Preference-Based Policy Iteration: Leveraging Preference Learning for Reinforcement Learning

Weiwei Cheng[1], Johannes Fürnkranz[2],
Eyke Hüllermeier[1], and Sang-Hyeun Park[2]

[1] Department of Mathematics and Computer Science, Marburg University
{cheng,eyke}@mathematik.uni-marburg.de
[2] Department of Computer Science, TU Darmstadt
{juffi,park}@ke.tu-darmstadt.de

Abstract. This paper makes a first step toward the integration of two subfields of machine learning, namely preference learning and reinforcement learning (RL). An important motivation for a "preference-based" approach to reinforcement learning is a possible extension of the type of feedback an agent may learn from. In particular, while conventional RL methods are essentially confined to deal with numerical rewards, there are many applications in which this type of information is not naturally available, and in which only qualitative reward signals are provided instead. Therefore, building on novel methods for preference learning, our general goal is to equip the RL agent with qualitative policy models, such as ranking functions that allow for sorting its available actions from most to least promising, as well as algorithms for learning such models from qualitative feedback. Concretely, in this paper, we build on an existing method for approximate policy iteration based on roll-outs. While this approach is based on the use of classification methods for generalization and policy learning, we make use of a specific type of preference learning method called label ranking. Advantages of our preference-based policy iteration method are illustrated by means of two case studies.

1 Introduction

Standard methods for reinforcement learning (RL) assume feedback to be specified in the form of real-valued rewards. While such rewards are naturally generated in some applications, there are many cases in which precise numerical information is difficult to extract from the environment, or in which the specification of such information is largely arbitrary—as a striking though telling example, to which we shall return in Section 5, consider assigning a negative reward of -60 to the death of the patient in a medical treatment [17]. The quest for numerical information, even if accomplishable in principle, may also compromise efficiency in an unnecessary way. In a game playing context, for example, a short look-ahead from the current state may reveal that an action $\boldsymbol{a}$ is most likely superior to an action $\boldsymbol{a}'$; however, the precise numerical gains are only known at the end of the game. Moreover, external feedback, which is not produced by the

D. Gunopulos et al. (Eds.): ECML PKDD 2011, Part I, LNAI 6911, pp. 312–327, 2011.

environment itself but, say, by a human expert (e.g., "In this situation, action $\boldsymbol{a}$ would have been better than $\boldsymbol{a}'$"), is typically of a qualitative nature, too.

In order to make RL more amenable to qualitative feedback, we build upon formal concepts and methods from the rapidly growing field of preference learning [5]. Roughly speaking, we consider the RL task as a problem of learning the agent's preferences for actions in each possible state, that is, as a problem of *contextualized* preference learning (with the context given by the state). In contrast to the standard approach to RL, the agent's preferences are not necessarily expressed in terms of a utility function. Instead, more general types of preference models, as recently studied in preference learning, can be envisioned, such as total and partial order relations.

Interestingly, this approach is in a sense in-between the two extremes that have been studied in RL so far, namely learning numerical utility functions for all actions (as in Q-learning [15]) and, on the other hand, directly learning a policy which predicts a single best action in each state [11]. One may argue that the former approach is unnecessarily complex, since precise utility degrees are actually not necessary for taking optimal actions, whereas the latter approach is not fully effectual, since a prediction in the form of a single action does neither suggest alternative actions nor offer any means for a proper exploration. An order relation on the set of actions seems to provide a reasonable compromise, as it supports the exploration of acquired knowledge, i.e., the selection of (presumably) optimal actions, as well as the exploration of alternatives, i.e., the selection of suboptimal but still promising actions.

In this paper, we make a first step toward the integration of preference learning and reinforcement learning. We build upon a policy learning approach called approximate policy iteration, which will be detailed in Section 2, and propose a preference-based variant of this algorithm (Section 3). While the original approach is based on the use of classification methods for generalization and policy learning, we employ label ranking algorithms for incorporating preference information. Advantages of our preference-based policy iteration method are illustrated by means of two case studies presented in Sections 4 and 5.

2 Approximate Policy Iteration

Conventional reinforcement learning assumes a scenario in which an agent moves through a (finite) *state space* S by repeatedly selecting actions from a set of *actions* $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$. A Markovian *state transition* function $\delta : S \times A \to \mathbb{P}(S)$, where $\mathbb{P}(S)$ denotes the set of probability distributions over S, randomly takes the agent to a new state, depending on the current state and the chosen action. Occasionally, the agent receives feedback about its actions in the form of a reward signal $r : S \times A \to \mathbb{R}$, where $r(\boldsymbol{s}, \mathbf{a})$ is the reward the agent receives for performing action $\mathbf{a}$ in state $\boldsymbol{s}$. The goal of the agent is to choose its actions so as to maximize its expected total reward.

The most common task is to learn a *policy* $\pi : S \to A$ that prescribes the agent how to act optimally in each situation (state). More specifically, the goal

is often defined as maximizing the expected sum of rewards (given the initial state $\boldsymbol{s}$), with future rewards being discounted by a factor $\gamma \in [0, 1]$:

$$V^{\pi}(\boldsymbol{s}) \;=\; E\left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \pi(\boldsymbol{s}_t)) \,|\, \boldsymbol{s}_0 = \boldsymbol{s}\right] \tag{1}$$

where $(\boldsymbol{s}_0, \boldsymbol{s}_1, \boldsymbol{s}_2, \ldots)$ is a trajectory of π through the state space. With $V^*(\boldsymbol{s})$ the best possible value that can be achieved for (1), a policy is called optimal if it achieves the best value in each state $\boldsymbol{s}$. Thus, one possibility to learn an optimal policy is to learn an evaluation of states in the form of a value function [12], or to learn a so-called Q-function which returns the expected reward for a given state-action pair [15]:

$$Q^{\pi}(\boldsymbol{s}, \mathbf{a}) = r(\boldsymbol{s}, \mathbf{a}) + \gamma \cdot V^{\pi}(\delta(\boldsymbol{s}, \mathbf{a}))$$

Instead of determining optimal actions *indirectly* through learning the value function or the Q-function, one may try to learn a policy *directly* in the form of a mapping from states to actions. Approaches following this line include actor-critic algorithms, which learn both the value function (the critic) and an explicit policy (the actor) simultaneously [1,10,2], and policy gradient methods, which search for a good parameter setting in a space of parametrized policies [16,9,13].

