Journal of Machine Learning Research 18 (2017) 1-46 Submitted 12/16; Revised 11/17; Published 12/17

A Survey of Preference-Based Reinforcement Learning Methods

Christian Wirth CWIRTH@KE.TU-DARMSTADT.DE
Knowledge Engineering Group, Technische Universität Darmstadt
Hochschulstraße 10, 64289 Darmstadt, Germany

Riad Akrour RIAD@ROBOT-LEARNING.DE
Computational Learning for Autonomous Systems, Technische Universität Darmstadt
Hochschulstraße 10, 64289 Darmstadt, Germany

Gerhard Neumann GNEUMANN@LINCOLN.AC.UK
Computational Learning, School of Computer Science, University of Lincoln
Brayford Pool, Lincoln, LN6 7TS, Great Britain

Johannes Fürnkranz JUFFI@KE.TU-DARMSTADT.DE
Knowledge Engineering Group, Technische Universität Darmstadt
Hochschulstraße 10, 64289 Darmstadt, Germany

Editor: George Konidaris

Abstract

Reinforcement learning (RL) techniques optimize the accumulated long-term reward of a suitably chosen reward function. However, designing such a reward function often requires a lot of task-specific prior knowledge. The designer needs to consider different objectives that do not only influence the learned behavior but also the learning progress. To alleviate these issues, preference-based reinforcement learning algorithms (PbRL) have been proposed that can directly learn from an expert's preferences instead of a hand-designed numeric reward. PbRL has gained traction in recent years due to its ability to resolve the reward shaping problem, its ability to learn from non numeric rewards and the possibility to reduce the dependence on expert knowledge. We provide a unified framework for PbRL that describes the task formally and points out the different design principles that affect the evaluation task for the human as well as the computational complexity. The design principles include the type of feedback that is assumed, the representation that is learned to capture the preferences, the optimization problem that has to be solved as well as how the exploration/exploitation problem is tackled. Furthermore, we point out shortcomings of current algorithms, propose open research questions and briefly survey practical tasks that have been solved using PbRL.

Keywords: Reinforcement Learning, Preference Learning, Qualitative Feedback, Markov Decision Process, Policy Search, Temporal Difference Learning, Preference-based Reinforcement Learning

1. Introduction

Due to its solid theoretical foundations, *reinforcement learning* (RL) has become a very attractive tool that has seen many successful applications. It achieved a world first success of machine learning, when TD-Gammon learned to play backgammon (Tesauro, 1995), and

©2017 Christian Wirth, Riad Akrour, Gerhard Neumann and Johannes Fürnkranz.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v18/16-634.html.

its applications which used to focus on games and robotics (Mnih et al., 2015; Kober et al., 2013) now extend to diverse problems in the industrial and medical domain (Wang and Usher, 2005; Zhao et al., 2011). However, the success of reinforcement learning applications often crucially depends on the prior knowledge that is put into the definition of the reward function. Although a reward function is a transferable and compact definition of the learning task, the learned policy may be sensitive to small changes of the reward, possibly yielding very different behaviors, depending on the relative values of the rewards. Thus, the choice of the reward function may have a crucial impact on the success of the learner and many tasks, in particular in robotics, require a high amount of reward-engineering. Four problems are mainly responsible for the difficulties in reward engineering.

(1) Reward Hacking: The agent may maximize the given reward, without performing the intended task (Amodei et al., 2016). For example, consider a robotic vacuum cleaner obtaining a positive reward in case all dirt has been removed. As the robot only observes dirt via camera, it may try to cover up the dirt, effectively removing it from its sensor space

(2) Reward Shaping: In many applications, the reward does not only define the goal but also guides the agent to the correct solution, which is also known as reward shaping (Ng et al., 1999). Striking a balance between the goal definition and the guidance task is also often very difficult for the experimenter.

(3) Infinite rewards: Some applications require infinite rewards, for instance, if certain events should be avoided at any cost. For example, in the cancer treatment problem (Zhao et al., 2009), the death of a patient should be avoided at any cost. However, an infinitely negative reward breaks classic reinforcement learning algorithms and arbitrary, finite values have to be selected.

(4) Multi-objective trade-offs: If it is required to aggregate multiple numeric reward signals, a trade-off is required but the trade-off may not be explicitly known. Hence, different parameterizations have to be evaluated for obtaining a policy that correlates with the expected outcome.

The *reward shaping* problem has especially gained attention in recent years, as it is closely related to intrinsic vs. extrinsic motivation problem (Barto, 2013). An optimal reward would define an extrinsic motivation, e.g., for achieving a defined goal, while also motivating the agent to perform useful actions in states where extrinsic motivation is absent. Hence, the intrinsic motivation can be seen as a guidance method. Several methods have been proposed to learn such a reward automatically, but usually only using an extrinsic goal description (Singh et al., 2009, 2004). However, expert's may have an intuition about useful actions or intrinsic motivation that should be used to guide the agent. Defining such a motivation can be challenging, especially in terms of scalar, numeric rewards and several publications show that an agent's performance can be very sensitive to the used values (Singh et al., 2009; Lu et al., 2016).

