Machine Learning manuscript No.
(will be inserted by the editor)

Towards Preference-Based Reinforcement Learning

Johannes Fürnkranz, Eyke Hüllermeier,
Weiwei Cheng, Sang-Hyeun Park

the date of receipt and acceptance should be inserted later

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 the observation that in many real-world domains, numerical feedback signals are not readily available, or are defined arbitrarily in order to satisfy the needs of conventional RL algorithms. Instead, we propose an alternative framework for reinforcement learning, in which qualitative reward signals can be directly used by the learner. The framework may be viewed as a generalization of the conventional RL framework in which only a partial order between policies is required instead of the total order induced by their respective expected long-term reward. 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. As a proof of concept, we realize a first simple instantiation of this framework that defines preferences based on utilities observed for trajectories. To that end, 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 preference-based policy iteration are illustrated by means of two case studies.

Keywords reinforcement learning, preference learning

J. Fürnkranz · S.-H. Park
Department of Computer Science, TU Darmstadt
E-mail: {juffi, park}@ke.tu-darmstadt.de

E. Hüllermeier · W. Cheng
Department of Mathematics and Computer Science, Marburg University
E-mail: {eyke, cheng}@mathematik.uni-marburg.de

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 domains in which precise numerical information is difficult to extract from the environment, or in which the specification of such information is largely arbitrary. 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 environment itself but, say, by a human expert (e.g., "In this situation, action $\boldsymbol{a}$ would have been better than $\mathbf{a}'$"), 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 (Fürnkranz and Hüllermeier 2010). 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 (e.g., Watkins and Dayan 1992) and, on the other hand, directly learning a policy which predicts a single best action in each state (e.g., Lagoudakis and Parr 2003). 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), but at the same time also provides information about which alternatives are more promising than others.

The main contribution of this paper is a formal framework for preference-based reinforcement learning. Its key idea is the observation that, while a numerical reward signal induces a total order on the set of trajectories, a qualitative reward signal only induces a partial order on this set. This makes the problem considerably more difficult, because crucial steps such as the comparison of policies, which can be realized in a numerical setting by estimating their expected reward, are becoming more complex. In this particular case, we propose a solution based on stochastic dominance between probability dis-

tributions on the space of trajectories. Once having defined preferences of trajectories, we can also deduce preferences between states and actions. The proposed framework is quite related to the approach of Akrour et al. (2011). As we will discuss in more detail in Section 8, the main differences are that their approach works with preferences over policies, and uses this information to directly learn to rank policies, whereas we learn to rank actions.

We will start the paper with a discussion on the importance of qualitative feedback for reinforcement learning (Section 2), which we motivate with an example from the domain of chess, where annotated game traces provide a source for feedback in the form of action and state preferences. We then show how such preferences can be embedded into a formal framework for preference-based reinforcement learning, which is based on preferences between trajectories (Section 3). For a first instantiation of this algorithm, we build upon a policy learning approach called approximate policy iteration, which reduces the problem to iteratively learning a policy in the form of a classifier that predicts the best action in a state. We introduce a preference-based variant of this algorithm by replacing the classifier with a label ranker, which is able to make better use of the information provided by roll-out evaluations of all actions in a state. Preference learning and label ranking are briefly recapitulated in Section 4, and their use for policy iteration is introduced in Section 5. 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 this preference-based policy iteration method are illustrated by means of two case studies presented in Sections 6 and 7. In the last two sections, we discuss our plans for future work and conclude the paper.[1]

2 Reinforcement Learning and Qualitative Feedback

In this section, we will informally introduce a framework for reinforcement learning from qualitative feedback. We will start with a brief recapitulation of conventional reinforcement learning (Section 2.1) and then discuss our alternative proposal, which can be seen as a generalization that does not necessarily require a numerical feedback signal but is also able to exploit qualitative feedback (Section 2.2). Finally, we illustrate this setting using the game of chess as an example domain (Section 2.3).

2.1 Reinforcement Learning

Conventional reinforcement learning assumes a scenario in which an agent moves through a (finite) state space by taking different actions. Occasionally, the agent receives feedback about its actions in the form of a reward signal. The goal of the agent is to choose its actions so as to maximize its expected

[1] A preliminary version of this paper appeared as (Cheng et al. 2011).

total reward. Thus, reinforcement learning may be considered to be half-way between unsupervised learning (where the agent does not receive any form of feedback) and supervised learning (where the agent would be told the correct action in certain states).

The standard formalization of a reinforcement learning problem builds on the notion of a Markov Decision Process (MDP; Puterman 2005) and consists of

- a set of *states* $S = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ in which the agent operates; normally, a state does not have an internal structure, though it may be described in terms of a set of features (which allows, for example, functional representations of policies);
- a (finite) set of *actions* $A = \{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$ the agent can perform; sometimes, only a subset $A(\mathbf{s}_i) \subset A$ of actions is applicable in a state $\mathbf{s}_i$;
- 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; thus, $\tau(\mathbf{s}, \boldsymbol{a}, \mathbf{s'}) = \delta(\mathbf{s}, \boldsymbol{a})(\mathbf{s'})$ is the probability that action $\boldsymbol{a}$ in state $\mathbf{s}$ leads the agent to state $\mathbf{s'}$.
- a *reward function* $r : S \times A \to \mathbb{R}$, where $r(\mathbf{s}, \boldsymbol{a})$ is the reward the agent receives for performing action $\boldsymbol{a}$ in state $\mathbf{s}$; the concrete reward may depend on the successor state, in which case $r(\mathbf{s}, \boldsymbol{a})$ is given by the expectation of $r(\mathbf{s}, \boldsymbol{a}, \mathbf{s'})$ with respect to $\delta(\mathbf{s}, \boldsymbol{a})$.

Often, the training of the RL agent is organized into so-called *episodes* or *state-action sequences*

$$\Big(\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \ldots, \mathbf{s}_n, \mathbf{a}_n, \mathbf{s}_{(n+1)}\Big),$$

wherein $\mathbf{s}_0$ is the state the agent starts with, $\mathbf{a}_i$ the action it takes in state $\mathbf{s}_i$, and $\mathbf{s}_{i+1}$ the successor state produced by this action. The associated accumulated reward $\sum_{t=0}^{n} r(\mathbf{s}_t, \mathbf{a}_t)$ is an indicator of how appropriate the actions have been chosen.

Learning from episodes and accumulated rewards is natural in many reinforcement learning settings. In robotics, for example, each action (for example, a movement) of the robot may cause a certain cost, hence a negative reward, and these cost values are accumulated until a certain goal state is reached (for example, the robot finds itself in a desired spatial position). Another example is reinforcement learning in games, where each state-action sequence is one game. This example is somewhat special in the sense that a true (non-zero) reward signal only comes at the very end, indicating whether the game was won or lost.

The most common task in RL 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 $\mathbf{s}$), with future rewards being discounted by a factor $\gamma \in [0, 1]$:

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

where $\boldsymbol{\sigma} = (\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \ldots)$ is the trajectory produced by π through the state space.

With $V^*(\mathbf{s}) = \sup_{\pi \in \Pi} V^\pi(s)$ the best possible value that can be achieved for (1), a policy is called optimal if it achieves the best value in each state $\mathbf{s}$. Thus, one possibility to learn an optimal policy is to learn an evaluation of states in the form of a value function (Sutton 1988), or to learn a so-called *Q-function* that returns the expected reward for a given state-action pair (Watkins and Dayan 1992).

2.2 Learning from Qualitative Feedback

Existing algorithms for reinforcement learning possess interesting theoretical properties and have been used successfully in a number of applications. However, they also exhibit some practical limitations, notably regarding the type of feedback they can handle.

On the one hand, these methods are rather demanding with regard to the training information requested by the learner. In order to learn a value function or a Q-function, feedback must be specified in the form of real-valued rewards. It is true that, in some applications, information of that type is naturally generated by the environment; for example, the waiting time of people in learning elevator control (Crites and Barto 1998), or the distance covered by a robot learning to walk. In general, however, precise numerical information is difficult to extract from the environment, and designing the reward function is part of framing the problem as a reinforcement learning problem. Sometimes, this may become difficult and largely arbitrary—as a striking though telling example, to which we shall return in Section 7, consider assigning a negative reward of -60 to the death of the patient in a medical treatment (Zhao et al. 2009). Likewise, in (robot) soccer, a corner ball is not as good as a goal but better than a throw-in; again, however, it is difficult to quantify the differences in terms of real numbers.

The main objective of this paper is to define a reinforcement learning framework that is not restricted to numerical, quantitative feedback, but also able to handle qualitative feedback expressed in the form of preferences over states, actions, or trajectories. Feedback of this kind is more natural and less difficult to acquire in many applications. As will be seen later on, comparing pairs of actions, states or trajectories instead of evaluating them numerically does also have a number of advantages from a learning point of view. Before describing our framework more formally in Section 3, we discuss the problem of chess playing as a concrete example for illustrating the idea of qualitative feedback.

2.3 Example: Qualitative Feedback in Chess

In games like chess, reinforcement learning algorithms have been successfully employed for learning meaningful evaluation functions (Baxter et al. 2000;

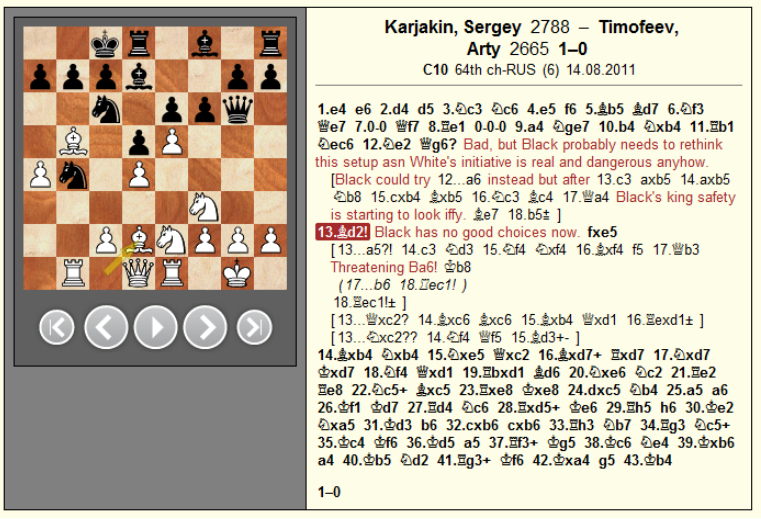

Fig. 1 An annotated chess game (screen-shot taken from `http://chessbase.com/`).

Beal and Smith 2001; Droste and Fürnkranz 2008). These approaches have all been modeled after the success of TD-Gammon (Tesauro 2002), a learning system that uses temporal-difference learning (Sutton 1988) for training a game evaluation function (Tesauro 1992). However, all these algorithms were trained exclusively on self-play, entirely ignoring human feedback that is readily available in annotated game databases.