A particularly interesting approach is approximate policy iteration with roll-outs [11,3]. The key idea of this approach is to use a generative model of the underlying process to perform simulations that in turn allow for approximating the value of an action in a given state (Algorithm 1). To this end, the action is performed, resulting in a state $\boldsymbol{s}_1 = \delta(\boldsymbol{s}, \mathbf{a})$. The value of this state is estimated by performing so-called *roll-outs*, i.e., by repeatedly selecting actions following a policy π for at most T steps, and finally accumulating the observed rewards. This is repeated K times and the average reward over these K roll-outs is returned as an approximate Q-value $\tilde{Q}^{\pi}(\boldsymbol{s}_0, \mathbf{a})$ for taking action $\mathbf{a}$ in state $\boldsymbol{s}_0$ (leading to $\boldsymbol{s}_1$) and following policy π thereafter.

These roll-outs are then used in a policy iteration loop (Algorithm 2), which iterates through each state, simulates all actions in this state, and determines the action $\mathbf{a}^*$ that promises the highest Q-value. If $\mathbf{a}^*$ is significantly better than all alternative actions in this state (indicated with the symbol $>_T$ in line 10), a training example $(\boldsymbol{s}, \mathbf{a}^*)$ is added to a training set $\mathcal{T}$. Eventually, $\mathcal{T}$ is used to directly learn a mapping from states to actions, which forms the new policy π'. This process is repeated several times, until some stopping criterion is met (e.g., if the policy does not improve from one iteration to the next).

We should note some minor differences between the version presented in Algorithm 2 and the original formulation [11]. Most notably, the training set here is formed as a multi-class training set, whereas in [11] it was formed as a binary training set, learning a binary policy predicate $\hat{\pi} : S \times A \to \{0, 1\}$. We chose the more general multi-class representation because, as we will see in the following, it lends itself to an immediate generalization to a ranking scenario.

Algorithm 1. ROLLOUT$(E, s_1, \gamma, \pi, K, T)$: Estimation of state-action values

Require: generative environment model E, sample state s_0, discount factor γ, policy π, number of trajectories/roll-outs K, max. length/horizon of each trajectory T

for $k = 1$ to K **do**
  $\boldsymbol{s} \leftarrow \boldsymbol{s}_1,\ \tilde{Q}_k \leftarrow 0,\ t \leftarrow 1$
  while $t < T$ and $\neg$TERMINALSTATE$(\boldsymbol{s})$ **do**
    $(\boldsymbol{s}', r) \leftarrow$ SIMULATE$(E, \boldsymbol{s}, \pi(\boldsymbol{s}))$
    $\tilde{Q}_k \leftarrow \tilde{Q}_k + \gamma^t r$
    $\boldsymbol{s} \leftarrow \boldsymbol{s}',\ t \leftarrow t + 1$
  end while
end for
$\tilde{Q} = \frac{1}{K} \sum_{k=1}^{K} \tilde{Q}_k$
return $\tilde{Q}$

3 Preference-Based Reinforcement Learning

The key idea of our approach is to replace the (quantitative) *evaluation of individual actions* by the (qualitative) *comparison between pairs of actions*. Comparisons of that kind are in principle enough to make optimal decisions. Besides, they are often more natural and less difficult to acquire, especially in applications where the environment does not provide numerical rewards in a natural way. As will be seen later on, comparing pairs instead of evaluating individual actions does also have a number of advantages from a learning point of view.

The basic piece of information we consider is a pairwise preference of the form $\mathbf{a}_i \succ_{\boldsymbol{s}} \mathbf{a}_j$ or, more specifically, $\mathbf{a}_i \succ^{\pi}_{\boldsymbol{s}} \mathbf{a}_j$, suggesting that in state $\boldsymbol{s}$, taking action $\mathbf{a}_i$ (and following policy π afterward) is better than taking action $\mathbf{a}_j$. A preference of this kind can be interpreted in different ways. For example, assuming the existence of an underlying (though not necessarily known) reward function, it may simply mean that $Q^{\pi}(\boldsymbol{s}, \mathbf{a}_i) > Q^{\pi}(\boldsymbol{s}, \mathbf{a}_j)$.

Evaluating a trajectory $t = (\boldsymbol{s}_0, \boldsymbol{s}_1, \boldsymbol{s}_2, \ldots)$ in terms of its (expected) total reward reduces the comparison of trajectories to the comparison of real numbers; thus, comparability is enforced and a total order on trajectories is induced. More generally, and arguably more in line with the idea of qualitative feedback, one may assume a partial order relation $\sqsupset$ on trajectories, which means that trajectories t and t' can also be incomparable. A contextual preference can then be defined as follows:

$$\mathbf{a}_i \succ^{\pi}_{\boldsymbol{s}} \mathbf{a}_j \quad \Leftrightarrow \quad \mathbf{P}(t(\mathbf{a}_i) \sqsupset t(\mathbf{a}_j)) > \mathbf{P}(t(\mathbf{a}_j) \sqsupset t(\mathbf{a}_i)) \ ,$$

where $t(\mathbf{a}_i)$ denotes the (random) trajectory produced by taking action $\mathbf{a}_i$ in state $\boldsymbol{s}_0$ and following π thereafter, and $\mathbf{P}(t \sqsupset t')$ is the probability that trajectory t is preferred to t'. An in-depth discussion of this topic is beyond the scope of this paper, however, an example in which $\sqsupset$ is a Pareto-dominance relation will be presented in Section 5.