Preference-based reinforcement learning (PbRL) is a paradigm for learning from non-numerical feedback in sequential domains. Its key idea is that the requirement for a numerical feedback signal is replaced with the assumption of a preference-based feedback signal

that indicates relative instead of absolute utility values. Preferences enable a definition of feedback that is not subject to arbitrary reward choices, reward shaping, reward engineering or predefined objective trade-offs. PbRL aims at rendering reinforcement learning applicable to a wider spectrum of tasks and non-expert users. Several algorithms have been developed for solving this class of problems, however, the field still lacks a coherent framework. Furthermore, most publications do not explicitly state the assumptions that are relevant for the application of the algorithms. In this paper, we will survey and categorize preference-based formulations of reinforcement learning and make these assumptions explicit. We also show the relation to other RL-based settings that deviate from the basic setting, such as inverse reinforcement learning or learning with advice.

This paper is based on a preliminary survey (Wirth and Fürnkranz, 2013c), but has been extended in almost every aspect. In particular, the design principles have been reworked to better capture all relevant aspects, we provide substantially more depth about the differences, advantages and disadvantages of the surveyed methods, and we considerably broadened the scope of the survey. The paper starts with the problem setting and the task description in Sec. 2, including a short discussion of related approaches. We then explain the design principles of the algorithms that fit in this framework in Sec. 3. Experimental domains, including some real-world problems that have been solved with PbRL, are described in Sec. 4. Open research questions are pointed out in Sec. 5 before the paper concludes.

2. Preliminaries

Preference-based reinforcement learning algorithms try to solve the *reinforcement learning* problem (Sutton and Barto, 1998) using preferences between states, actions or trajectories.[1] The goal is to learn a policy that is most consistent with the preferences of the expert. Hence, a PbRL algorithm must determine the intention of the expert and find a policy that is maximally consistent with it. For improving its estimate, the algorithms create trajectories which are evaluated by the expert, thereby, exploring the state-action space. Therefore, many algorithms use the *policy evaluation/policy improvement* cycle (Sutton and Barto, 1998) encountered in classical reinforcement learning. Yet, this cycle differs in the evaluation method from classical RL as we can't obtain and propagate numeric feedback, but can only use pairwise preferences.

2.1 Preference Learning

Preference learning is a recent addition to the suite of learning tasks in machine learning (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. The key difference to conventional supervised machine learning settings is that the training information is typically not given in the form of scalar target values (like in classification and regression), but instead in the form of pairwise comparisons expressing *preferences* between different objects or labels. Thus, in a way, preference learning is comparable to reinforcement learning in that in both settings, the learner may not receive information about the

1. State and action preferences can be mapped to trajectory preferences, as we discuss in Sec. 3.1

correct target choice, but instead feedback about a single option (in reinforcement learning) or a comparison between two options (in preference learning).

A preference $\boldsymbol{z}_i \succ \boldsymbol{z}_j$ denotes that $\boldsymbol{z}_i$ is preferred over $\boldsymbol{z}_j$ for two choices $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ in a set of possible choices $\mathcal{Z}$. Several short-hand notations can be used (Kreps, 1988):

– $\boldsymbol{z}_i \succ \boldsymbol{z}_j$**:** The first choice is *strictly preferred*.

– $\boldsymbol{z}_i \prec \boldsymbol{z}_j$**:** The second choice is *strictly preferred*, i.e., $\boldsymbol{z}_j \succ \boldsymbol{z}_i$.

– $\boldsymbol{z}_i \sim \boldsymbol{z}_j$**:** The choices are *indifferent*, meaning neither $\boldsymbol{z}_i \succ \boldsymbol{z}_j$ nor $\boldsymbol{z}_j \succ \boldsymbol{z}_i$ holds.

– $\boldsymbol{z}_i \succeq \boldsymbol{z}_j$**:** The first choice is *weakly preferred*, i.e., $\boldsymbol{z}_j \succ \boldsymbol{z}_i$ does not hold.

– $\boldsymbol{z}_i \preceq \boldsymbol{z}_j$**:** The second choice is *weakly preferred*, i.e., $\boldsymbol{z}_i \succ \boldsymbol{z}_j$ does not hold.

Depending on what constitutes the set of choices $\mathcal{Z}$, Fürnkranz and Hüllermeier (2010) distinguish between *learning from object preferences*, where the training data is given in the form of pairwise comparisons between data objects, and *learning from label preferences*, where the training data is given in the form of pairwise comparisons between labels that are attached to the objects. In the former case, a common performance task is to rank a new set of objects (*object ranking*; Kamishima et al. 2010), whereas in the latter case, the performance task is to rank the set of labels for a new object (*label ranking*; Vembu and Gärtner 2010). For example, in a reinforcement learning context, states may be considered as objects, and actions may be considered as labels, so that the decision problem of ranking the states according to their desirability is an object ranking problem, whereas ranking the available actions in a given state may be considered as a label ranking problem (Fürnkranz et al., 2012).

There are two main approaches to learning representations of preferences, namely *utility functions*, which evaluate individual alternatives, and *preference relations*, which compare pairs of competing alternatives. From a machine learning point of view, the two approaches give rise to two kinds of learning problems, to which we will return in Sec. 3.2.