Chess games are recorded in standardized notation (Edwards 1994), which provides a multitude of options for annotating chess games. Figure 1 shows an example. In addition to textual annotations, many moves are annotated with normed symbols, which have been popularized by the Chess Informant book series. The most important symbols are the following:

- *Qualitative move evaluation:* Each move can be annotated with a postfix that indicates the quality of the move. Six symbols are commonly used, representing different quality levels:
  - blunder (`??`),
  - bad move (`?`),
  - dubious move (`?!`),
  - interesting move (`!?`),
  - good move (`!`),
  - excellent move (`!!`).
- *Qualitative position evaluation:* Each position can be annotated with a symbol that indicates the qualitative value of this position:
  - white has decisive advantage (+−),
  - white has the upper hand (±),
  - white is better (⩲),

- equal chances for both sides (=),
- black is better (⩲),
- black has the upper hand (∓),
- black has decisive advantage (−+),
- the evaluation is unclear (∞).

For example, the left-hand side of Figure 1 shows the game position after the 13th move of white. Here, black made a mistake (13...♕g6?), but he is already in a difficult position. From the alternative moves, 13...a5?! is somewhat better, but even here white has the upper hand at the end of the variation (18.♖ec1!±). On the other hand, 13...♘×c2?? is an even worse choice, ending in a position that is clearly lost for black (+−).

It is important to note that this feedback is of qualitative nature, i.e., it is not clear what the expected reward is in terms of, e.g., percentage of won games from a position with evaluation ±. However, it is clear that positions with evaluation ± are preferable to positions with evaluation ⩱ or worse (=, ⩲, ∓, −+).

Also note that the feedback for positions typically applies to the entire sequence of moves that has been played up to reaching this position. The qualitative position evaluations may be viewed as providing an evaluation of the trajectory that lead to this particular position, whereas the qualitative move evaluations may be viewed as evaluations of the expected value of a trajectory that starts at this point.

Note, however, that even though there is a certain correlation between these two types of annotations (good moves tend to lead to better positions and bad moves tend to lead to worse positions), they are not interchangable. A very good move may be the only move that saves the player from imminent doom, but must not necessarily lead to a very good position. Conversely, a bad move may be a move that misses a chance to mate the opponent right away, but the resulting position may still be good for the player.

3 A Formal Framework for Preference-Based Reinforcement Learning

In this section, we define our framework in a more rigorous way. To this end, we first introduce a preference relation on trajectories. Based on this relation, we then derive a preference order on policies, and eventually on states and actions.

3.1 Preferences over Trajectories

Our point of departure is *preferences over trajectories*, that is, over the set Σ of all conceivable trajectories that may be produced by an RL agent in a certain environment.[2] Thus, our main assumption is that, given two (finite or

[2] Note that if S is countable, then Σ is countable, too.

infinite) trajectories $\boldsymbol{\sigma}, \boldsymbol{\sigma}' \in \Sigma$ (typically though not necessarily both starting in the same initial state), a user is able to compare them in terms of preference. In conventional RL, this comparison is reduced to the comparison of the associated (discounted) cumulative rewards: $\boldsymbol{\sigma} = (\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \ldots)$ is preferred to $\boldsymbol{\sigma}' = (\mathbf{s}'_0, \mathbf{s}'_1, \mathbf{s}'_2, \ldots)$ if the rewards accumulated along $\boldsymbol{\sigma}$ are higher than those accumulated along $\boldsymbol{\sigma}'$, that is, if[3]

$$\sum_t \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) > \sum_t \gamma^t r(\mathbf{s}'_t, \mathbf{a}'_t).$$

The comparison of trajectories in terms of cumulative rewards immediately induces a total order on Σ. Given that our qualitative framework does not necessarily allow for mapping trajectories to real numbers, we have to relax the assumption of completeness, that is, the assumption that each pair of trajectories can be compared with each other. Instead, we only assume Σ to be equipped with a *partial order relation* $\sqsupseteq$, where $\boldsymbol{\sigma} \sqsupseteq \boldsymbol{\sigma}'$ signifies that $\boldsymbol{\sigma}$ is at least as good as $\boldsymbol{\sigma}'$. Since the weak preference relation $\sqsupseteq$ is only partial, it allows for the incomparability of two trajectories. More specifically, it induces a strict preference ($\sqsupset$), an indifference ($\sim$) and an incomparability ($\perp$) relation as follows:

$$\begin{aligned}
\boldsymbol{\sigma} \sqsupset \boldsymbol{\sigma}' &\Leftrightarrow (\boldsymbol{\sigma} \sqsupseteq \boldsymbol{\sigma}') \wedge \neg(\boldsymbol{\sigma}' \sqsupseteq \boldsymbol{\sigma}) \\
\boldsymbol{\sigma} \sim \boldsymbol{\sigma}' &\Leftrightarrow (\boldsymbol{\sigma} \sqsupseteq \boldsymbol{\sigma}') \wedge (\boldsymbol{\sigma}' \sqsupseteq \boldsymbol{\sigma}) \\
\boldsymbol{\sigma} \perp \boldsymbol{\sigma}' &\Leftrightarrow \neg(\boldsymbol{\sigma} \sqsupseteq \boldsymbol{\sigma}') \wedge \neg(\boldsymbol{\sigma}' \sqsupseteq \boldsymbol{\sigma})
\end{aligned}$$

To illustrate the idea of incomparability, consider the case where an RL agent pursues several goals simultaneously. This can be captured by means of a vector-valued reward function measuring multiple instead of a single criterion, a setting known as *multiobjective reinforcement learning* (Gábor et al. 1998; Mannor and Shimkin 2004). An example of this kind will be studied in Section 7 in the context of medical therapy planning. Since a trajectory $\boldsymbol{\sigma}$ can be better than a trajectory $\boldsymbol{\sigma}'$ in terms of one objective but worse with respect to another, some trajectories may be incomparable in this setting.

3.2 Preferences over Policies

What we are eventually interested in, of course, is preferences over actions or, more generally, policies. Note that a policy π (together with an initial state or an initial probability distribution over states) induces a *probability distribution* over the set Σ of trajectories. In fact, fixing a policy π means fixing a state transition function, which means that trajectories are produced by a simple Markov process.

[3] Note that, formally, we here assume the rewards to depend only on states, while being independent of the actions that have let to a state. This is not a true restriction, since information about actions can be incorporated in an extended state description.

What is needed, therefore, is a preference relation over $\mathfrak{P}(\Sigma)$, the set of all probability distributions over Σ. Again, the standard setting of RL is quite reductionistic in this regard:

1. First, trajectories are mapped to real numbers (accumulated rewards), so that the comparison of probability distributions over Σ is reduced to the comparison of distributions over $\mathbb{R}$.
2. This comparison is further simplified by mapping probability distributions to their expected value. Thus, a policy π is preferred to a policy π' if the expected accumulated reward of the former is higher than the one of the latter.

Since the first reduction step (mapping trajectories to real numbers) is not feasible in our setting, we have to compare probability distributions over Σ directly. What we can exploit, nevertheless, is the order relation $\sqsupseteq$ on Σ. Indeed, recall that a common way to compare probability distributions on *totally* ordered domains is *stochastic dominance*. More formally, if X is equipped with a total order $\geq$, and P and P' are probability distributions on X, then P dominates (is preferred to) P' if $P(X_a) \geq P'(X_a)$ for all $a \in X$, where $X_a = \{x \in X \,|\, x \geq a\}$. Or, to put it in words, the probability to reach a or something better is always as high under P as it is under P'.

However, since $\sqsupseteq$ is only a partial order, stochastic dominance cannot be applied immediately. What is needed, therefore, is a generalization of this concept to the case of partial orders: What does it mean for P to dominate P' if both distributions are defined on a partially ordered set? This question is interesting and non-trivial, especially since both distributions may allocate probability mass on elements that are not comparable. Somewhat surprisingly, it seems that the generalization of stochastic dominance to distributions over partial orders has not received much attention in the literature so far, with a few notable exceptions.

We adopt a generalization proposed by Massey (1987). Roughly speaking, the idea of this approach is to find a proper replacement for the (half-open) intervals X_a that are used to define stochastic dominance in the case of total orders. To this end, he makes use of the notion of an *increasing set*. For any subset $\Gamma \subset \Sigma$, let

$$\Gamma^{\uparrow} = \{\boldsymbol{\sigma} \in \Sigma \,|\, \boldsymbol{\sigma} \sqsupseteq \boldsymbol{\sigma}' \text{ for some } \boldsymbol{\sigma}' \in \Gamma\}.$$

Then, a set Γ is called an increasing set if $\Gamma = \Gamma^{\uparrow}$. Informally speaking, this means that Γ is an increasing set if it is closed under addition of "better" elements: For no element in Γ, there is an element which, despite being at least as good, is not yet included.

The idea, then, is to say that a probability distribution P on Σ stochastically dominates a distribution P', if $P(\Gamma) \geq P'(\Gamma)$ for all $\Gamma \in \mathcal{I}(\Sigma)$, where $\mathcal{I}(\Sigma)$ is a suitable family of increasing sets of Σ. One reasonable choice of such a family, resembling the "pointwise" generation of (half-open) intervals X_a in the case of total orders through the elements $a \in X$, is

$$\mathcal{I}(\Sigma) \,=\, \{\,\{\boldsymbol{\sigma}\}^{\uparrow} \,|\, \boldsymbol{\sigma} \in \Sigma\} \cup \{\Sigma, \emptyset\}.$$

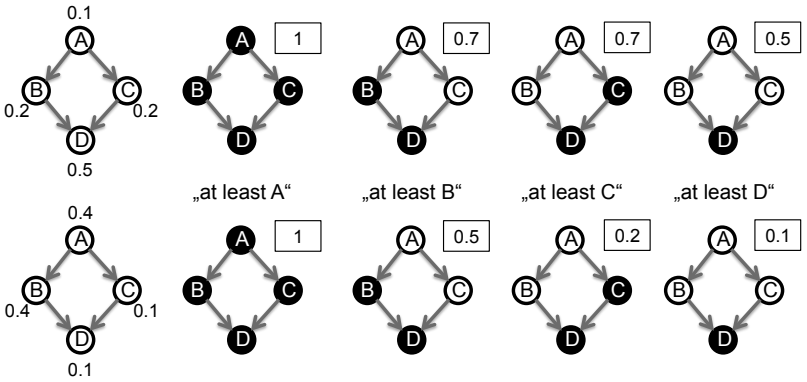

Fig. 2 Illustration of generalized stochastic dominance: A partial order on a set of four elements A, B, C, D (e.g., trajectories) is indicated by the corresponding Hasse diagram. Suppose that a probability distribution P assigns, respectively, the probabilities 0.1, 0.2, 0.2, 0.5 to these elements (above), while a distribution P' assigns probabilities 0.4, 0.4, 0.1, 0.1 (below). Then, P dominates P', since the probabilities of all increasing sets ("at least A", "at least B", "at least C", "at least D"), as indicated in the framed boxes, are larger under P than under P'.

See Fig. 2 for a simple illustration.