Algorithm 2. Multi-class variant of Approx. Policy Iteration with Roll-Outs[11]

Require: generative environment model E, sample states S, discount factor γ, initial (random) policy π_0, number of trajectories/roll-outs K, max. length/horizon of each trajectory T, max number of policy iterations p

$\pi' \leftarrow \pi_0$
repeat
  $\pi \leftarrow \pi'$, $\mathcal{T} \leftarrow \emptyset$
  for each $\boldsymbol{s} \in S$ **do**
    for each $\mathbf{a} \in A$ **do**
      $(\boldsymbol{s}', r) \leftarrow \textsc{Simulate}(E, \boldsymbol{s}, \mathbf{a})$ # do (possibly off-policy) action $\mathbf{a}$
      $\tilde{Q}^{\pi}(\boldsymbol{s}, \mathbf{a}) \leftarrow \textsc{Rollout}(E, \boldsymbol{s}', \gamma, \pi, K, T) + r$ # estimate state-action value
    end for
    $\mathbf{a}^* \leftarrow \arg\max_{\mathbf{a} \in A} \tilde{Q}^{\pi}(\boldsymbol{s}, \mathbf{a})$
    if $\tilde{Q}^{\pi}(\boldsymbol{s}, \mathbf{a}^*) >_T \tilde{Q}^{\pi}(\boldsymbol{s}, \mathbf{a})$ for all $\mathbf{a} \in A, \mathbf{a} \neq \mathbf{a}^*$ **then**
      $\mathcal{T} \leftarrow \mathcal{T} \cup \{(\boldsymbol{s}, \mathbf{a}^*)\}$
    end if
  end for
  $\pi' \leftarrow \textsc{learn}(\mathcal{T})$
until $\textsc{StoppingCriterion}(E, \pi, \pi', p)$

In order to realize our idea of preference-based approximate policy iteration, to be detailed in Section 3.2 below, we need a learning method that induces a suitable preference model on the basis of training information in the form of pairwise preferences of the above kind. Ideally, given a state, the model allows one to rank the possible actions from (presumably) most to least desirable. A setting nicely matching these requirements is the setting of *label ranking*.

3.1 Label Ranking

Like in the conventional setting of supervised learning (classification), assume to be given an instance space X and a finite set of labels $Y = \{y_1, y_2, \ldots, y_k\}$. In label ranking, the goal is to learn a "label ranker" in the form of an $X \to \mathfrak{S}_Y$ mapping, where the output space $\mathfrak{S}_Y$ is given by the set of all total orders (permutations) of the set of labels Y. Thus, label ranking can be seen as a generalization of conventional classification, where a complete ranking

$$y_{\tau_{\boldsymbol{x}}^{-1}(1)} \succ_{\boldsymbol{x}} y_{\tau_{\boldsymbol{x}}^{-1}(2)} \succ_{\boldsymbol{x}} \cdots \succ_{\boldsymbol{x}} y_{\tau_{\boldsymbol{x}}^{-1}(k)}$$

is associated with an instance $\boldsymbol{x}$ instead of only a single class label. Here, $\tau_{\boldsymbol{x}}$ is a permutation of $\{1, 2, \ldots, k\}$ such that $\tau_{\boldsymbol{x}}(i)$ is the position of label y_i in the ranking associated with $\boldsymbol{x}$.

The training data $\mathcal{T}$ used to induce a label ranker typically consists of a set of pairwise preferences of the form $y_i \succ_{\boldsymbol{x}} y_j$, suggesting that, for instance $\boldsymbol{x}$, y_i is preferred to y_j. In other words, a single "observation" consists of an instance $\boldsymbol{x}$ together with an ordered pair of labels (y_i, y_j).

Several methods for label ranking have already been proposed in the literature; we refer to [14] for a comprehensive survey. The idea of *learning by pairwise comparison* (LPC) [8] is to train a separate model $\mathcal{M}_{i,j}$ for each pair of labels $(y_i, y_j) \in Y \times Y$, $1 \leq i < j \leq k$; thus, a total number of $k(k-1)/2$ models is needed. At classification time, a query $\boldsymbol{x}$ is submitted to all models, and each prediction $\mathcal{M}_{i,j}(\boldsymbol{x})$ is interpreted as a vote for a label. More specifically, assuming scoring classifiers that produce normalized scores $f_{i,j} = \mathcal{M}_{i,j}(\boldsymbol{x}) \in [0,1]$, the *weighted voting* technique interprets $f_{i,j}$ and $f_{j,i} = 1 - f_{i,j}$ as weighted votes for classes y_i and y_j, respectively, and predicts the class y^* with the highest sum of weighted votes, i.e., $y^* = \arg\max_i \sum_{j \neq i} f_{i,j}$. We refer to [8] for a more detailed description of LPC in general and a theoretical justification of the weighted voting procedure in particular.

3.2 Preference-Based Approximate Policy Iteration

Recall the scenario described at the end of Section 2, where the agent has access to a generative model E, which takes a state $\boldsymbol{s}$ and an action $\mathbf{a}$ as input and returns a successor state $\boldsymbol{s}'$ and the reward $r(\boldsymbol{s}, \mathbf{a})$. As in [11], this scenario is used for generating training examples via roll-outs, i.e., by using the generative model and the current policy π for generating a training set $\mathcal{T}$, which is used for training a multi-class classifier that can be used as a policy.

Following our idea of preference-based RL, we train a *label ranker* instead of a *classifier*: Using the notation from Section 3.1 above, the instance space X is given by the state space S, and the set of labels Y corresponds to the set of actions A. Thus, the goal is to learn a mapping $S \to \mathfrak{S}_A$, which maps a given state to a total order (permutation) of the available actions. In other words, the task of the learner is to learn a function that is able to rank all available actions in a state. The training information is provided in the form of binary action preferences of the form $(\boldsymbol{s}, \mathbf{a}_k \succ \mathbf{a}_j)$, indicating that in state $\boldsymbol{s}$, action $\mathbf{a}_k$ is preferred to action $\mathbf{a}_j$.

From a training point of view, a key advantage of this approach is that pairwise preferences are much easier to elicit than examples for unique optimal actions. Our experiments in Sections 4 and 5 utilize this in different ways.