2.2 Markov Decision Processes with Preferences

In this section, we will introduce *Markov decision processes with preferences (MDPP)*, a formal, mathematical framework for preference-based reinforcement learning. It is based upon Markov decision processes without reward (MDP\R), which have been proposed by Abbeel and Ng (2004), but extends the setting to preference based feedback. An MDPP is defined by a sextuple $(S, A, \mu, \delta, \gamma, \rho)$ with a *state space* S and *action space* A. The states and actions can be discrete or continuous. $A(s)$ defines the set of actions available in state s and $\mu(s)$ is the distribution of possible initial states. The states can also be represented by feature vectors $\boldsymbol{\phi}(s)$ and state action pairs by the feature vector $\boldsymbol{\varphi}(s, a)$. We will denote a trajectory $\boldsymbol{\tau}$ as a sequence of states and actions, i.e.,

$$\boldsymbol{\tau} = \{s_0, a_0, s_1, a_1, \ldots, s_{n-1}, a_{n-1}, s_n\}\,.$$

Some algorithms (Wirth et al., 2016; Wilson et al., 2012; Akrour et al., 2011, 2012; Jain et al., 2013, 2015; Gritsenko and Berenson, 2014; Zucker et al., 2010; Jain et al., 2013, 2015;

Gritsenko and Berenson, 2014) also require the definition of a feature-based description of a trajectory, which we denote as $\boldsymbol{\psi}(\boldsymbol{\tau})$.

As in the standard MDP definition, the *state transition model* $\delta(s' \mid s, a)$ is assumed to be stochastic and the parameter $\gamma \in [0, 1)$ is the discount factor. A *policy* $\pi(a \mid s)$ is a conditional distribution that assigns probabilities to action choices based on the current state. The policy is typically given by a parametric distribution $\pi(a \mid s, \boldsymbol{\theta})$, however, non-parametric approaches are also considered (Akrour et al., 2012; Wirth et al., 2016; Busa-Fekete et al., 2013, 2014).

In contrast to conventional MDPs, we do not assume a numeric reward signal $r(s, a)$. Instead, the agent can observe a preference relation over trajectories $\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j$. Later, in Sec. 3.1, we will also consider preferences between states and actions, and show that they can be reduced to preferences between trajectories. We further assume that a preference for a given pair of trajectories is generated stochastically with a probability distribution ρ because the expert can err and therefore introduce noise. We use $\rho(\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j)$ for denoting the probability with which $\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j$ holds for a given pair of trajectories $(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j)$. The distribution ρ is typically not known, yet the agent can observe a set of preferences

$$\zeta = \{\zeta_i\} = \{\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2}\}_{i=1\ldots N}\,, \tag{1}$$

which has been sampled using ρ. $\boldsymbol{\Upsilon}$ denotes the set of all trajectories that are part of ζ. A key problem in PbRL is to obtain a representative set of preferences ζ. Two common simplifications are to disregard the stochasticity of the expert and assume strict preferences. The strict preference assumption implies a total order, i.e., for each pair $\boldsymbol{\tau}_i$ and $\boldsymbol{\tau}_j$, exactly one of the two strict preference relations holds. Phrased otherwise, $\rho(\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j) = 1 - \rho(\boldsymbol{\tau}_j \succ \boldsymbol{\tau}_i)$. As a consequence, there are no incomparable pairs of trajectories. Incomparability can occur when evaluating trajectories based on multiple criteria, where it is impossible to improve one criteria while decreasing another one. If there are incomparable pairs where no preference relation can be defined, the preferences form a partial order.

2.3 Objective

The general objective of the agent is to find a policy π^* that maximally complies with the given set of preferences ζ. A preference $\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2 \in \zeta$ is satisfied if

$$\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2 \Leftrightarrow \Pr_\pi(\boldsymbol{\tau}_1) > \Pr_\pi(\boldsymbol{\tau}_2),$$

with

$$\Pr_\pi(\boldsymbol{\tau}) = \mu(s_0) \prod_{t=0}^{|\boldsymbol{\tau}|} \pi(a_t \mid s_t)\delta(s_{t+1} \mid s_t, a_t),$$

as the probability of realizing a trajectory $\boldsymbol{\tau}$ with a policy π. However, satisfying this condition is typically not sufficient as the difference can be infinitesimal. Moreover, due to the stochasticity of the expert, the observed preferences can also contradict each other.

We can now reformulate the objective for a single preference as a maximization problem concerning the difference of the probabilities of trajectories $\boldsymbol{\tau}_1$ and $\boldsymbol{\tau}_2$, e.g.,

$$\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2 \Leftrightarrow \pi^* = \arg\max_\pi \left(\Pr_\pi(\boldsymbol{\tau}_1) - \Pr_\pi(\boldsymbol{\tau}_2)\right), \tag{2}$$

which is equivalent to creating, if feasible, only dominating trajectories. However, this definition disregards the question of how to deal with multiple preferences.

