MDP, which is used for learning potential (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 for the R-max algorithm. Since, the method suggested in this paper is specific to particular properties of the R-max algorithm, and also attempts to preserve the theoretical convergence properties of this algorithm, the introduction to our method starts with 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 presented and empirically evaluated.

5. R-MAX AND REWARD SHAPING

R-max 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 also that it has been recently analysed with regard to reward shaping [Asmuth et al., 2008], makes it a good candidate for our analysis. Particularly, 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., $R\text{-}max = \max_{(s,a,s')} r(s,a,s')$ when $\gamma = 1$, or $R\text{-}max = \max_{(s,a,s')} r(s,a,s')/(1-\gamma)$ 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:

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

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 $t(s,a,s') \leftarrow t(s,a,s') + r$. If for a given state-action pair $\sum_{s'} c(s,a,s') \geq m$, where m is a parameter which depends on the figures which define the precision of the algorithm [Brafman and Tennenholtz, 2002] (the higher value of m the more precise the model is, and the more accurate value function calculation according to this model), 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 (Equation 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 (Equation 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:

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

where $F(s,s')$ is computed according to Equation 1, that is, it is a difference of the value of potential Φ of states s' and s. Thus, 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 proceeding text 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. LEARNING POTENTIAL AND FREE SPACE ASSUMPTION

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

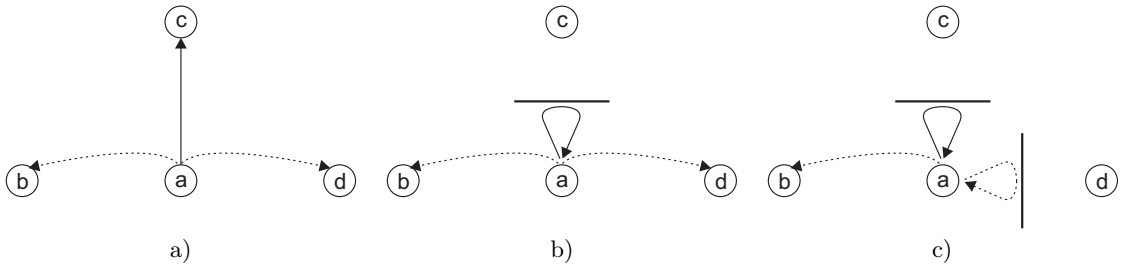

Figure 1: Probabilistic actions in a stochastic environment. Solid lines show expected effects of actions, and dashed lines show low-probability unexpected failures. Solid lines between circular states reflect no connectivity (walls) between those states.

moves the robot form 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 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 Figure 1.

An important property of the model defined according to the free state assumption is that it leads to an admissible value function when comparing 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 can not 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 section 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 section where the full algorithm is presented.

7. LEARNING POTENTIAL FOR R-MAX

In this section 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 can not be specified manually. The approach based on the free space assumption is presented in Algorithm 1. 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 5, and in Line 2 the initial FSA-model is created according to Section 6. The next two lines use these models to compute the potential Φ and the value function of the R-max algorithm. The VI method implements standard value iteration (Equation 3) and VI-RS is with reward shaping according to Equation 4. Since VI-RS requires the potential Φ to compute $F(s, s')$ in Equation 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 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