Section 4 demonstrates that a comparison of only two actions is less difficult than "proving" the optimality of one among a possibly large set of actions, and that, as a result, our preference-based approach better exploits the gathered training information. Indeed, the procedure proposed in [11] for forming training examples is very wasteful with this information. An example $(\boldsymbol{s}, \mathbf{a}^*)$ is only generated if $\mathbf{a}^*$ is "provably" the best action among all candidates, namely if it is (significantly) better than all other actions in the given state. Otherwise, if this superiority is not confirmed by a statistical hypothesis test, all information about this state is ignored. In particular, no training examples would be generated in states where multiple actions are optimal, even if they are clearly better

than all remaining actions.[1] For the preference-based approach, on the other hand, it suffices if only two possible actions yield a clear preference in order to obtain (partial) training information about that state. Note that a corresponding comparison may provide useful information even if both actions are suboptimal.

In Section 5, an example will be shown in which actions are not necessarily comparable, since the agent seeks to optimize multiple criteria at the same time (and is not willing to aggregate them into a one-dimensional target). In general, this means that, while at least some of the actions will still be comparable in a pairwise manner, a unique optimal action does not exist.

Regarding the type of prediction produced, it was already mentioned earlier that a ranking-based reinforcement learner can be seen as a reasonable compromise between the estimation of a numerical utility function (like in Q-learning) and a classification-based approach which provides only information about the optimal action in each state: The agent has enough information to determine the optimal action, but can also rely on the ranking in order to look for alternatives, for example to steer the exploration towards actions that are ranked higher. We will briefly return to this topic at the end of the next section. Before that, we will discuss the experimental setting in which we evaluate the utility of the additional ranking-based information.

4 Case Study I: Exploiting Action Preferences

In this section, we compare three variants of approximate policy iteration following Algorithm 2. They only differ in the way in which they use the information gathered from the performed roll-outs.

Approximate Policy Iteration (API) generates one training example $(\boldsymbol{s}, \mathbf{a}^*)$ if $\mathbf{a}^*$ is the best available action in $\boldsymbol{s}$, i.e., if $\tilde{Q}^\pi(s, \mathbf{a}^*) >_T \tilde{Q}^\pi(s, \mathbf{a})$ for all $\mathbf{a} \neq \mathbf{a}^*$. If there is no action that is better than all alternatives, no training example is generated for this state.

Pairwise Approximate Policy Iteration (PAPI) works in the same way as API, but the underlying base learning algorithm is replaced with a label ranker. This means that each training example $(\boldsymbol{s}, \mathbf{a}^*)$ of API is transformed into $a - 1$ training examples of the form $(\boldsymbol{s}, \mathbf{a}^* \succ \mathbf{a})$ for all $\mathbf{a} \neq \mathbf{a}^*$.

Preference-Based Policy Iteration (PBPI) is trained on all available pairwise preferences, not only those involving the best action. Thus, whenever $\tilde{Q}^\pi(s, \mathbf{a}_k) >_T \tilde{Q}^\pi(s, \mathbf{a}_l)$ holds for a pair of actions $(\mathbf{a}_k, \mathbf{a}_l)$, PBPI generates a corresponding training example $(\boldsymbol{s}, \mathbf{a}_k \succ \mathbf{a}_l)$. Note that, contrary to PAPI, $\mathbf{a}_k$ does not need to be the best action. In particular, it is not necessary that there is a clear best action in order to generate training examples. Thus, from the same roll-outs, PBPI will typically generate more training information than PAPI or API.

[1] In the original formulation as a binary problem, it is still possible to produce negative examples, which indicate that the given action is certainly not the best action (because it was significantly worse than the best action).

4.1 Application Domains

Following [3], we evaluated these variants on two well-known problems, inverted pendulum and mountain car. We will briefly recapitulate these tasks, which were used in their default setting, unless stated otherwise.

In the *inverted pendulum* problem, the task is to push or pull a cart so that it balances an upright pendulum. The available actions are to apply a force of fixed strength of 50 Newton to the left (-1), to the right (+1) or to apply no force at all (0). The mass of the pole is 2 kg and of the cart 9 kg. The pole has a length of 0.5 m and each time step is set to 0.1 seconds. Following [3], we describe the state of the pendulum using only the angle and angular velocity of the pole, ignoring the position and the velocity of cart. For each time step, where the pendulum is above the horizontal line, a reward of 1 was given, else 0. A policy was considered *sufficient*, if it is able to balance the pendulum longer than 1000 steps (100 sec). The random samples in this setting were generated by simulating a uniform random number (max 100) of uniform random actions from the initial state (pole straight up, no velocity for cart and pole). If the pendulum fell within this sequence, the procedure was repeated.

In the *mountain car* problem, the task is to drive a car out of a steep valley. To do so, it has to repeatedly go up on each side of the hill, gaining momentum by going down and up to the other side, so that eventually it can get out. Again, the available actions are (-1) for left or backward and $(+1)$ for right or forward and (0) for a fixed level of throttle. The states or feature vectors consist of the horizontal position and the current velocity of the car. Here, the agent received a reward of -1 in each step until the goal was reached. A policy which needed less than 75 steps to reach the goal was considered as sufficient.

4.2 Experimental Setup

In addition to these conventional formulations using three actions in each state, we also used versions of these problems with 5, 9, and 17 actions, because in these cases it becomes less and less likely that a unique best actions can be found, and the benefit from being able to utilize information from states where no clear winner emerges increases. The range of the original action set $\{-1, 0, 1\}$ was partitioned equidistantly into the given number of actions, for e.g., using 5 actions, the set of action signals is $\{-1, -0.5, 0, 0.5, 1\}$. Also, a uniform noise term in $[-0.2, 0.2]$ was added to the action signal, such that all state transitions are non-deterministic. For training the label ranker we use LPC (cf. Section 3.1) with simple multi-layer perceptrons (as implemented in the Weka machine learning library [7] with its default parameters) as base classifiers. The discount factor for both settings was set to 1 and the maximal length of the trajectory for the inverted pendulum task was set to 1500 steps and 1000 for the mountain car task. The policy iteration algorithms terminated if the learned policy was sufficient or if the policy performance decreased or if the number of policy iterations reached 10. For the evaluation of the policy performance, 100 simulations beginning from the corresponding initial states were utilized.