In general, the goal of all algorithms is to minimize a single preference loss $L(\pi, \zeta_i)$, e.g., $L(\pi, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) = -(\Pr_\pi(\boldsymbol{\tau}_1) - \Pr_\pi(\boldsymbol{\tau}_2))$. The domain of L differs, depending on the approach. Yet, in the multi-preference case, minimizing the loss for one preference might interfere with minimizing the loss for another preference. Hence, without further information, we are not able to define a single loss function but can only specify a multi-objective criterion for the optimal policy where each preference defines a loss function

$$\boldsymbol{L}(\pi, \zeta) = (L(\pi, \zeta_0), L(\pi, \zeta_1), \ldots, L(\pi, \zeta_n))$$

Optimizing this multi-objective loss results in several Pareto-optimal solutions. However, we typically want to obtain a single optimal solution. To do so, we can define a scalar objective as the *weighted pairwise disagreement loss* $\mathcal{L}$, as introduced by Duchi et al. (2010),

$$\mathcal{L}(\pi, \zeta) = \sum_{i=1}^{N} \alpha_i L(\pi, \zeta_i), \tag{3}$$

where α_i is the weight or importance assigned to ζ_i. Using such a weighting is a common strategy in preference learning where different evaluation criteria, like the Kendall or the Spearmen distance (Kamishima et al., 2010), are available for different domains. We introduce the weighting α_i to illustrate all preference evaluation criteria in a unified framework.

In PbRL, we want to find a policy that maximizes the realization probability of undominated trajectories while not generating dominated trajectories, if possible. Therefore, preferences concerning the least dominated trajectories should have the highest weight, because the according preferences are most important. Although in other settings, it might be more relevant to realize preferences involving high risk trajectories for preventing major accidents. Hence, different weights α_i can be required. However, all available approaches disregard this problem and use a uniform weight distribution, but focus on requesting preferences over (nearly) undominated trajectories, which has a similar effect as a weighting. In Sec. 3, we explain the different definitions of the single preference loss $L(\pi, \zeta_i)$.

2.4 Preference-Based Reinforcement Learning Algorithms

Learning from preferences is a process that involves two actors: an *agent*, that acts according to a given policy and an *expert* evaluating its behavior. The process is typically composed of several components, as illustrated in Fig. 1. The learning usually starts with a set of trajectories, either predefined or sampled from a given policy. An expert evaluates one or more trajectory pairs $(\boldsymbol{\tau}_{i1}, \boldsymbol{\tau}_{i2})$ and indicates her preference. The method for indicating a preference can differ (cf. Sec. 3.1). For computing a preference-based policy π, three different learning approaches can be found in the literature:

- *learning a policy* computes a policy that tries to maximally comply with the preferences;
- *learning a preference model* learns a model for approximating the expert's preference relation, and

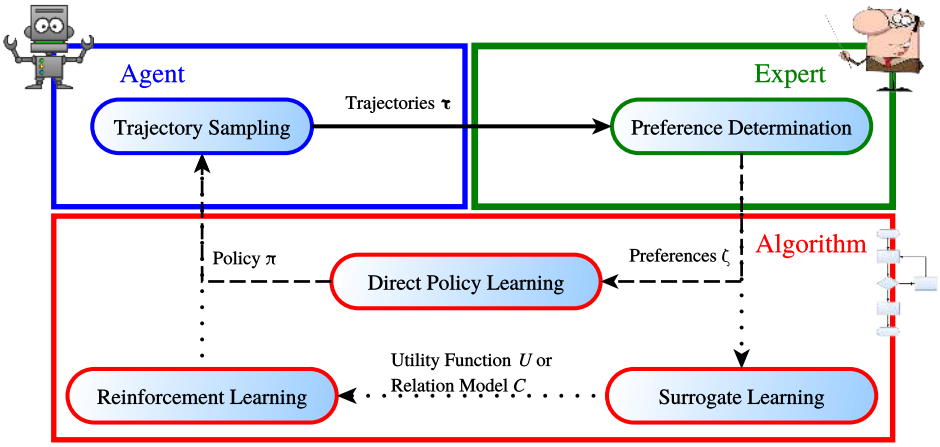


Figure 1: PbRL: Learning policies from preferences via direct (dashed path) and surrogate-based (dotted path) approaches.

- *learning a utility function* estimates a numeric function for the expert's evaluation criterion.

The dashed path in Fig. 1 relates to policy learning. Both other approaches, learning either a preference model or a utility function, are *surrogate-based* in that they learn a surrogate from the obtained preferences, which is in turn used for deriving a maximizing policy (cf. the dotted path in Fig. 1). For this optimization, often conventional reward-based *policy improvement* (PI) algorithms can be used. All three cases are explained in more detail in Sec. 3.2, but the result is always a new policy π_{new}, that can be used to sample new trajectories.

Typically, new trajectories are evaluated repeatedly, and hence, learning from preferences becomes an interactive process between the expert and the algorithm, where the algorithm presents new trajectory pairs to the expert and the expert evaluates them. However, in the interactive setting, a criterion has to be defined for the selection of the preference queries that should be presented to the expert, as described in Sec. 3.4. This criterion must also resolve the exploration/exploitation trade-off as we can only obtain feedback for explicitly requested preferences.