Recalling that policies are associated with probability distributions over Σ, it is clear that a dominance relation on such distributions immediately induces a preference relation $\succeq$ on the class Π of policies: $\pi \succeq \pi'$ if the probability distribution associated with π dominates the one associated with π'. It is also obvious that this relation is reflexive and antisymmetric. An important question, however, is whether it is also transitive, and hence a partial order. This is definitely desirable, and in fact already anticipated by the interpretation of $\succeq$ as a preference relation.

Massey (1987) shows that the above dominance relation is indeed transitive, provided the family $\mathcal{I}(\Sigma)$ fulfills two technical requirements. First, it must be *strongly separating*, meaning that whenever $\boldsymbol{\sigma} \not\sqsupseteq \boldsymbol{\sigma'}$, there is some $\Gamma \in \mathcal{I}(\Sigma)$ such that $\boldsymbol{\sigma'} \in \Gamma$ and $\boldsymbol{\sigma} \notin \Gamma$ (that is, if $\boldsymbol{\sigma'}$ is not worse than $\boldsymbol{\sigma}$, then it can be separated from $\boldsymbol{\sigma}$). The second condition is that $I(\Sigma)$ is a *determining class*, which, roughly speaking, means that it is sufficiently rich, so that each probability measure is uniquely determined by its values on the sets in $I(\Sigma)$.

In summary, provided these technical conditions are met, the partial order $\sqsupseteq$ on the set of trajectories Σ that we started with induces a partial order $\succeq$ on the set of policies Π. Of course, since $\succeq$ is only a partial order, there is not necessarily a unique optimal policy. Instead, optimality ought to be defined in terms of (Pareto) dominance: A policy $\pi^* \in \Pi$ is optimal if there is no other policy π' such that $\pi' \succ \pi$. We denote the set of all optimal policies by Π^*.

3.3 Preferences on States and Actions

From the preference relation $\succeq$ on policies, a preference relation on actions can be deduced. For an action $\mathbf{a}$ and a state $\mathbf{s}$, denote by $\Pi(\mathbf{s}, \mathbf{a})$ the set of policies π such that $\pi(\mathbf{s}) = \mathbf{a}$. It is then natural to say that $\mathbf{a}$ is an optimal action in state $\mathbf{s}$ if $\Pi(\mathbf{s}, \mathbf{a}) \cap \Pi^* \neq \emptyset$, that is, if there is an optimal policy $\pi^* \in \Pi^*$ such that $\pi^*(\mathbf{s}) = \mathbf{a}$. Again, note that there is not necessarily a unique optimal action.

The above condition only distinguishes between optimal and non-optimal actions in a given state. Of course, it might be desirable to discriminate between non-optimal actions in a more granular way, that is, to distinguish different levels of non-optimality. This can be done by applying the above choice principle in a recursive manner: With Π^{**} denoting the non-dominated policies in $\Pi \setminus \Pi^*$, the "second-best" actions are those for which $\Pi(\mathbf{s}, \mathbf{a}) \cap \Pi^{**} \neq \emptyset$. Proceeding in this way, that is, successively removing the optimal (non-dominated) policies and considering the optimal ones among those that remain, a Pareto ranking (Srinivas and Deb 1995) on the set of actions is produced. As mentioned earlier, this relation might be weak, i.e., there might be ties between different actions.

In the same way, one can derive a (weak) linear order on states from the preference relation $\sqsupseteq$ on trajectories. Let $\Sigma(\mathbf{s}) \subset \Sigma$ denote the set of trajectories originating in state $\mathbf{s}$, and let Σ^* be the set of trajectories that are non-dominated according to $\sqsupseteq$. Then, a state is optimal (i.e., has Pareto rank 1) if $\Sigma(\mathbf{s}) \cap \Sigma^* \neq \emptyset$, second-best (Pareto rank 2) if $\Sigma(\mathbf{s}) \cap \Sigma^{**} \neq \emptyset$, with Σ^{**} the set of non-dominated trajectories in $\Sigma \setminus \Sigma^*$, etc.

An important theoretical question, which is, however, beyond the scope of this paper, concerns the mutual dependence between the above preference relations on trajectories, states and actions, and a simple characterization of these relations. An answer to this question is indeed a key prerequisite for developing qualitative counterparts to methods like value and policy iteration. For the purpose of this paper, this point is arguably less relevant, since our approach of preference-based policy iteration, to be detailed in Section 5, is explicitly based on the construction and evaluation of trajectories through rollouts. The idea, then, is to learn the above linear order (ranking) of actions (as a function of the state) from these trajectories. A suitable learning method for doing so will be introduced in Section 5.

Finally, we note that learning a ranking function of the above kind is also a viable option in the standard setting where rewards are numerical, and policies can therefore be evaluated in terms of expected cumulative rewards (instead of being compared in terms of generalized stochastic dominance). In fact, the case study presented in Section 6 will be of that type.

4 Preference Learning and Label Ranking

The topic of *preference learning* has attracted considerable attention in machine learning in recent years (Fürnkranz and Hüllermeier 2010). Roughly speaking, preference learning is about inducing predictive preference models from empirical data, thereby establishing a link between machine learning and research fields related to preference modeling and decision making.

4.1 Utility Functions vs. Preference Relations

There are two main approaches to representing preferences, namely in terms of

- *utility functions* evaluating individual alternatives, and
- *preference relations* comparing pairs of competing alternatives.

The first approach is quantitative in nature, in the sense that a utility function is normally a mapping from alternatives to real numbers. This approach is in line with the standard RL setting: A value $V(\mathbf{s})$ assigned to a state $\mathbf{s}$ by a value function V can be seen as a utility of that state. Likewise, a Q-function assigns a degree of utility to an action, namely, $Q(\mathbf{s}, \mathbf{a})$ is the utility of choosing action $\mathbf{a}$ in state $\mathbf{s}$. The second approach, on the other hand, is qualitative in nature, as it is based on comparing alternatives in terms of qualitative preference relations.[4] The main idea of our paper is to exploit this approach in the context of RL.

From a machine learning point of view, the two approaches give rise to two kinds of learning problems: learning utility functions and learning preference relations. The latter deviates more strongly than the former from conventional problems like classification and regression, as it involves the prediction of complex structures, such as rankings or partial order relations, rather than single values. Moreover, training input in preference learning will not, as it is usually the case in supervised learning, be offered in the form of complete examples but may comprise more general types of information, such as relative preferences or different kinds of indirect feedback and implicit preference information.

4.2 Label Ranking

Among the problems in the realm of preference learning, the task of "learning to rank" has probably received the most attention in the machine learning literature so far. In general, a preference learning task consists of some set of items for which preferences are known, and the task is to learn a function that predicts preferences for a new set of items, or for the same set of items in a different context. The preferences can be among a set of objects, in which case we speak of *object ranking* (Kamishima et al. 2010), or among a set of labels

[4] Although the valuation of such relations is possible, too.

that are attached to a set of objects, in which case we speak of *label ranking* (Vembu and Gärtner 2010).

The task of a policy is to pick one of a set of available actions for a given state. This setting can be nicely represented as a label ranking problem, where the task is to rank a set of actions (labels) in dependence of a state description (object). More formally, 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 of all labels

$$y_{\tau_{\mathbf{x}}^{-1}(1)} \succ_{\mathbf{x}} y_{\tau_{\mathbf{x}}^{-1}(2)} \succ_{\mathbf{x}} \ldots \succ_{\mathbf{x}} y_{\tau_{\mathbf{x}}^{-1}(k)}$$

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

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

4.3 Learning by Pairwise Comparison

Several methods for label ranking have already been proposed in the literature; we refer to (Vembu and Gärtner 2010) for a comprehensive survey. In this paper, we chose *learning by pairwise comparison* (LPC; Hüllermeier et al. 2008), but other choices would be possible. The key idea of LPC 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 $\mathbf{x}$ is submitted to all models, and each prediction $\mathcal{M}_{i,j}(\mathbf{x})$ is interpreted as a vote for a label. More specifically, assuming scoring classifiers that produce normalized scores $f_{i,j} = \mathcal{M}_{i,j}(\mathbf{x}) \in [0, 1]$, 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., $y^* = \arg\max_i \sum_{j \neq i} f_{i,j}$.

Note that the total complexity for training the quadratic number of classifiers of the LPC approach is only linear in the number of observed preferences (Hüllermeier et al. 2008). More precisely, the learning complexity of LPC is $O(n \times d)$, where n is the number of training examples, and d is the average number of preferences that have been observed per state. In the worst case (for each training example we observe a total order of the labels), d can be as large as $k \cdot (k-1)/2$ (k being the number of labels). However, in many practical problems, it is considerably smaller.

Algorithm 1 ROLLOUT$(E, s_1, \gamma, \pi, K, L)$: Estimation of state values

Require: generative environment model E, sample state $\mathbf{s}_1$, discount factor γ, policy π, number of trajectories/roll-outs K, max. length/horizon of each trajectory L

```
for k = 1 to K do
  s ← s_1, Q̃_k ← 0, t ← 1
  while t < L and ¬TerminalState(s) do
    (s', r) ← Simulate(E, s, π(s))
    Q̃_k ← Q̃_k + γ^t r
    s ← s', t ← t + 1
  end while
end for
Q̃ = (1/K) Σ_{k=1}^{K} Q̃_k
return Q̃
```

Querying the quadratic number of classifiers can also be sped up considerably so that the best action can be determined after querying only approximately $k \cdot \log(k)$ classifiers (Park and Fürnkranz 2012). Thus, the main obstacle for tackling large-scale problems with LPC is the memory required for storing the quadratic number of classifiers. Nevertheless, Loza Menćia et al. (2010) have shown that it is applicable to multilabel problems with up to half a million examples and up to 1000 labels.

We refer to (Hüllermeier et al. 2008) for a more detailed description of LPC in general and a theoretical justification of the weighted voting procedure in particular. We shall use label ranking techniques in order to realize our idea of preference-based approximate policy iteration, which is described in the next section.

5 Preference-Based Policy Iteration

In this section, we describe a first, simple instantiation of preference-based reinforcement learning. We build on previous work on approximate policy iteration, which essentially reduces policy learning to a classification problem (Section 5.1). We then show how we can generalize this approach to a preference-based formulation of policy iteration, using a label ranker instead of a classifier (Section 5.2), and finally discuss how this framework can be implemented in a setting in which preferences are determined via roll-outs (Section 5.3).