For each task and method, we tried five numbers of state samples $s \in \{10, 20, 50, 100, 200\}$, five maximum numbers of roll-outs $r \in \{10, 20, 50, 100, 200\}$, and three levels of significance $c \in \{0.025, 0.05, 0.1\}$. Each of the $5 \times 5 \times 3 = 75$ parameter combinations was evaluated ten times, such that the total number of experiments per learning task was 750. We tested both domains, mountain car and inverted pendulum, with $a \in \{3, 5, 9, 17\}$ different actions each.

Our prime evaluation measure is the *success rate* (SR), i.e., the percentage of learned sufficient policies. Following [3], we plot a cumulative distribution of the success rates of all different parameter settings over a measure of learning complexity, where each point (x, y) indicates the minimum complexity x needed to reach a success rate of y. However, while [3] simply use the number of roll-outs (i.e., the number of sampled states) as a measure of learning complexity, we use the number of performed actions over all roll-outs, which is a more fine-grained complexity measure. The two would coincide if all roll-outs are performed a constant number of times. However, this is typically not the case, as some roll-outs may stop earlier than others. Thus, we generated graphs by sorting all successful runs over all parameter settings (i.e., runs which yielded a sufficient policy) in increasing order regarding the number of applied actions and by plotting these runs along the x-axis with a y-value corresponding to its cumulative success rate. This visualization can be interpreted roughly as the development of the success rate in dependence of the applied learning complexity.

4.3 Complete State Evaluations

Figure 1 shows the results for the inverted pendulum and the mountain car tasks. One can clearly see that for an increasing number of actions, PBPI reaches a significantly higher success rate than the two alternative approaches, and it typically also has a much faster learning curve, i.e., it needs to take fewer actions to reach a given success rate. Another interesting point is that the maximum success level decreases with an increasing number of actions for API and PAPI, but it remains essentially constant for PBPI. Overall, these results clearly demonstrate that the additional information about comparisons of lower-ranked action pairs, which is ignored in API and PAPI, can be put to effective use when approximate policy iteration is extended to use a label ranker instead of a mere classifier.

4.4 Partial State Evaluations

So far, based on the API strategy, we always evaluated all possible actions at each state, and generated preferences from their pairwise comparisons. A possible advantage of the preference-based approach is that it does not need to evaluate all options at a given state. In fact, one could imagine to select only two actions for a state and compare them via roll-outs. While such a partial state evaluation will, in general, not be sufficient for generating a training example for API, it suffices to generate a training preference for PBPI. Thus, such a *partial PBPI* strategy also allows for considering a far greater number of states, using the same number of roll-outs, at the expense that not all actions of each state will

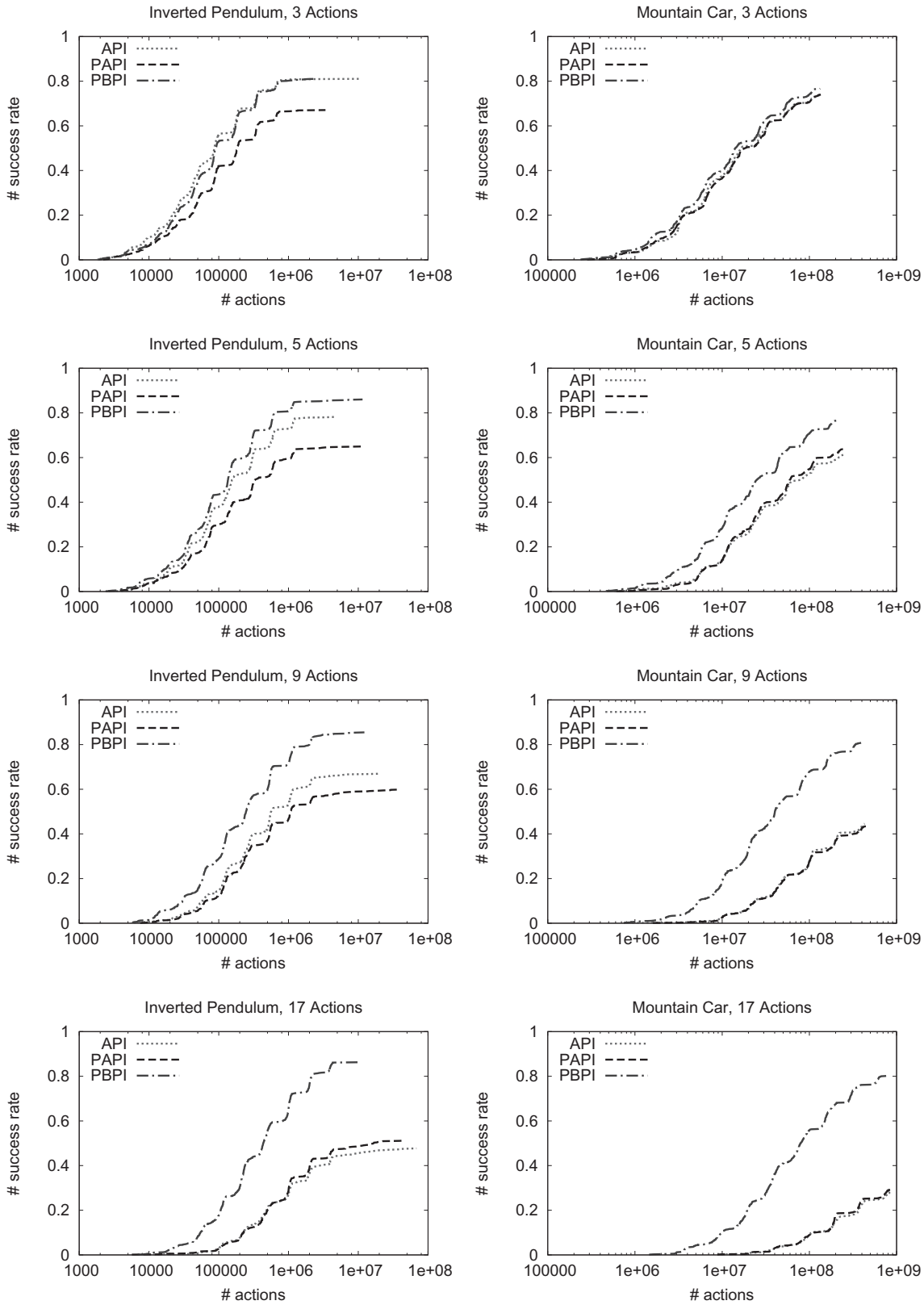

Fig. 1. Comparison of API, PAPI and PBPI for the inverted pendulum task (left) and the mountain car task (right). The number of actions is increasing from top to bottom.