Algorithm 1 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: Φ ← VI(FSA-model)
 4: R-value-function ← VI-RS(R-model, Φ)
 5:
 6: s ← the start state
 7: repeat
 8:    a ← choose an action using R-value-function
 9:    (s', r) ← execute a in state s
10:    known ← 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:       Φ ← VI(FSA-model)
14:       R-value-function ← VI-RS(R-model, Φ)
15:    end if
16:    if for the first time ¬known in at least k episodes then
17:       FSA-model ← increase FSA transitions
18:       Φ ← VI(FSA-model)
19:       R-value-function ← VI-RS(R-model, Φ)
20:    end if
21:    if s' is terminal then
22:       s ← the start state
23:    else
24:       s ← s'
25:    end if
26: until terminal condition
```

the FSA-model in Line 11). The update of the R-model is according to the description in Section 5, 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 it was discussed in Section 5 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 how 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), 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 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 Φ when compared with the value function of the R-max algorithm. The potential 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 was not admissible indeed, the new value function will lead to additional exploration of the state space, since the new potential Φ is surly admissible in the sense of the value m of the R-max algorithm.

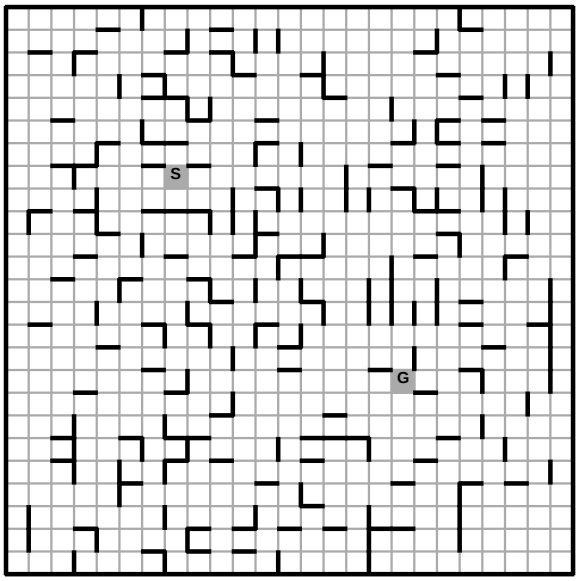

Figure 2: The stochastic navigation maze domain.

8. EXPERIMENTAL METHODOLOGY

Algorithm 1 was evaluated empirically on the navigation maze task that is shown in Figure 2. 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 Figure 1a). 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.

2. R-max-RS - the R-max algorithm with reward shaping according to Asmuth et al. [2008], the potential 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 1 in which the initial FSA-model is used during the entire learning process.

4. RS-FSA stands for the full version of the Algorithm 1, that is, with refinement of the FSA-model, recalculation of potential Φ, and final re-initialisation to ensure admissible potential with regard to the parameter m.

In all graphs in this section all evaluations were computed for 30 runs of all algorithms. The cumulative score of each algorithm is reported 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 and 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, 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 Figure 3.

9. RESULTS

Figure 3 shows the cumulative reward obtained by the four learning algorithms. The number of learning episodes was 1500. For better readability of results, Figure 3a shows the cumulative reward in 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 can not 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 good as R-max-RS. Curves for these two algorithms overlap since 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 Figure 3a 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 Figure 3b 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 Figure 3b than in Figure 3a. 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, Figure 3b is compared with the bottom-right part of Figure 3a. Curves are much further apart in Figure 3b which shows a bigger difference.

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

10. 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 of states is not available, or very difficult to represent directly in the form of a shaped reward. For this reason, this paper discusses a solution which allows applying the potential-based reward shaping when the potential function cannot be defined manually.

This paper introduces an algorithm to learn the potential in model-based RL. This algorithm applies the paradigm of estimating an abstract value function and uses this value function as a potential for the actual learning task. Specifically, our algorithm 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 can not 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 and Tennenholtz, 2002, Kearns and Singh, 2002].

References

J. Asmuth, M. L. Littman, and R. Zinkov. Potential-based shaping in model-based reinforcement learning. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2008.

D. P. Bertsekas. *Dynamic Programming and Optimal Control (2 Vol Set).* Athena Scientific, 3rd edition, 2007.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 1996.

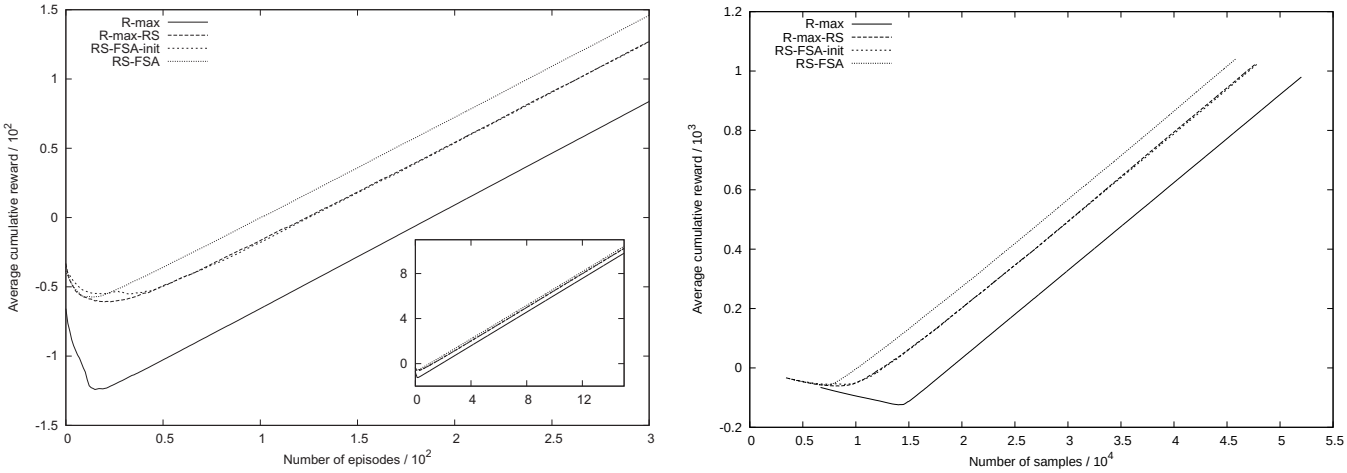

Figure 3: The cumulative reward plotted as a function of: a) the number of learning episodes, and b) the overall number of samples.

R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, pages 213–231, 2002.

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

M. Ghallab, D. Nau, and P. Traverso. *Automated Planning, Theory and Practice*. Elsevier, Morgan Kaufmann Publishers, 2004.

V. Gullapalli and A. G. Barto. Shaping as a method for accelerating reinforcement learning. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pages 554–559, 1992.

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

G. Konidaris and A. Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *The 23th International Conference on Machine Learning (ICML'06)*, 2006.

M. J. Mataric. Reward functions for accelerated learning. In *In Proceedings of the 11th International Conference on Machine Learning*, pages 181–189, 1994.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

A. Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *In Proceedings of Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pages 278–287, 1999.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994. ISBN 0471619779.

J. Randløv and P. Alstrom. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning*, pages 463–471, 1998.

D. C. Rayner, K. Davison, V. Bulitko, K. Anderson, and J. Lu. Real-time heuristic search with a priority queue. In *Proceedings of the 2007 International Joint Conference on Artificial Intelligence*, pages 2372–2377, 2007.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002. ISBN 0137903952.

A. L. Strehl, L. Li, and M. L. Littman. Pac reinforcement learning bounds for rtdp and rand-rtdp. In *Proceedings of AAAI Workshop on Learning for Search*, 2006.

R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Conference on Machine Learning*, pages 216–224, 1990.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, March 1998. ISBN 0262193981.

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

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