A single pass through the PbRL cycle is often performed in a batch setting, where available trajectories and preference data are used to compute a policy offline.

2.5 Related Problem Settings

Preference-based reinforcement learning is closely related to several other learning settings, which we will briefly discuss in this section.

2.5.1 Learning with Advice

Advice-based algorithms (Torrey and Taylor, 2013; Knox and Stone, 2010, 2012; Griffith et al., 2013) differ from PbRL in that the (possibly preference-based) advice is given in addition to a numeric reward signal. Hence, the main source of feedback is still numeric, not preference-based, and preferences are only used as additional constraints for speeding up the learning process. In addition, not all advice-taking algorithms use preferences. The advice can also be defined via other means, such as rules (Maclin et al., 2005).

Faust et al. (2016) assume that advice is only given in terms of attracting and repulsing states, without an additional reward signal. Such advice is similar to state preferences, but the states are demonstrated by the expert and not evaluated in a pairwise manner.

2.5.2 Ordinal Feedback

PbRL methods can also be used to solve related problem settings. For example, domains with *ordinal feedback signals* can be cast as preference problems by deriving pairwise preferences from the ranking of the ordinal symbols. However, conversely, trying to determine a grading for the ordinal symbols (Weng, 2011) is not feasible if only pairwise preferences are available. Daniel et al. (2014) assume a numeric rating of trajectories. A numeric rating also implicitly defines a ranking which can be again reduced to preferences. While a ranking implicitly contains more information than a binary preference it is not clear whether this additional information can also be transmitted reliably by the expert. Kupcsik et al. (2015) introduced an approach that can use preference as well as numeric feedback. The presented experiments indicate that mixing preference feedback and numeric feedback queries lead to a better performance. However, more experiments with human subjects are needed to shed light on the question of how to combine these feedback types.

2.5.3 Inverse Reinforcement learning

In *inverse reinforcement learning* (IRL; Ng and Russell 2000; Zhifei and Meng Joo 2012), the goal is not to find a policy, but the reward signal that explains the expert's behavior. Stated differently, IRL assumes that the expert maximizes an internal reward function and the goal is to identify this function. However, the expert may use an unknown, true reward which cannot be reconstructed as it may not be subject to the Markov property. Although, it is usually sufficient to find a reward function that produces the same output in each state.

In this setting, the trajectories are supplied by the expert, not by the algorithm. There are also no explicit pairwise preferences, yet they can be derived based on implicit assumptions. In the IRL setting, the demonstrated sequences can be assumed to be preferred over all other trajectories. Hence, the reward-based utility function learning approaches may also be used to solve IRL problems. Many IRL approaches are based on preferences that are defined by implicit assumptions and can be applied to reward-based utility PbRL by adapting them to explicit preferences (see Sec. 3.2.3). Such implicit preferences are also used by Knox and Stone (2009), however, the approach is based on positive (preferred) and negative (dominated) feedback. All positively evaluated sequences are implicitly preferred over all negatively evaluated ones. While most IRL settings assume optimal demonstrations, a few methods (Rothkopf and Dimitrakakis, 2011; Ziebart et al., 2008) relax this assumption and allow sub-optimal demonstrations, based on a parametric optimality prior like a softmax

function. Furthermore, as all trajectories are given by an expert, IRL approaches are not able to obtain new feedback for trajectories and, hence, can not improve upon the expert demonstrations.

2.5.4 Unsupervised Learning

A different line of work (Meunier et al., 2012; Mayeur et al., 2014) uses preferences for learning without a supervised feedback signal. It is assumed that the performance of the policy degrades over time, which is reasonable in some domains. They apply their approach to a robot following task, where a robot needs to follow a second robot based on visual information. Without an optimal policy, the path of the follower will deviate from the leader over time. Therefore, states encountered early in the trajectory are assumed to have higher quality than later states, which again defines an implicit preference. These preferences are used to learn the value function, which induces the policy.

2.5.5 Dueling Bandits

Another related setting is the *dueling bandit* (Yue et al., 2012), which uses preferences in an interactive, but non sequential problem setting. Feedback concerns comparisons of arms of a k-armed bandit. However, PbRL can be phrased as a dueling bandit problem, when using policies as arms. Busa-Fekete et al. (2013, 2014) realized that idea, which we discuss in Sec. 3.2.1. Approaches that learn a preference model from action preferences also share similarities (cf. Sec. 3.2.2). Each state defines a bandit with the actions as arms, but it is required to optimize over sequences of bandits. Furthermore, the mentioned approaches extend the dueling bandit setting by either generalizing over multiple states (bandits) or by deriving (multiple) action preferences from trajectory preferences. Related to dueling bandits is the use of two-point feedback for online convex optimization (Agarwal et al., 2010; Shamir, 2017), which has recently been extended to structured prediction problems (Sokolov et al., 2016).