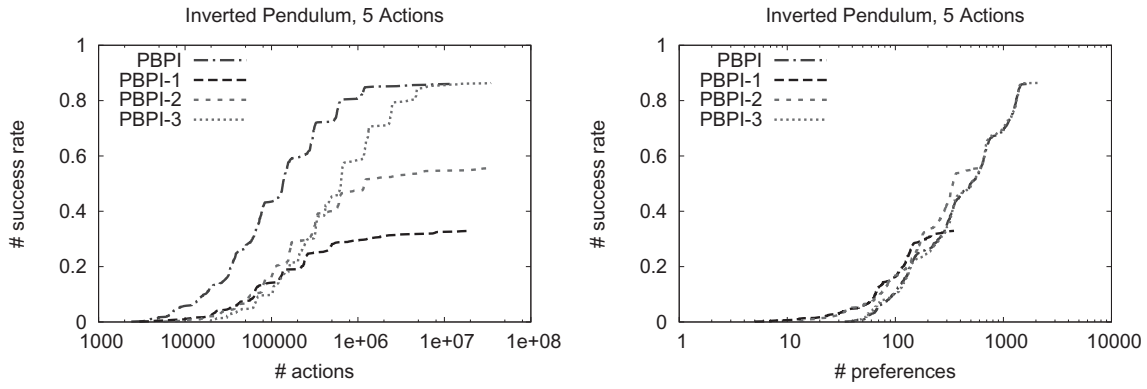

Fig. 2. Comparison of complete state evaluation (PBPI) with partial state evaluation in three variants (PBPI-1, PBPI-2, PBPI-3)

be explored. Such an approach may thus be considered to be orthogonal to recent approaches for roll-out allocation strategies [3,6].

In order to investigate this effect, we also experimented with three partial variants of PBPI, which only differ in the number of states that they are allowed to visit. The first (PBPI-1) allows the partial PBPI variant to visit only the same total number of states as PBPI. The second (PBPI-2) adjusts the number of visited sample states by multiplying it with $\frac{k}{2}$, to account for the fact that the partial variant performs only 2 action roll-outs in each state, as opposed to k action roll-outs for PBPI. Thus, the total number of action roll-outs in PBPI and PBPI-2 is constant. Finally, for the third variant (PBPI-3), we assume that the number of preferences that are generated from each state is constant. While PBPI generates up to $\frac{k(k-1)}{2}$ preferences from each visited state, partial PBPI generates only one preference per state, and is thus allowed to visit $\frac{k(k-1)}{2}$ as many states.

Figure 2 shows the results for the inverted pendulum with five different actions (the results for the other problems are quite similar). The left graph shows the success rate over the total number of taken actions, whereas the right graph shows the success rate over the total number of training preferences. From the right graph, no clear differences can be seen. In particular, the curves for PBPI-3 and PBPI almost coincide. This is not surprising, because both generate the same number of preference samples, albeit for different random states. However, the left graph clearly shows that the exploration policies that do not generate all action roll-outs for each state are more wasteful with respect to the total number of actions that have to be taken in the roll-outs. Again, this is not surprising, because evaluating all five actions in a state may generate up to 10 preferences for a single state, or, in the case of PBPI-2, only a total of 5 preferences if 2 actions are compared in each of 5 states.

Nevertheless, the results demonstrate that partial state evaluation is feasible. This may form the basis of novel algorithms for exploring the state space. For example, it was suggested that a policy-based generation of states may be preferable to a random selection [4]. While this may clearly lead to faster convergence

in some domains, it may also fail to find optimal solutions in other cases [11]. Selecting a pair of actions and following the better one may be a simple but effective way of trading off exploration and exploitation for state sampling. We are currently working on a more elaborate investigation of this issue.

5 Case Study II: Learning from Qualitative Feedback

In a second experiment, we applied preference-based reinforcement learning to a simulation of optimal therapy design in cancer treatment, using a model that was recently proposed in [17]. In this domain, it is arguably more natural to define preferences that induce a partial order between states than to define an artificial numerical reward function that induces a total order between states.

5.1 Cancer Clinical Trials Domain

The model proposed in [17] captures a number of essential factors in cancer treatment: (i) the tumor growth during the treatment; (ii) the patient's (negative) wellness, measured in terms of the level of toxicity in response to chemotherapy; (iii) the effect of the drug in terms of its capability to reduce the tumor size while increasing toxicity; (iv) the interaction between the tumor growth and patient's wellness. The two state variables, the tumor size S and the toxicity X, are modeled using a system of difference equations: $S_{t+1} = S_t + \Delta S_t$ and $X_{t+1} = X_t + \Delta X_t$, where the time variable t denotes the number of months after the start of the treatment and assumes values $t = 0, 1, \ldots, 6$. The terms ΔS and ΔX indicate the increments of the state variables that depend on the action, namely the dosage level D:

$$\begin{aligned} \Delta S_t &= \left[a_1 \cdot \max(X_t, X_0) - b_1 \cdot (D_t - d_1)\right] \times \mathbf{1}(S_t > 0) \\ \Delta X_t &= a_2 \cdot \max(S_t, S_0) + b_2 \cdot (D_t - d_2) \end{aligned} \tag{2}$$

These changing rates produce a piecewise linear model over time. We fix the parameter values following the recommendation of [17]: $a_1 = 0.15, a_2 = 0.1, b_1 = b_2 = 1.2$ and $d_1 = d_2 = 0.5$. By using the indicator term $\mathbf{1}(S_t > 0)$, the model assumes that once the patient has been cured, namely the tumor size is reduced to 0, there is no recurrence. Note that this system does not reflect a specific cancer but rather models the generic development of the chemotherapy process.