5.1 Approximate Policy Iteration

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

Algorithm 2 Multi-class variant of Approx. Policy Iteration with Roll-Outs (Lagoudakis and Parr 2003)

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 L, max. number of policy iterations p

1: $\pi' \leftarrow \pi_0,\ i \leftarrow 0$
2: **repeat**
3: $\quad \pi \leftarrow \pi',\ \mathcal{E} \leftarrow \emptyset$
4: $\quad$ **for each** $\mathbf{s} \in S$ **do**
5: $\quad\quad$ **for each** $\mathbf{a} \in A$ **do**
6: $\quad\quad\quad (\mathbf{s}', r) \leftarrow \text{SIMULATE}(E, \mathbf{s}, \mathbf{a})$ # do (possibly off-policy) action $\mathbf{a}$
7: $\quad\quad\quad \tilde{Q}^{\pi}(\mathbf{s}, \mathbf{a}) \leftarrow \text{ROLLOUT}(E, \mathbf{s}', \gamma, \pi, K, L) + r$ # estimate state-action value
8: $\quad\quad$ **end for**
9: $\quad\quad \mathbf{a}^* \leftarrow \arg\max_{\mathbf{a} \in A} \tilde{Q}^{\pi}(\mathbf{s}, \mathbf{a})$
10: $\quad\quad$ **if** $\tilde{Q}^{\pi}(\mathbf{s}, \mathbf{a}^*) >_L \tilde{Q}^{\pi}(\mathbf{s}, \mathbf{a})$ for all $\mathbf{a} \in A, \mathbf{a} \neq \mathbf{a}^*$ **then** # please see text for $>_L$
11: $\quad\quad\quad \mathcal{E} \leftarrow \mathcal{E} \cup \{(\mathbf{s}, \mathbf{a}^*)\}$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: $\quad \pi' \leftarrow \text{LEARN}(\mathcal{E}),\ i \leftarrow i + 1$
15: **until** $\text{STOPPINGCRITERION}(E, \pi, \pi', p, i)$

explicit policy (the actor) simultaneously (Barto et al. 1983; Konda and Tsitsiklis 2003; Bhatnagar et al. 2009), and policy gradient methods, which search for a good parameter setting in a space of parametrized policies (Williams 1992; Kersting and Driessens 2008; Sutton et al. 2000).

A particularly interesting approach is approximate policy iteration with roll-outs (Lagoudakis and Parr 2003; Dimitrakakis and Lagoudakis 2008). This approach assumes access to a generative model E of the underlying process, i.e., a model which takes a state $\mathbf{s}$ and an action $\mathbf{a}$ as input and returns a successor state $\mathbf{s}'$ and the reward $r(\mathbf{s}, \mathbf{a})$. The key idea, then, is to use this generative model 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 $\mathbf{s}_1 = \delta(\mathbf{s}, \mathbf{a})$. 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 L steps, and finally accumulating the observed rewards. This is repeated K times, and the average reward over these K roll-outs is used to approximate the Q-value $\tilde{Q}^{\pi}(\mathbf{s}_0, \mathbf{a})$ for taking action $\mathbf{a}$ in state $\mathbf{s}_0$ (leading to $\mathbf{s}_1$) and following policy π thereafter.

These roll-outs are 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 $>_L$ in line 10), a training example $(\mathbf{s}, \mathbf{a}^*)$ is added to a training set $\mathcal{E}$. Eventually, $\mathcal{E}$ 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).

Algorithm 3 Preference-Based Policy Iteration.

Require: sample states S, initial (random) policy π_0, max. number of policy iterations p, procedure EvaluatePreference for determining the preference between a pair of actions for a given policy in a given state.

```
1: π' ← π_0, i ← 0
2: repeat
3:    π ← π', E ← ∅
4:    for each s ∈ S do
5:       for each (a_k, a_j) ∈ A × A do
6:          EvaluatePreference(s, a_k, a_j, π)
7:          if a_k ≻_s a_j then
8:             E ← E ∪ {(s, a_k ≻ a_j)}
9:          end if
10:      end for
11:   end for
12:   π' ← LearnLabelRanker(E), i ← i + 1
13: until StoppingCriterion(π, π', p, i)
```

The choice of the sampling procedure to generate $\mathcal{E}$ is not a trivial task as discussed in (Lagoudakis and Parr 2003; Dimitrakakis and Lagoudakis 2008). Choices of procedures range from simple uniform sampling of the state space, sampling schemes incorporating domain-expert knowledge and other more sophisticated schemes. We emphasize that we do not contribute to this aspect and will use in this work rather simple sampling schemes, which will be described in detail in the experimental sections.

We should note some minor differences between the version presented in Algorithm 2 and the original formulation (Lagoudakis and Parr 2003). Most notably, the training set here is formed as a multi-class training set, whereas in (Lagoudakis and Parr 2003) 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.

5.2 Preference-Based Policy Iteration

Following our idea of preference-based RL, we propose to train a *label ranker* instead of a *classifier*: Using the notation from Section 4.2 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 $(\mathbf{s}, \mathbf{a}_k \succ \mathbf{a}_j)$, indicating that in state $\mathbf{s}$, action $\mathbf{a}_k$ is preferred to action $\mathbf{a}_j$.

Algorithm 3 shows a high-level view of the preference-based policy iteration algorithm PBPI. Like Algorithm 2, it starts with a random policy π_0 and continously improves this policy by collecting experience on a set of sample

states S. However, in each state $\mathbf{s}$, PBPI does not evaluate each individual action $\mathbf{a}$, but pairs of preferences $(\mathbf{a}_k, \mathbf{a}_j)$. For each such pair of preferences, we then call a routine EvaluatePreference which determines whether $\mathbf{a}_k \succeq \mathbf{a}_j$ holds in state $\mathbf{s}$ or not. If it holds, the training example $(\mathbf{s}, \mathbf{a}_k \succ \mathbf{a}_j)$ is added to the training set $\mathcal{E}$. If all states in S have been evaluated in that way, a new policy in the form of a label ranker is learned from $\mathcal{E}$. This process is iterated until the policies converge or a predetermined number p of iterations has been reached.

Note that in practice, looping through all pairs of actions may not be necessary. For example, in our motivating chess example, preferences may only be available for a few action pairs per state, and it might be more appropriate to directly enumerate the preferences. Thus, the complexity of this algorithm is essentially $O(|S| \cdot d)$, where d is the average number of observed preferences per state (the constant number of iterations is hidden in the $O(.)$ notation). This complexity directly corresponds to the complexity of LPC as discussed in Section 4.3. If a total order of actions is observed in each state (as is the case in the study in Section 6), the complexity may become as bad as linear in the number of visited states and quadratic in the number of actions. However, if only a few preferences per state can be observed (as is e.g., the case in our motivating chess example), the complexity is only linear in the number of visited states $|S|$ and essentially independent of the number of actions (e.g. if we only compare a single pair of actions for each state). Of course, in the latter case, we might need considerably more iterations to converge, but this cannot be solved with the choice of a different label ranking algorithm. Thus, we recommend the use of LPC for settings where very few action preferences are observed per state. Its use is not advisable for problems with very large action spaces, because the storage of a quadratic number of classifiers may become problematic in such cases.

The key point of the algorithm is the implementation of EvaluatePreference, which determines the preference between two actions in a state. As discussed in Section 2, this can be realized in different ways. In particular, we can always reduce the conventional utility-based setting to this particular case by defining the preference $\mathbf{a}_k \succeq_{\mathbf{s}} \mathbf{a}_j$ as "in state $\mathbf{s}$, policy π gives a higher expected reward for $\mathbf{a}_k$ than for $\mathbf{a}_j$". This is the approach that we take in the special case that we discuss in the next section, and evaluate in the remainder the paper.

However, we note that we could also implement this preference by counting the success of each of the two actions $\mathbf{a}_k$ and $\mathbf{a}_j$ in state $\mathbf{s}$, and performing a sign test for determining the overall preference. This approach allows to entirely omit the aggregation of utility values and instead only aggregates preferences. In preliminary experiments, we noted essentially the same qualitative results as those reported below. The results are, however, not directly comparable to the results of approximate policy iteration, because the sign test is more conservative than the t-test, which is used in API. For this reason, the comparative evaluation in Section 6 focuses on a quantitative preference definition.

5.3 Preference-Based Approximate Policy Iteration

Recall the scenario described at the end of Section 5.1, where the agent has access to a generative model E, which takes a state $\mathbf{s}$, and an action $\mathbf{a}$ as input and returns a successor state $\mathbf{s}'$ and the reward $r(\mathbf{s}, \mathbf{a})$. Lagoudakis and Parr (2003) use this setting for generating training examples via roll-outs, i.e., by using the generative model and the current policy π for generating a training set $\mathcal{E}$, which is in turn used for training a multi-class classifier that can be used as a policy. Instead of training a classifier on the optimal action in each state, we can instead use the PBPI algorithm to train a label ranker on all pairwise comparisons of actions in each state. We call this algorithm *preference-based approximate policy iteration (PBPI)*.

Note that generating the training information for PBAPI is no more expensive than generating the training information for API, because in both cases all actions in a state have to be run for a number of iterations. On the contrary, we argue that 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 6 and 7 utilize this in different ways.

Section 6 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 by Lagoudakis and Parr (2003) for forming training examples is very wasteful with this information. An example $(\mathbf{s}, \mathbf{a}^*)$ 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.[5] For the preference-based approach, on the other hand, it suffices if only two possible actions $\mathbf{a}_k$ and $\mathbf{a}_j$ yield a clear preference (either $\mathbf{a}_k \succ \mathbf{a}_j$ or $\mathbf{a}_j \succ \mathbf{a}_k$) 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 7, 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

[5] 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).

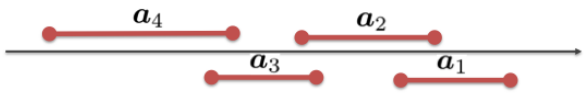

Fig. 3 Comparing actions in terms of confidence intervals on expected rewards: Although a best action cannot be identified uniquely, pairwise preferences like $\mathbf{a}_1 \succ \mathbf{a}_3$ and $\mathbf{a}_2 \succ \mathbf{a}_4$ can still be derived.

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.

6 Case Study I: Exploiting Action Preferences

In this section, we compare three variants of approximate policy iteration following Algorithms 2 and 3. 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 $(\mathbf{s}, \mathbf{a}^*)$ if $\mathbf{a}^*$ is the best available action in $\mathbf{s}$, i.e., if $\tilde{Q}^\pi(s, \mathbf{a}^*) >_L \tilde{Q}^\pi(s, \mathbf{a})$ 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 $(\mathbf{s}, \mathbf{a}^*)$ of API is transformed into $a - 1$ training examples of the form $(\mathbf{s}, \mathbf{a}^* \succ \mathbf{a})$ 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 $\tilde{Q}^\pi(s, \mathbf{a}_k) >_L \tilde{Q}^\pi(s, \mathbf{a}_l)$ holds for a pair of actions $(\mathbf{a}_k, \mathbf{a}_l)$, PBPI generates a corresponding training example $(\mathbf{s}, \mathbf{a}_k \succ \mathbf{a}_l)$.

Note that, in the last setting, an example $(\mathbf{s}, \mathbf{a}_k \succ \mathbf{a}_l)$ can be generated although $\mathbf{a}_k$ is not the best action. In particular, in contrast to PAPI, it is not necessary that there is a clear best action in order to generate training examples. Consequently, from the same roll-outs, PBPI will typically generate more training information than PAPI or API (see Figure 3). This is actually an interesting illustration of the increased flexibility of preference-based training information: Even if there might be no obvious "correct" alternative, some options may still be preferred to others.