3. Design Principles for PbRL

Preference-based reinforcement learning algorithms are subject to different design choices, which we will review in this section. First, we consider the different types of preferences, i.e., *trajectory*, *action* or *state* preferences. These types introduce different levels of complexity for the expert and algorithm (Sec. 3.1). The second design choice is how the learning problem is phrased, i.e, *directly learning a policy*, *learning a preference relation model* or a *utility function* as surrogate (Sec. 3.2). The learned representation is connected to the problem of assigning preferences to states and actions, given a delayed feedback signal. We explain this problem and possible approaches in Sec. 3.3. The next design principle is how to collect new feedback, i.e., how new trajectories are generated and selected to obtain new preferences, which we discuss in Sec. 3.4. Having obtained a representation, we need to derive an optimized policy. In Sec. 3.5, we discuss different optimization strategies that can be used to this end. Furthermore, PbRL algorithms may employ different techniques for capturing the transition dynamics, which come with different assumptions, as explained in Sec. 3.6. Sec. 3.7 shows a tabular overview of all algorithms.

3.1 Types of Feedback

There are three different types of preference feedback that can be found in the literature, action, state and trajectory preferences. The most important difference of the preference types is that they impose different challenges for the expert and the algorithm.

3.1.1 Action Preferences

An *action preference* compares two actions for the same state. The action preference $a_{i1} \succ_s a_{i2}$ expresses that in state s, action a_{i1} should be preferred to a_{i2}.

We need to distinguish between short-term optimality (e.g. optimal, immediate reward in terms of reinforcement learning) and long-term optimality (optimal, expected return). Knox and Stone (2012) and Thomaz and Breazeal (2006) analyzed this problem in an advice setting and observed that feedback relating to immediate rewards are difficult to use and feedback concerning the expected, long-term return should be preferred. This observation can be explained by the argument that it is difficult to aggregate short-term action preferences, as they are only valid for a given state. It is unclear how the short-term preferences relate to long-term optimality as we are not able to trade-off preferences for different states. Action preferences are demanding for the expert, she needs to be familiar with the expected long-term outcome. Computationally, this problem is fairly simple as it is sufficient to select the most preferred action in every state, which already implies the best long-term outcome. Hence, only generalization issues remain. The action-preference-based approach of Fürnkranz et al. (2012) also uses long-term action preferences, but not concerning an unknown, optimal policy. They supply a policy for computing the long term expectation. Hence, the expert does not need to be familiar with the expected, optimal policy but can consider samples from the provided policy for its comparison.

3.1.2 State Preferences

A *state preference* $s_{i1} \succ s_{i2}$ determines that state s_{i1} is preferred to state s_{i2}. A state preference is equivalent to saying that there is an action in state s_{i1} that is preferred to all actions available in state s_{i2}.

State preferences are more informative than action preferences as they define relations between parts of the global state space. Yet, they also suffer from the long-term/short-term optimality problem. Long-term state preferences define a clearly defined setting as it is sufficient to discover the most preferred successor state for each state and maximize its selection probability, as analyzed by Wirth and Fürnkranz (2012, 2015). Short-term state preferences do not define a trade-off, because it is unclear whether visiting an undominated state once should be preferred over visiting a rarely dominated state multiple times. Short-term state preferences are used by Zucker et al. (2010) where they try to resolve the trade-off problem using an approximated reward.

State preferences are slightly less demanding for the expert as it is not required to compare actions for a given state. However, the expert still needs to estimate the future outcome of the policy for a given state. State preferences do not directly imply a policy, but it can be easily derived with knowledge of the transition model, as explained in Sec. 3.2.3.

3.1.3 Trajectory Preferences

A *trajectory preference* $\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2}$ specifies that the trajectory $\boldsymbol{\tau}_{i1}$ should be preferred over the dominated trajectory $\boldsymbol{\tau}_{i2}$. Trajectory preferences are the most general form of feedback and the most widely used.

Trajectory preferences are arguably the least demanding preferences type for the expert as she can directly evaluate the outcomes of full trajectories. Trajectories can be evaluated in terms of the resulting behavior (e.g., a jerky vs. a straight movement towards a goal state), or by considering the occurrences of states that are known to be good. Usually, the goal is to keep complexity away from the expert, rendering trajectory preferences the most common approach. Yet, a difficulty with trajectory preferences is that the algorithm needs to determine which states or actions are responsible for the encountered preferences, which is also known as the *temporal credit assignment problem*. This assignment problem is particularly difficult if not all trajectories are starting in the same state. Trajectories with different starting states are solutions to different initial situations. Hence, the preference could be attributed to the initial situation or the applied policy. In practice, no algorithm known to the authors is capable of dealing with preferences between trajectories with different initial states.

Trajectory preferences are the most general form of preference-based feedback because, formally, all preferences can be mapped to trajectory preferences when we admit trajectories with only a single element. An action preference $a_{i1} \succ_s a_{i2}$ is equivalent to the trajectory preference $\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2}, \boldsymbol{\tau}_{i1} = \{s, a_{i1}\}, \boldsymbol{\tau}_{i2} = \{s, a_{i2}\}$. The case of a state preference $s_{i1} \succ s_{i2}$ can be reformulated as indicated above, yielding the set of trajectory preferences $\{\forall a_{i2} \in A(s_{i2}) : (s_{i1}, \pi^*(s_{i1})) \succ (s_{i2}, a_{i2})\}$.