The possible death of a patient in the course of a treatment is modeled by means of a hazard rate model. For each time interval $(t-1, t]$, this rate is defined as a function of tumor size and toxicity: $\lambda(t) = \exp(c_0 + c_1 S_t + c_2 X_t)$, where c_0, c_1, c_2 are cancer-dependent constants. Again following [17], we let $c_0 = -4, c_1 = c_2 = 1$. By setting $c_1 = c_2$, the tumor size and the toxicity have an equally important influence on patient's survival. The probability of the patient's death during the time interval $(t-1, t]$ is calculated as

$$P_{\text{death}} = 1 - \exp\left[-\int_{t-1}^{t} \lambda(x)\, dx\right].$$

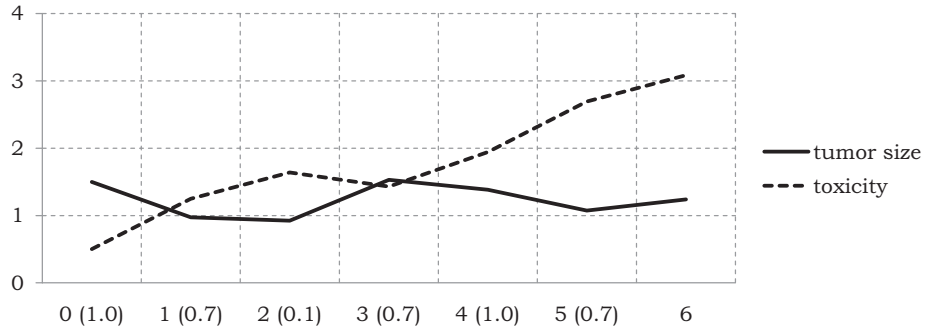

Fig. 3. Illustration of the simulation model showing the patient's status during the treatment. The initial tumor size is 1.5 and the initial toxicity is 0.5. On the x-axis is the month with the corresponding dosage level the patient receives. The dosage levels are selected randomly.

5.2 A Preference-Based Approach

The problem is to learn an optimal treatment policy π mapping states (S, X) to actions in the form of a dosage level D, where the dosage level is a number between 0 and 1 (minimum and maximum dosage, respectively). In [17], the authors tackle this problem by means of RL, and indeed obtained interesting results. However, using standard RL techniques, there is a need to define a numerical reward function depending on the tumor size, wellness, and possibly the death of a patient. More specifically, four threshold values and eight utility scores are needed, and the authors themselves notice that these quantities strongly influence the results.

We consider this as a key disadvantage of the approach, since in a medical context, a numerical function of that kind is extremely hard to specify and will always be subject to debate. Just to give a striking example, the authors defined a negative reward of -60 for the death of a patient, which, of course, is a rather arbitrary number. As an interesting alternative, we tackle the problem using a more qualitative approach.

To this end, we treat the criteria (tumor size, wellness, death) independently of each other, without the need to aggregate them in a mathematical way; in fact, the question of how to "compensate" or trade off one criterion against another one is always difficult, especially in fields like medicine. Instead, we compare two policies π and π' as follows: $\pi \succeq \pi'$ if the patient survives under π but not under π', and both policies are incomparable if the patient does neither survive under π nor under π'. Otherwise, if the patient survives under both policies, let C_X denote the maximal toxicity during the 6 months of treatment under π and, correspondingly, C'_X under treatment π'. Likewise, let C_S and C'_S denote the respective size of the tumor at the end of the therapy. Then, we define preference via Pareto dominance as

$$\pi \succeq \pi' \quad \Leftrightarrow \quad (C_X \leq C'_X) \text{ and } (C_S \leq C'_S) \tag{3}$$

It is important to remark that $\succeq$ thus defined, as well as the induced strict order $\succ$, are only *partial* order relations. In other words, it is thoroughly possible that two policies are incomparable. For our preference learning framework, this means that less pairwise comparisons may be generated as training examples. However, in contrast to standard RL methods as well as the classification approach of [11], this is not a conceptual problem. In fact, since these approaches are based on a numerical reward function and, therefore, implicitly assume a total order among policies (and actions in a state), they are actually not applicable in the case of a partial order.

5.3 Experimental Setup and Results

For training, we generate 1000 patients at random. That is, we simulate 1000 patients experiencing the treatment based on model (2). The initial state of each patient, S_0 and X_0, are generated independently and uniformly from $(0, 2)$. Then, for the following 6 months, the patient receives a monthly chemotherapy with a dosage level taken from one of four different values (actions) 0.1 (low), 0.4 (medium), 0.7 (high) and 1.0 (extreme), where 1.0 corresponds to the maximum acceptable dose.[2] As an illustration, Fig. 3 shows the treatment process of one patient according to model (2) under a randomly selected chemotherapy policy. The patient's status is clearly sensitive to the amount of received drug. When dosage level is too low, the tumor size grows towards a dangerous level, while with a very high dosage level, the toxicity level will strongly affect the patient's wellness. The preferences are generated via Pareto dominance relation 3 using roll-outs. We use LPC and choose a linear classifier, logistic regression, as the base learner (again using the Weka implementation). The policy iteration stops when (i) the difference between two consequential learned policies is smaller than a pre-defined threshold, or (ii) the number of policy iterations reaches 10.

For testing, we further generate 200 virtual patients. In Fig. 4, the average values of the two criteria (C_X, C_S) are shown as points for the constant policies low, medium, high, extreme (i.e., the policies prescribing a constant dosage regardless of the state). As can be seen, all four policies are Pareto-optimal, which is hardly surprising in light of the fact that toxicity and tumor size are conflicting criteria: A reduction of the former tends to increase the latter, and vice versa. The figure also shows the convex hull of the Pareto-optimal policies.

Finally, we add the results for two other policies, namely the policy learned by our preference-based approach and a random policy, which, in each state, picks a dose level at random. Although these two policies are again both Pareto-optimal, it is interesting to note that our policy is outside the convex hull of the constant policies, whereas the random policy falls inside. Recalling the interpretation of the convex hull in terms of randomized strategies, this means that the random policy can be outperformed by a randomization of the constant policies, whereas our policy can not.