The problems we are going to tackle in this section do still fall into the standard framework of reinforcement learning. Thus, rewards are numerical,

trajectories are evaluated by accumulated rewards, and policies by expected cumulative discounted rewards. The reason is that, otherwise, it is not possible to compare with the original approach to approximate policy iteration. Nevertheless, as will be seen, preference-based learning from qualitative feedback can even be useful in this setting.

6.1 Application Domains

Following (Dimitrakakis and Lagoudakis 2008), 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 (Dimitrakakis and Lagoudakis 2008), 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 S 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. For this setting, the random samples S were generated by uniform sampling over valid horizontal positions (excluding the goal state) and valid velocities.

6.2 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 used LPC (cf. Section 4.2) and for all three considered variants simple multi-layer perceptrons (as implemented in the Weka machine learning library (Hall et al. 2009) with its default parameters) was used as (base) learning algorithm. 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. Furthermore, for statistical testing unpaired t-tests assuming equal variance (homoscedastic t-test) were used.

For each task and method, the following parameter settings were evaluated:

- five numbers of state samples $|S| \in \{10, 20, 50, 100, 200\}$,
- five maximum numbers of roll-outs $K \in \{10, 20, 50, 100, 200\}$,
- 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.

6.3 Evaluation

Our prime evaluation measure is the *success rate* (SR), i.e., the percentage of learned sufficient policies. Following (Dimitrakakis and Lagoudakis 2008), 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 (Dimitrakakis and Lagoudakis 2008) 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.

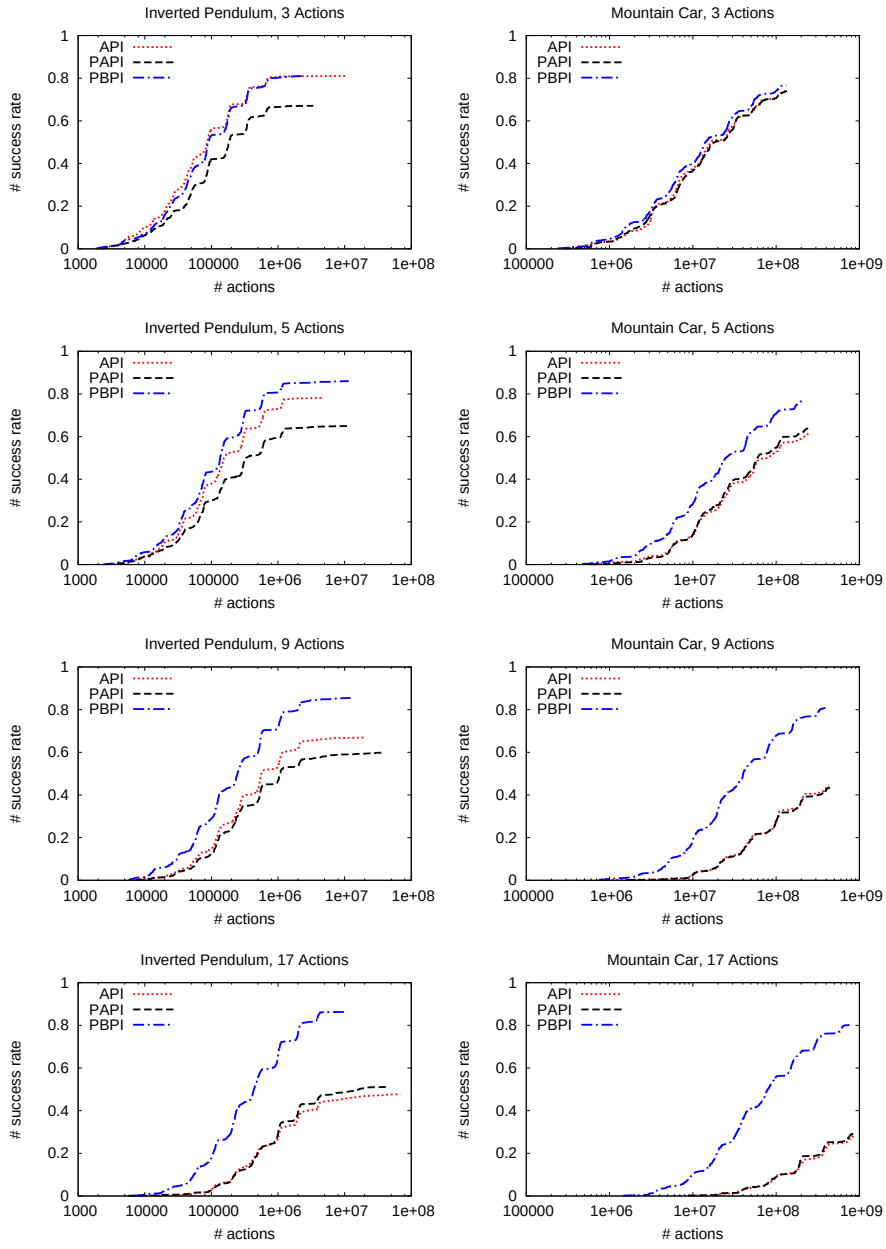

Fig. 4 Comparison of API, PAPI and PBPI for the inverted pendulum task (left) and the mountain car task (right). The number of actions is increasing from top to bottom.

Table 1 Training information generation for API, PAPI and PBPI. For each algorithm, we show the fraction of training states that could be used, as well as the fraction of the $\frac{1}{2} \cdot |A| \cdot (|A| - 1)$ possible preferences that could on average be generated from a training state. Note that the values for PAPI are only approximately true (the measures are taken from the API experiments and may differ slightly due to random issues).

		API/PAPI		PBPI	
	$\|A\|$	States	Preferences	States	Preferences
IP	3	0.589 ± 0.148	0.393 ± 0.099	0.736 ± 0.108	0.631 ± 0.119
	5	0.400 ± 0.162	0.160 ± 0.065	0.759 ± 0.101	0.581 ± 0.128
	9	0.273 ± 0.154	0.061 ± 0.034	0.776 ± 0.087	0.543 ± 0.124
MC	3	0.316 ± 0.218	0.211 ± 0.145	0.453 ± 0.245	0.349 ± 0.229
	5	0.231 ± 0.181	0.093 ± 0.072	0.510 ± 0.263	0.311 ± 0.216
	9	0.149 ± 0.125	0.033 ± 0.028	0.539 ± 0.273	0.281 ± 0.201

6.4 Complete State Evaluations

Figure 4 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.

Part of the problem is that API and PAPI are very wasteful with sample states. The third column of Table 1 shows the fraction of states for which a training example could be generated, i.e., for which one action turned out to be significantly better than all other actions. It can be clearly seen that this fraction decreases with the number of available actions in each problem (shown in the second column). It is not surprising that the number of training states from which PBPI could generate at least a training preference (column 5 of Table 1) is higher than API and PAPI in all cases. PBPI can generate some training information from a state when at least one pair of actions yields a significant difference, whereas API and PAPI only uses a state if one action is better than all other actions. Thus, this quantity is at least the amount of API/PAPI. The result that this quantity for PBPI is non-decreasing for increasing number of actions may be explained by the fact that to some extent a found preference for a given number of actions $|A|$ for a particular state is also existent in the next setting with a higher number of actions $|A'|$. Here, the set of possible preferences of A is a subset of the possible preferences of A'.

However, even if we instead look at the fraction of possible preferences that could be used, we see that there is only a small decay for the PBPI. This decay

can be explained by the fact that the more actions we have in the mountain car and inverted pendulum problems, the more similar they are to each other and the more likely it is that we can detect actions pairs that have approximately the same quality and cannot be discriminated via roll-outs. For API and PAPI, on the other hand, the decay in the number of generated preferences is the same as with the number of usable states, because each usable training state produces exactly $|A| - 1$ preferences.[6] Thus, the fourth column differs from the third by a factor of $|A|/2$.

6.5 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 be explored. Such an approach may thus be considered to be orthogonal to recent approaches for roll-out allocation strategies (cf. Section 8.2).

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 $\frac{|A|}{2}$, to account for the fact that the partial variant performs only 2 action roll-outs in each state, as opposed to $|A|$ 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 $\frac{|A|(|A|-1)}{2}$ preferences from each visited state, partial PBPI generates only one preference per state, and is thus allowed to visit $\frac{|A|(|A|-1)}{2}$ as many states. These modifications were integrated into Algorithm 2 by adapting Line 2 to iterate only over two randomly chosen actions and changing the number of considered sample states S to the values as described above.

Figure 5 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.

[6] Strictly speaking, only the best action for each state is generated and used within API, but for the sake of comparison in terms of preferences this directly relates to $|A| - 1$ preferences involving the best action.

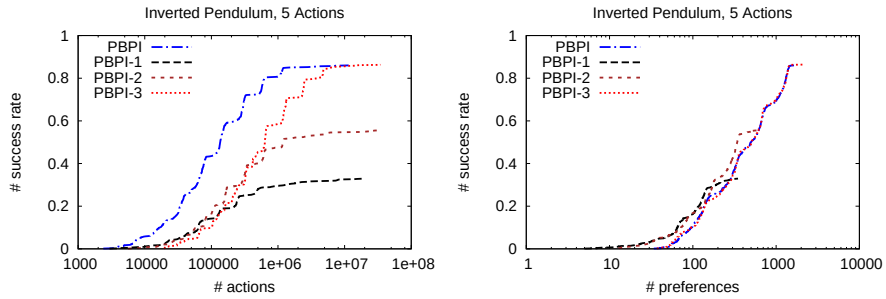

Fig. 5 Comparison of complete state evaluation (PBPI) with partial state evaluation in three variants (PBPI-1, PBPI-2, PBPI-3).

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. We will briefly return to this issue in Section 8.2.

7 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 (Zhao et al. 2009). 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.