3.2 Defining the Learning Problem

As discussed in Sec. 2.4, preferences can be used to directly learn a policy, or to learn a qualitative preference model or a quantitative utility function, both of which can then be used for deriving a policy. In the following, we will discuss different options for representing a policy, a preference model or a utility function, which have been used in various algorithms in the literature. Unless required otherwise, we only consider the loss for the preferred relation $\succ$, as explained in Sec. 2.2. We use the symbol $L^{\succ}$ for a loss that is only valid for the $\succ$ relation to differentiate from the general, single preference loss L, introduced in Sec. 2.3

3.2.1 Learning a Policy

Direct policy learning assumes a parametric policy space. The learning task is to find a parametrization that maximizes the correspondence of the policies with the observed preferences. Two different approaches have been tried, namely to induce a distribution over a parametric policy space (Wilson et al., 2012), or to compare and rank policies (Busa-Fekete et al., 2013, 2014).

Approximating the Policy Distribution. Wilson et al. (2012) approximate the policy distribution via a *Bayesian likelihood function* subject to the preference-based data probability. A parameterized policy space $\pi(a \mid s, \boldsymbol{\theta})$ is used for defining the policy distribution $\Pr(\pi \mid \zeta)$. Algorithm 1 shows how to collect the preferences. The basic strategy is to sam-

Algorithm 1 Policy Likelihood

Require: prior $\Pr(\pi)$, step limit k, sample limit m, iteration limit n

$\zeta = \emptyset$ ▷ Start with empty preference set
for $i = 0$ **to** n **do**
  $\tilde{\zeta} = \emptyset$ ▷ Clear potential preference queries
  for $j = 0$ **to** m **do**
    $s \sim \mu(s)$ ▷ Draw initial state s
    $\pi_1, \pi_2 \sim \Pr(\pi \mid \zeta)$ ▷ Draw two policies from posterior
    for $j = 0$ **to** n **do**
      $\boldsymbol{\tau}^{\pi_1} = \textsc{rollout}(s, \pi_1, k)$ ▷ Create k-step trajectory using policy π_1
      $\boldsymbol{\tau}^{\pi_2} = \textsc{rollout}(s, \pi_2, k)$ ▷ Create k-step trajectory using policy π_2
      $\tilde{\zeta} \leftarrow \tilde{\zeta} \cup (\boldsymbol{\tau}^{\pi_1}, \boldsymbol{\tau}^{\pi_2})$ ▷ Add trajectory pair to potential preference queries
    end for
  end for
  $(\boldsymbol{\tau}_1, \boldsymbol{\tau}_2) = \textsc{selectQuery}(\tilde{\zeta})$ ▷ Select a preference query
  $\zeta \leftarrow \zeta \cup \textsc{obtainTrajectoryPreferences}(\boldsymbol{\tau}_1, \boldsymbol{\tau}_2)$ ▷ Query expert
end for
return $\arg\max_\pi \Pr(\pi \mid \zeta)$ ▷ Return maximum-a-posterior

ple two policies, described by a parameter vector, from the posterior distribution $\Pr(\pi \mid \zeta)$ induced by the preferences. Each policy pair is used to create one or more trajectories, defining a potential preference query by pairing the resulting trajectories. One of multiple selection criteria, that we discuss in Sec. 3.4.3, is used to select the query to pose to the expert. Variants of the criteria also depend on the policy that created the trajectory or allow to stop the sampling process before m policy pairs are created.

The likelihood $\Pr(\pi \mid \zeta)$ is modeled by comparing trajectories $\boldsymbol{\tau}^\pi$ that are realized by a policy π with the preferred and dominated trajectories in ζ. The likelihood is high if the realized trajectories $\boldsymbol{\tau}^\pi$ of the policy π are closer to preferred trajectory, i.e.,

$$\Pr(\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2 \mid \pi) = \Phi\left(\frac{\mathbb{E}[d(\boldsymbol{\tau}_1, \boldsymbol{\tau}^\pi)] - \mathbb{E}[d(\boldsymbol{\tau}_2, \boldsymbol{\tau}^\pi)]}{\sqrt{2}\sigma_p}\right), \tag{4}$$

where the function Φ is the c.d.f of $\mathcal{N}(0, 1)$, which resembles a sigmoidal squashing function. The parameter σ_p accounts for feedback noise to allow the expert to err. The function d is a distance function comparing two trajectories. The policy distribution is then given by applying Bayes theorem, i.e,

$$\Pr(\pi \mid \zeta) \propto \Pr(\pi) \prod_i \Pr(\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2 \mid \pi), \tag{5}$$

and the posterior is approximated using *Markov Chain Monte Carlo* simulation (Andrieu et al., 2003). This approach requires the specification of a meaningful distance function, which is hard to define in many domains and requires a large amount of domain knowledge. Wilson et al. (2012) use an Euclidean distance function. Yet, Euclidean distances are hard to use in many domains such as those with high-dimensional continuous state spaces. Furthermore, if the state space is not continuous it may be hard to define a suitable distance