[2] We exclude the value 0, as it is a common practice to let the patient keep receiving certain level of chemotherapy agent during the treatment in order to prevent the tumor relapsing.

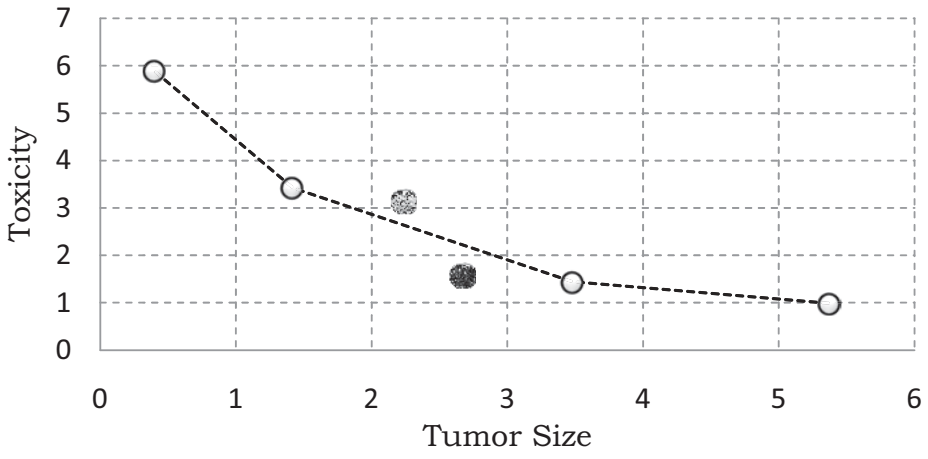

Fig. 4. Illustration of patients status under different treatment policies. On the x-axis is the tumor size after 6 months. On the y-axis is the highest toxicity during the 6 months. From top to bottom: Extreme dose level (1.0), high dose level (0.7), random dose level, learned dose level, medium dose level (0.4), low dose level (0.1). The values are averaged from 200 patients.

6 Conclusions

The goal of this work is to make first steps towards lifting conventional reinforcement learning into a qualitative setting, where reward is not available on an absolute, numerical scale, but where comparative reward functions can be used to decide which of two actions is preferable in a given state. To cope with this type of training information, we proposed a preference-based extension of approximate policy iteration. Whereas the original approach essentially reduces reinforcement learning to classification, we tackle the problem by means of a preference learning method called label ranking. In this setting, a policy is represented by a ranking function that maps states to total orders of all available actions.

To demonstrate the feasibility of this approach, we performed two case studies. In the first study, we showed that additional training information about lower-ranked actions can be successfully used for improving the learned policies. The second case study demonstrated one of the key advantages of a qualitative policy iteration approach, namely that a comparison of pairs of actions is often more feasible than the quantitative evaluation of single actions.

The work reported in this paper provides a point of departure for extensions along several lines. For example, while the setting we assumed is not uncommon in the literature, the existence of a generative model is a strong assumption. In future work, we will therefore focus on generalizing our approach toward an on-line learning setting with on-policy updates.

Acknowledgments. We would like to thank the Frankfurt Center for Scientific Computing for providing computational resources. This research was supported by the *German Science Foundation (DFG).*

References

1. Barto, A.G., Sutton, R.S., Anderson, C.: Neuron-like elements that can solve difficult learning control problems. IEEE Transaction on Systems, Man and Cybernetics 13, 835–846 (1983)
2. Bhatnagar, S., Sutton, R.S., Ghavamzadeh, M., Lee, M.: Natural actor-critic algorithms. Automatica 45(11), 2471–2482 (2009)
3. Dimitrakakis, C., Lagoudakis, M.G.: Rollout sampling approximate policy iteration. Machine Learning 72(3), 157–171 (2008)
4. Fern, A., Yoon, S.W., Givan, R.: Approximate policy iteration with a policy language bias: Solving relational markov decision processes. Journal of Artificial Intelligence Research 25, 75–118 (2006)
5. Fürnkranz, J., Hüllermeier, E. (eds.): Preference Learning. Springer, Heidelberg (2010)
6. Gabillon, V., Lazaric, A., Ghavamzadeh, M.: Rollout allocation strategies for classification-based policy iteration. In: Auer, P., Kaski, S., Szepesvàri, C. (eds.) Proceedings of the ICML 2010 Workshop on Reinforcement Learning and Search in Very Large Spaces (2010)
7. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The weka data mining software: An update. SIGKDD Explorations 11(1), 10–18 (2009)
8. Hüllermeier, E., Fürnkranz, J., Cheng, W., Brinker, K.: Label ranking by learning pairwise preferences. Artificial Intelligence 172, 1897–1916 (2008)
9. Kersting, K., Driessens, K.: Non-parametric policy gradients: a unified treatment of propositional and relational domains. In: Cohen, W.W., McCallum, A., Roweis, S.T. (eds.) Proceedings of the 25th International Conference on Machine Learning (ICML 2008), pp. 456–463. ACM, Helsinki (2008)
10. Konda, V.R., Tsitsiklis, J.N.: On actor-critic algorithms. SIAM Journal of Control and Optimization 42(4), 1143–1166 (2003)
11. Lagoudakis, M.G., Parr, R.: Reinforcement learning as classification: Leveraging modern classifiers. In: Fawcett, T.E., Mishra, N. (eds.) Proceedings of the 20th International Conference on Machine Learning (ICML 2003), pp. 424–431. AAAI Press, Washington, DC (2003)
12. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning 3, 9–44 (1988)
13. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Solla, S.A., Leen, T.K., Müller, K.-R. (eds.) Advances in Neural Information Processing Systems 12 (NIPS-1999), pp. 1057–1063. MIT Press, Denver (1999)
14. Vembu, S., Gärtner, T.: Label ranking algorithms: A survey. In: Fürnkranz and Hüllermeier [5], pp. 45–64.
15. Watkins, C.J., Dayan, P.: Q-learning. Machine Learning 8, 279–292 (1992)
16. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8, 229–256 (1992)
17. Zhao, Y., Kosorok, M., Zeng, D.: Reinforcement learning design for cancer clinical trials. Statistics in Medicine 28, 3295–3315 (2009)