7.1 Cancer Clinical Trials Domain

The model proposed in (Zhao et al. 2009) 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

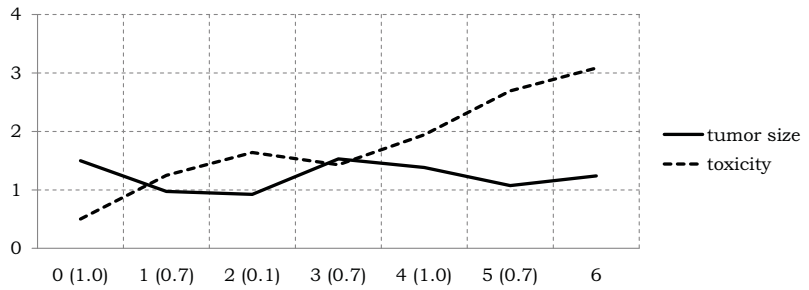

Fig. 6 Illustration of the simulation model showing the patient's status during the treatment. The initial tumor size is 1.5 and the initial toxicity is 0.5. On the x-axis is the month with the corresponding dosage level the patient receives. The dosage levels are selected randomly.

size Y and the toxicity X, are modeled using a system of difference equations: $Y_{t+1} = Y_t + \Delta Y_t$ 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 ΔY and ΔX indicate the increments of the state variables that depend on the action, namely the dosage level D, which is a number between 0 and 1 (minimum and maximum dosage, respectively):

$$\begin{aligned} \Delta Y_t &= \left[a_1 \cdot \max(X_t, X_0) - b_1 \cdot (D_t - d_1)\right] \times \mathbf{1}(Y_t > 0) \\ \Delta X_t &= a_2 \cdot \max(Y_t, Y_0) + b_2 \cdot (D_t - d_2) \end{aligned} \tag{2}$$

These changing rates produce a piecewise linear model over time. We fix the parameter values following the recommendation of (Zhao et al. 2009): $a_1 = 0.15, a_2 = 0.1, b_1 = b_2 = 1.2$ and $d_1 = d_2 = 0.5$. By using the indicator term $\mathbf{1}(Y_t > 0)$, 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: $\lambda(t) = \exp(c_0 + c_1 Y_t + c_2 X_t)$, where c_0, c_1, c_2 are cancer-dependent constants. Again following (Zhao et al. 2009), 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

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

7.2 A Preference-Based Approach

The problem is to learn an optimal treatment policy π mapping states (Y, X) to actions in the form of a dosage level D (recall that this is a number between

0 and 1). In (Zhao et al. 2009), 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, as outlined in Section 3, we proceed from a preference relation on trajectories, that is, we compare trajectories in a qualitative way. More specifically, trajectories $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma'}$ are compared as follows: $\boldsymbol{\sigma} \sqsupseteq \boldsymbol{\sigma'}$ if the patient survives under $\boldsymbol{\sigma}$ but not under $\boldsymbol{\sigma'}$, and both policies are incomparable if the patient does neither survive under $\boldsymbol{\sigma}$ nor under $\boldsymbol{\sigma'}$. Otherwise, if the patient survives under both policies, let C_X denote the maximal toxicity during the 6 months of treatment under $\boldsymbol{\sigma}$ and, correspondingly, C'_X under treatment $\boldsymbol{\sigma'}$. Likewise, let C_Y and C'_Y denote the respective size of the tumor at the end of the therapy. Then, we define preference via Pareto dominance as

$$\boldsymbol{\sigma} \sqsupseteq \boldsymbol{\sigma'} \quad \Leftrightarrow \quad (C_X \leq C'_X) \text{ and } (C_Y \leq C'_Y) \tag{3}$$

It is important to remark that $\sqsupseteq$ thus defined, as well as the induced preference order $\succeq$ on policies, are only *partial* order relations. In other words, it is thoroughly possible that two trajectories (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 (Lagoudakis and Parr 2003), 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 even applicable in the case of a partial order.

7.3 Experimental Setup

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, Y_0 and X_0, are generated independently and uniformly from $(0, 2)$. Then, for the following 6 months, the patient receives a monthly

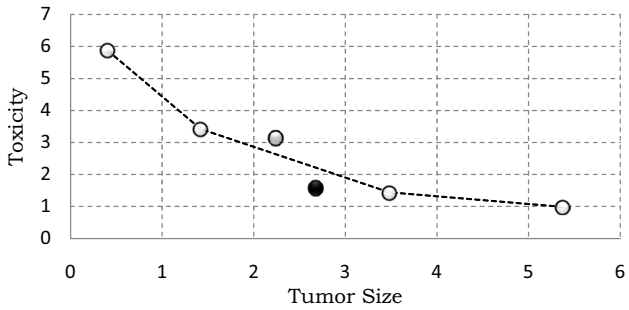

Fig. 7 Illustration of patients status under different treatment policies. On the x-axis is the tumor size after 6 months. On the y-axis is the highest toxicity during the 6 months. From top to bottom: Extreme dose level (1.0), high dose level (0.7), random dose level, learned dose level, medium dose level (0.4), low dose level (0.1). The values are averaged from 200 patients.

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.[7] As an illustration, Fig. 6 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.

Action preferences are generated via Pareto dominance relation (3) using roll-outs. Essentially this means that for each pair of actions $\mathbf{a}_k$ and $\mathbf{a}_j$ in a state $\mathbf{s}$, we compute 10 pairs of trajectories and compare each pair in terms of the above preference relation, i.e., the first trajectory is preferred to the second one if the latter involves the death of the patient and the former not or, in case the patient survives in both cases, we have a dominance relation in the sense of (3). Then, we generate a preference for $\mathbf{a}_k$ over $\mathbf{a}_j$ if the number of comparisons in favor of the former is significantly higher than the number of comparisons in favor of the latter (according to a simple sign test at significance level 0.1)

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 p reaches 10.

[7] 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.

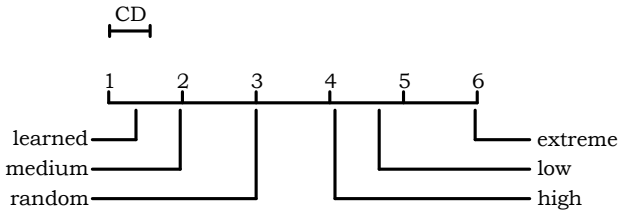

Fig. 8 Average ranks of the policies according to death rates. CD stands for the critical difference of ranks derived from a Nemenyi test at $\alpha = 0.05$.

7.4 Results

For testing, we generate 200 new virtual patients. In Fig. 7, the average values of the two criteria (C_X, C_Y) 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.

In a second evaluation, we focus on the death rate—which, after all, is a function of toxicity and tumor size. In each test trial, we now compute a patient's probability to survive during the whole treatment. Averaging over all patients, we rank the policies according to this criterion. The average ranks of policies are shown in Fig. 8. As can be seen, the treatments produced by our preference-based RL method have a lower death rate than the other polices. In fact, the Nemenyi test even indicates that the differences are statistically significant at a significance level of $\alpha = 0.05$ (Demšar 2006).

8 Related Work

In this section, we give an overview of existing work which is, in one way or the other, related to the idea of preference-based reinforcement learning as introduced in this paper. In Section 8.1, we start with policy search approaches that use the reinforcement signal for directly modifying the policy. The preference-based approach of Akrour et al. (2011) may be viewed in this context. Preference-based policy iteration is, on the other hand, closer related to approaches that use supervised learning algorithms for learning a policy

(Section 8.2). In Section 8.3, we discuss the work of Maes (2009), which tackles a quite similar learning problem, albeit in a different context and having other goals in mind. Our work is also related to multi-objective reinforcement learning, because in both learning settings, trajectories may not be comparable (Section 8.4). Finally we also discuss alternative approaches for incorporating external advice (Section 8.5) and for handling qualitative information (Section 8.6).

8.1 Preference-based Policy Search

Akrour et al. (2011) propose a framework that is quite similar to ours. In their architecture, the learning agent shows a set of policies to a domain expert who gives feedback in the form of pairwise preferences between the policies. This information is then used in order to learn to estimate the value of parametrized policies in a way that is consistent with the preferences provided by the expert. Based on the new estimates, the agent selects another set of policies for the expert, and the process is repeated until a termination criterion is met.

Thus, just like in our approach, the key idea of Akrour et al. (2011) is to combine preference learning and reinforcement learning, taking qualitative preferences between trajectories as a point of departure.[8] What is different, however, is the type of (preference) learning problem that is eventually solved: Akrour et al. (2011) seek to directly learn a ranking function in the policy space from global preferences on the level of complete trajectories, whereas we propose to proceed from training information in the form of local preferences on actions in a given state. Correspondingly, they solve an object ranking problem, with objects given by parametrized policies, making use of standard learning-to-rank methods (Kamishima et al. 2010).

In a way, the approach of Akrour et al. (2011) may be viewed as a preference-based variant of policy search. Just as so-called *policy gradient* methods search for a good parameter setting in a space of parametrized policies by using the reinforcement signal to derive the direction into which the policy should be corrected, the above-mentioned approach uses a qualitative preference signal for driving the policy learner towards better policies. Numerical policy gradients can be computed in closed form from a parametrized policy (Ng and Jordan 2000), be estimated empirically from action samples of the policy (Williams 1992), or learned by regression (Kersting and Driessens 2008). In the actor-critic framework, where algorithms learn both the value function (the critic) and an explicit policy (the actor) simultaneously (Barto et al. 1983), the policy gradient can be estimated from the predicted values of the value function (Konda and Tsitsiklis 2003; Sutton et al. 2000). A particularly interesting policy gradient approach is the *natural actor-critic* (Peters and Schaal 2008a). The key idea of this approach is to fight the large variance in conventional gradient approaches by the use of the natural gradient, i.e., the gradient that

[8] We assume here that policies are demonstrated to the user in the form of "representative" trajectories, but this is not entirely clear from the description in (Akrour et al. 2011).

does not assume that the parameter space is Euclidean but takes its structure into account (Amari 1998; Kakade 2001). Good surveys of current work in this area can be found in (Bhatnagar et al. 2009; Peters and Schaal 2008b).

8.2 Supervised Policy Learning

The above-mentioned approaches use the (qualitative or quantiative) reinforcement signal to directly optimize the policy, whereas preference-based approximate policy iteration is quite related to approaches that use supervised learning algorithms to learn a policy. In particular, our work directly builds upon approximate policy iteration (Lagoudakis and Parr 2003), which had the goal of using modern classifiers to learn the policy. In general, the key idea of such approaches is to learn a policy in the form of a P-function, which directly maps states to optimal actions (Tadepalli et al. 2004). The P-function can be represented in commonly used concept representations such as relational decision trees (Džeroski et al. 2001), decision lists (Fern et al. 2006), support vector machines (Lagoudakis and Parr 2003), etc. As the P-function needs to capture less information than the Q-function (it does not need to perfectly fit the Q-values), the hope is that it leads to more compact representations and to a faster convergence.

A key issue for such approaches is the strategy used for generating examples for the supervised learning algorithm. The original roll-out strategy proposed by Lagoudakis and Parr (2003) is rather wasteful with respect to the performed number of roll-outs, and several authors have tried to address this problem (Dimitrakakis and Lagoudakis 2008; Gabillon et al. 2010). First, the number of necessary roll-outs in a state, which we assumed to be a constant number K, can be dynamically adjusted so that the roll-outs are stopped as soon as a winning action clearly emerges. Hoeffding or Bernstein races can be used to determine the best action with a minimal number of roll-outs (Heidrich-Meisner and Igel 2009), elimination algorithms iteratively remove the worst action until a single winner emerges (Even-Dar et al. 2003; Audibert et al. 2010), and the UCB algorithm, which trades off exploration and exploitation in multi-armed bandit problems (Auer et al. 2002), can also be adapted to this setting (Dimitrakakis and Lagoudakis 2008). Recently, Gabillon et al. (2011) proposed to improve finite-horizon roll-out estimates by enhancing them with a critic which learns to estimate the value of the iterations beyond the horizon. All these approaches are, in a way, orthogonal to our approach, in that they all focus on the best action. In all of them, additional pairwise comparisons can emerge as a by-product of sampling for the best action, and the experiments of Section 6 show that it can be beneficial to use them.

A second approach for optimizing the use of roll-outs is to define a state exploration strategy. For example, it was suggested that a policy-based generation of states may be preferable to a random selection (Fern et al. 2006). While this may clearly lead to faster convergence in some domains, it may also fail to find optimal solutions in other cases (Lagoudakis and Parr 2003). Again,

such approaches can be straight-forwardly combined with our proposal. Moreover, preference-based approximate policy iteration provides a natural focus on the comparison between pairs of actions instead of sets of actions. This allows the use of fewer roll-outs for getting some information (one preference) from a state, and thus allows to move more quickly from state to state. For example, 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. Whether and how this additional flexibility can be used for more efficient exploration strategies is subject of future work.

Finally, Lazaric et al. (2010) propose *direct policy iteration*, which improves approximate policy iteration (Lagoudakis and Parr 2003), upon which our work builds, by optimizing a loss function that is based on the difference between the roll-out estimate of the action chosen by a policy and the maximum value obtainable by any action.

8.3 Reinforcement Learning for Structured Output Prediction

Maes (2009) solves a learning problem quite similar to ours, namely "policy learning as action ranking", and even makes use of pairwise learning techniques. More specifically, he learns a function called "action-ranking function" that ranks actions given states.

The context and the purpose of the approach, however, are quite different. The idea is to tackle structured output prediction problems with reinforcement learning: instead of predicting a structured output (such as a sequence or a tree) right away, the output is constructed step by step. This stepwise construction is modeled as following a path in a properly defined state space and, thus, can be cast as a reinforcement learning problem.

Apart from this difference, Maes (2009) makes use of ranking methods (instead of regression learning of action-value functions) for other reasons. While still making use of quantitative information about the costs of actions in a given state, he is mainly interested in facilitating the learning process and making it more efficient.

8.4 Multi-Objective Reinforcement Learning

In multi-objective reinforcement learning (MORL), the agent seeks to achieve two or more objectives at the same time, each with its own associated reward signal. Thus, unlike in standard RL, where the reward is a scalar, it is now a vector (Vamplew et al. 2011). Typically, the different objectives are in conflict with each other and cannot easily be optimized simultaneously. Instead, a policy must either optimize only one objective while neglecting the others, or try to find a trade-off between the conflicting criteria. What is sought, therefore, is a policy that is optimal in a Pareto sense.

MORL shares an important property with our approach to preference-based reinforcement learning, namely the fact that trajectories (policies) are

not necessarily comparable with each other. On the other hand, the reward signals in MORL are still numerical, thus making the problem amenable to other types of learning algorithms. In other words, MORL can be seen as a special case of our general framework; as such, it can be tackled by specialized algorithms that are presumably more effective than general algorithms for preference-based reinforcement learning.

8.5 External Advice and Off-Policy Learning

In addition to numerical reward functions, some authors have investigated ways for incorporating other forms of feedback, most notably external *advice*. For example, Maclin and Shavlik (1996) proposed an approach in which user-generated advice in the form of rules is transferred to the same neural network used by the reinforcement learning agent for learning the Q-function. Maclin et al. (2005) propose user-provided advice in the form of preferences over actions and use them for training a reinforcement learner via kernel-based regression. The constraints compare two actions, but are still quantitative in the sense of specifying a lower bound on the difference of the Q-values of two actions. These constraints can then be directly incorporated into the kernelized optimization algorithm. This form of advice has later been adapted for transfer learning (Torrey et al. 2005). A good survey of transfer learning in RL and its relation to advice-taking is given by Taylor and Stone (2009).

An alternative technique is to encode advice in the form of a reasonable starting policy, which can then be used to generate training examples for a relational reinforcement learner (Driessens and Džeroski 2004). Such examples could, e.g., also be generated by a human controller. Auer et al. (1995) consider the multi-armed bandit problem in the presence of experts, each of which instructs the learner on a particular arm to pull. Langford and Zhang (2008) consider a more general case where any contextual information can be considered in addition to the reward. For example, in a news recommender system, the context could describe news articles and user information (Li et al. 2010). Another approach for advice-taking with reinforcement learning has been tried by Fürnkranz et al. (2000). The key idea of this paper is tuning the weights of several advice-givers instead of the weights of an evaluation function.

The above is related to *off-policy learning*, where the learner does not have to follow the policy it tries to optimize. In the simplest case, when the value function is simply represented as a look-up table, off-policy learning is well understood. In fact, Q-learning is an off-policy learner. In the general case, however, when the state space is so complex that the value function has to be approximated, off-policy learning is known to become unstable even for linear function approximators (Precup et al. 2001; Maei et al. 2010). Recently, Langford et al. (2008) discuss how policies can be evaluated using off-line experience.

Preference-based reinforcement learning is also related to *inverse reinforcement learning* (Ng and Russell 2000; Abbeel and Ng 2010). In this framework,

the idea is to learn a reward function from traces of a presumably optimal policy so that the reward is consistent with the observed policy traces. Preference-based reinforcement learning follows a similar goal, but weakens some of the assumptions: first, we do not assume that the learner has access to observations of an optimal policy but instead only require comparisons between several, possibly suboptimal actions or trajectories. Second, the objective is not necessarily to learn a reward signal, but it suffices to learn a policy that is consistent with the observed trajectories. In that respect, the work is also related to early work in the area of *behavioral cloning* (Sammut 1996). Similarly, learning player behavior from game traces is an important topic in computer game research (Fürnkranz 2011).

8.6 Handling Qualitative Information

The exploitation of qualitative information in RL or, more generally, in MDPs, has received less attention so far. Bonet and Pearl (2002) propose a qualitative version of MDPs and POMDPs (Partially Observable MDPs) based on a so-called order-of-magnitude representation of transition probabilities and rewards. This approach is closely related to the work of Sabbadin (1999), who models uncertainty in terms of possibility instead of probability distributions. Epshteyn and DeJong (2006) present a framework which allows the expert to specify imprecise knowledge of transition probabilities in terms of stochastic dominance constraints.

Reyes et al. (2006) propose a method for learning qualitative MDPs. They argue that, in complex domains, it is not always possible to provide a reasonable state representation and transition model. Their approach is able to automatically produce a state abstraction and can learn a transition function over such abstracted states. A qualitative state, called q-state, is a group of states with similar properties and rewards.

There are also connections to some other fields of AI research dealing with qualitative knowledge. One such field is qualitative reasoning (Kuipers 1994), where the learning of qualitative models has also been considered (Bratko and Suc 2003; Zabkar et al. 2008), even though the focus here is less on control and more on the modeling and simulation of dynamical systems. Another field is qualitative decision making, in which qualitative variants of classical Bayesian decision theory and expected utility theory have been developed (Brafman and Tennenholtz 1997; Doyle and Thomason 1999; Dubois et al. 2003; Fargier and Sabbadin 2005).

9 Current and Future Work

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 which can be used

for performing roll-out simulations 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. One of our goals is to develop a preference-based version of Q-learning. The key missing link to achieve this goal is to find a preference-based equivalent to the Bellman equation, which allows to transfer information about action preferences from one state to the other.

A first step in that direction is to define and evaluate a ranking-based exploration strategy. The results on partial state evaluations (Section 6.5) indicate that an exploration strategy that is based on picking the better one of a pair of actions may be an interesting approach to try. It seems clear that information provided by a ranking gives more information than uninformed exploration strategies like ϵ-greedy strategies, and we believe that the loss of information that we suffer from only having a ranking instead of expected utilities or action probabilities is only minor. This, however, needs to be properly evaluated.

We are currently working on applying preference-based reinforcement learning to the chess domain that we have used to motivate our framework in Section 2.3. The main obstacle that we have to solve here is that preference-based policy iteration as proposed in this paper is for on-line learning, whereas the preferences that we want to learn from are only available off-line. An interesting way for solving this strategy could be an integration of off-line advice with on-line experience. This can be easily done in Algorithm 3 by merging preferences from off-line data with preferences that have been generated via roll-out analysis with the current policy. A problem here is that roll-out analysis with imperfect policies tends to be considerably less reliable for chess than for games like Go (Ramunajan et al. 2010; Arenz 2012), which is one of the motivations for resorting to game annotations in this domain. We are also aiming at lifting inverse reinforcement learning (Ng and Russell 2000; Abbeel and Ng 2010) to a preference-based formulation.

10 Conclusions

The main contribution of this work is a framework for preference-based reinforcement learning, which allows for lifting RL into a qualitative setting, where reward is not available on an absolute, numerical scale. Instead, comparative reward functions can be used to decide which of two actions is preferable in a given state, or, more generally, which of two trajectories is preferable.

To cope with this type of training information, we proposed an algorithm for preference-based policy iteration, which only depends on the availability of preference information between two actions. As a proof-of-concept, we instantiated this algorithm into a version of approximate policy iteration, where the preference information is determined via roll-outs. Whereas the original algorithm 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.

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

Abbeel, P. and Ng, A. Y. (2010). Inverse reinforcement learning. In Sammut, C. and Webb, G. I., editors, *Encyclopedia of Machine Learning*, pages 554–558. Springer-Verlag.

Akrour, R., Schoenauer, M., and Sebag, M. (2011). Preference-based policy learning. In Gunopulos, D., Hofmann, T., Malerba, D., and Vazirgiannis, M., editors, *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-11), Part I*, pages 12–27, Athens, Greece. Springer.

Amari, S. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276.

Arenz, O. (2012). *Monte-Carlo Chess.* Bachelor's Thesis, Knowledge Engineering Group, TU Darmstadt.

Audibert, J.-Y., Bubeck, S., and Munos, R. (2010). Best arm identification in multi-armed bandits. In Kalai, A. T. and Mohri, M., editors, *Proceedings of the 23rd Conference on Learning Theory (COLT-10)*, pages 41–53, Haifa, Israel. Omnipress.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (1995). Gambling in a rigged casino: The adversarial multi-arm bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322–331. IEEE Computer Society Press.

Barto, A. G., Sutton, R. S., and Anderson, C. (1983). Neuron-like elements that can solve difficult learning control problems. *IEEE Transaction on Systems, Man and Cybernetics*, 13:835–846.

Baxter, J., Tridgell, A., and Weaver, L. (2000). Learning to play chess using temporal differences. *Machine Learning*, 40(3):243–263.

Beal, D. F. and Smith, M. C. (2001). Temporal difference learning applied to game playing and the results of application to Shogi. *Theoretical Computer Science*, 252(1-2):105–119. Special Issue on Papers from the Computers and Games 1998 Conference.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., and Lee, M. (2009). Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482.

Bonet, B. and Pearl, J. (2002). Qualitative mdps and pomdps: An order-of-magnitude approximation. In *Proc. 18th Conference in Uncertainty in Artificial Intelligence (UAI-02)*, pages 61–68, Alberta, Canada.

Brafman, R. I. and Tennenholtz, M. (1997). Modeling agents as qualitative decision makers. *Artificial Intelligence*, 94(1-2):217–268.

Bratko, I. and Suc, D. (2003). Learning qualitative models. *AI Magazine*, 24(4):107–119.

Cheng, W., Fürnkranz, J., Hüllermeier, E., and Park, S.-H. (2011). Preference-based policy iteration: Leveraging preference learning for reinforcement learning. In Gunopulos,

D., Hofmann, T., Malerba, D., and Vazirgiannis, M., editors, *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-11), Part I*, pages 312–327, Athens, Greece. Springer.

Crites, R. and Barto, A. (1998). Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33:235–262.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

Dimitrakakis, C. and Lagoudakis, M. G. (2008). Rollout sampling approximate policy iteration. *Machine Learning*, 72(3):157–171.

Doyle, J. and Thomason, R. (1999). Background to qualitative decision theory. *AI Magazine*, 20(2):55–68.

Driessens, K. and Džeroski, S. (2004). Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304.

Droste, S. and Fürnkranz, J. (2008). Learning the piece values for three chess variants. *International Computer Games Association Journal*, 31(4):209–233.

Dubois, D., Fargier, H., and Perny, P. (2003). Qualitative decision theory with preference relations and comparative uncertainty: An axiomatic approach. *Artificial Intelligence*, 148(1–2):219–260.

Džeroski, S., De Raedt, L., and Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, 43(1–2):7–52.

Edwards, S. J. (1994). Portable game notation. `http://folk.uio.no/andreio/docs/pgnspec.pdf`.

Epshteyn, A. and DeJong, G. (2006). Qualitative reinforcement learning. In *Proc. 23rd International Conference on Machine Learning (ICML-06)*, pages 305–312, Pittsburgh, Pennsylvania, USA.

Even-Dar, E., Mannor, S., and Mansour, Y. (2003). Action elimination and stopping conditions for reinforcement learning. In Fawcett, T. and Mishra, N., editors, *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 162–169, Washington, DC. AAAI Press.

Fargier, H. and Sabbadin, R. (2005). Qualitative decision under uncertainty: back to expected utility. *Artificial Intelligence*, 164(1–2):245–280.

Fern, A., Yoon, S. W., and Givan, R. (2006). Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research*, 25:75–118.

Fürnkranz, J. (2011). Machine learning and game playing. In Sammut, C. and Webb, G. I., editors, *Encyclopedia of Machine Learning*, pages 633–637. Springer-Verlag.

Fürnkranz, J. and Hüllermeier, E., editors (2010). *Preference Learning*. Springer-Verlag.

Fürnkranz, J., Pfahringer, B., Kaindl, H., and Kramer, S. (2000). Learning to use operational advice. In Horn, W., editor, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-00)*, pages 291–295, Berlin. IOS Press.

Gabillon, V., Lazaric, A., and Ghavamzadeh, M. (2010). Rollout allocation strategies for classification-based policy iteration. In Auer, P., Kaski, S., and Szepesvàri, C., editors, *Proceedings of the ICML-10 Workshop on Reinforcement Learning and Search in Very Large Spaces*.

Gabillon, V., Lazaric, A., Ghavamzadeh, M., and Scherrer, B. (2011). Classification-based policy iteration with a critic. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1049–1056, New York, NY, USA. ACM.

Gábor, Z., Kalmár, Z., and Szepesvàri, C. (1998). Multi-criteria reinforcement learning. In *Proceedings of the 15th International Conference on Machine Learning (ICML-98)*, pages 197–205, Madison, WI. Morgan Kaufmann.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD explorations*, 11(1):10–18.

Heidrich-Meisner, V. and Igel, C. (2009). Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In Danyluk, A. P., Bottou, L., and Littman, M. L., editors, *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, ACM International Conference Proceeding Series, pages 401–408, Montreal, Canada.

Hüllermeier, E., Fürnkranz, J., Cheng, W., and Brinker, K. (2008). Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172:1897–1916.

Kakade, S. (2001). A natural policy gradient. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14 (NIPS-2001)*, pages 1531–1538, Vancouver, British Columbia, Canada. MIT Press.

Kamishima, T., Kazawa, H., and Akaho, S. (2010). A survey and empirical comparison of object ranking methods. In (Fürnkranz and Hüllermeier 2010), pages 181–201.

Kersting, K. and Driessens, K. (2008). Non-parametric policy gradients: a unified treatment of propositional and relational domains. In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, volume 307 of *ACM International Conference Proceeding Series*, pages 456–463, Helsinki, Finland. ACM.

Konda, V. R. and Tsitsiklis, J. N. (2003). On actor-critic algorithms. *SIAM Journal of Control and Optimization*, 42(4):1143–1166.

Kuipers, B. (1994). *Qualitative Reasoning*. MIT Press.

Lagoudakis, M. G. and Parr, R. (2003). Reinforcement learning as classification: Leveraging modern classifiers. In Fawcett, T. E. and Mishra, N., editors, *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 424–431, Washington, DC, USA. AAAI Press.

Langford, J., Strehl, A. L., and Wortman, J. (2008). Exploration scavenging. In Cohen, W. W., McCallum, A., and Roweis, S. T., editors, *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, volume 307 of *ACM International Conference Proceeding Series*, pages 528–535, Helsinki, Finland. ACM.

Langford, J. and Zhang, T. (2008). The epoch-greedy algorithm for multi-armed bandits with side information. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems 20 (NIPS-21)*, Vancouver, Canada. MIT Press.

Lazaric, A., Ghavamzadeh, M., and Munos, R. (2010). Analysis of a classification-based policy iteration algorithm. In Fürnkranz, J. and Joachims, T., editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 607–614. Omnipress.

Li, L., Chu, W., Langford, J., and Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. In Rappa, M., Jones, P., Freire, J., and Chakrabarti, S., editors, *Proceedings of the 19th International Conference on World Wide Web (WWW-10)*, pages 661–670, Raleigh, North Carolina. ACM.

Maes, F. (2009). Learning in Markov Decision Processes for Structured Prediction. PhD thesis, University Pierre et Marie Curie, Paris, France.

Loza Mencía, E., Park, S.-H., and Fürnkranz, J. (2010). Efficient voting prediction for pairwise multilabel classification. *Neurocomputing*, 73(7-9):1164–1176.

Maclin, R. and Shavlik, J. W. (1996). Creating advice-taking reinforcement learners. *Machine Learning*, 22(1-3):251–281.

Maclin, R., Shavlik, J. W., Torrey, L., Walker, T., and Wild, E. W. (2005). Giving advice about preferred actions to reinforcement learners via knowledge-based kernel regression. In Veloso, M. M. and Kambhampati, S., editors, *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI-05)*, pages 819–824, Pittsburgh, Pennsylvania. AAAI Press / The MIT Press.

Maei, H. R., Szepesvàri, C., Bhatnagar, S., and Sutton, R. S. (2010). Toward off-policy learning control with function approximation. In Fürnkranz, J. and Joachims, T., editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 719–726, Haifa, Israel. Omnipress.

Mannor, S. and Shimkin, N. (2004). A geometric approach to multi-criterion reinforcement learning. *Journal of Machine Learning Research*, 5:325–360.

Massey, W. A. (1987). Stochastic orderings for Markov processes on partially ordered spaces. *Mathematics of Operations Research*, 12(2):350–367.

Ng, A. Y. and Jordan, M. I. (2000). Pegasus: A policy search method for large mdps and pomdps. In Boutilier, C. and Goldszmidt, M., editors, *Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence (UAI-00)*, pages 406–415, Stanford University, Stanford, California. Morgan Kaufmann.

Ng, A. Y. and Russell, S. (2000). Algorithms for inverse reinforcement learning. In Langley, P., editor, *Proceedings of the 17th International Conference on Machine Learning (ICML-00)*, pages 663–670, Stanford, CA. Morgan Kaufmann.

Park, S.-H. and Fürnkranz, J. (2012). Efficient prediction algorithms for binary decomposition techniques. *D*ata Mining and Knowledge Discovery, 24(1): 40–77.

Peters, J. and Schaal, S. (2008a). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

Peters, J. and Schaal, S. (2008b). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682–697.

Precup, D., Sutton, R. S., and Dasgupta, S. (2001). Off-policy temporal difference learning with function approximation. In Brodley, C. E. and Danyluk, A. P., editors, *Proceedings of the 18th International Conference on Machine Learning (ICML-01)*, pages 417–424, Williams College, Williamstown, MA. Morgan Kaufmann.

Puterman, M. L. (2005). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 2nd edition.

Ramanujan, R., Sabharwal, A., and Selman B. (2010). On adversarial search spaces and sampling-based planning.
In R.I. Brafman, H. Geffner, J. Hoffmann, and H.A. Kautz, editors *Proceedings of 20th International Conference on Automated Planning and Scheduling (ICAPS-10)*, pages 242–245, Toronto, Ontario, Canada.

Reyes, A., Ibarguengoytia, P., Sucar, L., and Morales, E. (2006). Abstraction and refinement for solving continuous markov decision processes. In *Proc. 3rd European Workshop on Probabilistic Graphical Models*, pages 263–270, Prague, Czech Republic.

Sabbadin, R. (1999). A possibilistic model for qualitative sequential decision problems under uncertainty in partially observable environments. In *Proc. UAI, 15th Conference on Uncertainty in Artificial Intelligence*, pages 567–574, Stockholm, Sweden.

Sammut, C. (1996). Automatic construction of reactive control systems using symbolic machine learning. *Knowledge Engineering Review*, 11(1):27–42.

Srinivas, N. and Deb, K. (1995) Multiobjective optimization using nondominant sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248.

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

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In Solla, S. A., Leen, T. K., and Müller, K.-R., editors, *Advances in Neural Information Processing Systems 12 (NIPS-99)*, pages 1057–1063, Denver, Colorado, USA. The MIT Press.

Tadepalli, P., Givan, R., and Driessens, K. (2004). Relational reinforcement learning: an overview. In Tadepalli, P., Givan, R., and Driessens, K., editors, *Proceedings of the ICML'04 Workshop on Relational Reinforcement Learning*, pages 1–9.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 8:257–278.

Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1-2):181–199. Special Issue on Games, Computers and Artificial Intelligence.

Torrey, L., Walker, T., Shavlik, J. W., and Maclin, R. (2005). Using advice to transfer knowledge acquired in one reinforcement learning task to another. In Gama, J., Camacho, R., Brazdil, P., Jorge, A., and Torgo, L., editors, *Proceedings of the 16th European Conference on Machine Learning (ECML-05)*, pages 412–424, Porto, Portugal. Springer.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., and Dekker, E. (2010). Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine Learning*, 84(1-2):51–80.

Vembu, S. and Gärtner, T. (2010). Label ranking algorithms: A survey. In (Fürnkranz and Hüllermeier 2010), pages 45–64.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.

Zabkar, J., Bratko, I., and Mohan, A. (2008). Learning qualitative models by an autonomous robot. In *Proc. 22nd International Workshop on Qualitative Reasoning*, Boulder, Colorado.

Zhao, Y., Kosorok, M., and Zeng, D. (2009). Reinforcement learning design for cancer clinical trials. *Statistics in Medicine*, 28:3295–3315.