Algorithm 2 Policy Ranking

Require: candidate policies Π_0, step limit k, sample limit m, iteration limit n

for $i = 0$ **to** n **do**
  $\zeta = \emptyset$
  for 0 **to** m **do**
    $\pi_1, \pi_2 = \text{SELECTPOLICIES}(\Pi_i, \zeta)$ ▷ Select policies to compare
    $s \sim \mu(s)$ ▷ Draw initial state s
    $\boldsymbol{\tau}^{\pi_1} = \text{ROLLOUT}(s, \pi_1, k)$ ▷ Create k-step trajectory using policy π_a
    $\boldsymbol{\tau}^{\pi_2} = \text{ROLLOUT}(s, \pi_2, k)$ ▷ Create k-step trajectory using policy π_2
    $\zeta \leftarrow \zeta \cup \text{OBTAINTRAJECTORYPREFERENCES}(\boldsymbol{\tau}^{\pi_1}, \boldsymbol{\tau}^{\pi_2})$ ▷ Query expert
  end for
  $\Pi_{i+1} = \text{EDPS}(\Pi_i, \zeta)$ ▷ Compute candidate policies, based on preferences
end for
return $\arg\max_\pi \sum_{\pi,\pi' \in \Pi_n} \Pr(\pi \succ \pi')$ ▷ Return highest ranked policy

measure d. The definition of a parametric policy also requires domain knowledge, although good parametric policy representations are known for many domains, such as robotics (Kober et al., 2013). Often, such policy representations reduce the dimensionality of the policy and therefore the learning complexity.

Comparing and Ranking Policies. Alternatively, we can directly compare two policies π_1 and π_2 by querying the expert for different trajectory pairs $\boldsymbol{\tau}^{\pi_1}$ and $\boldsymbol{\tau}^{\pi_2}$ that have been realized by the two policies (Busa-Fekete et al., 2013, 2014). The method maintains a set of policies Π_i, described by their parameter vectors. The preference set ζ is used to compute a ranking over policies, as shown in Algorithm 2. A confidence bound method that we discuss in Sec. 3.4.1 determines which policies to select for generating the next trajectory pair for which a preference query is posed.

The outcome of this comparison is then used to compute new policies that are more likely to realize preferred trajectories. A policy π_1 is preferred over a policy π_2 if the generated trajectories $\boldsymbol{\tau}^{\pi_1}$ are on expectation preferred over the trajectories $\boldsymbol{\tau}^{\pi_2}$ realized by the second policy, i.e.,

$$\Pr(\pi_1 \succ \pi_2) \;\Leftrightarrow\; \mathbb{E}_{\boldsymbol{\tau}^{\pi_1}, \boldsymbol{\tau}^{\pi_2}}[\rho(\boldsymbol{\tau}^{\pi_1} \succ \boldsymbol{\tau}^{\pi_2})]. \tag{6}$$

For an observed set of preferences ζ, the expectation in (6) can be approximated as

$$\Pr(\pi_1 \succ \pi_2) \approx \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\boldsymbol{\tau}_i^{\pi_1} \succ \boldsymbol{\tau}_i^{\pi_2}). \tag{7}$$

The objective is to find the set of optimal polices $\pi^* = \arg\max_\pi \sum_{\pi,\pi' \in \Pi} \Pr(\pi \succ \pi')$. In contrast to Wilson et al. (2012), no distance function is required. However, preferences can not be reused as we need to obtain new preferences for each policy pair and, hence, a high amount of preferences is required. Busa-Fekete et al. (2013, 2014) deal with incomparabilities and indifference by assuming

$$\boldsymbol{\tau}_{i1} \sim \boldsymbol{\tau}_{i2} \;\Leftrightarrow\; \rho(\boldsymbol{\tau}_{i1} \prec \boldsymbol{\tau}_{i2}) = \rho(\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2}) = 0.5.$$

Algorithm 3 Preference-based Approximate Policy Iteration

Require: initial policy π_0, iteration limit m, state sample limit k, rollout limit n

for $i = 0$ **to** m **do**
  for 0 **to** k **do**
    $s \sim \mu(s)$ ▷ Sample k states per iteration
    $\boldsymbol{\Upsilon} = \emptyset, \zeta = \emptyset$
    for $\forall a \in A(s)$ **do**
      for 0 **to** n **do**
        $\boldsymbol{\Upsilon} \leftarrow \boldsymbol{\Upsilon} \cup \text{ROLLOUT}(s, a, \pi_i)$ ▷ Create trajectory, starting with (s, a)
      end for
    end for
    $\zeta \leftarrow \text{OBTAINACTIONPREFERENCES}(\boldsymbol{\Upsilon})$ ▷ Query expert
  end for
  $C^{\pi_i} = \text{COMPUTEPREFERNCEMODEL}(\zeta)$ ▷ Compute the preference function C
  $\pi_{i+1} = \text{COMPUTEPOLICY}(C^{\pi_i})$ ▷ Compute greedy policy
end for
return improved policy π_m

The optimization itself can be performed with algorithms similar to *evolutionary direct policy search* (EDPS; Heidrich-Meisner and Igel 2009). Policy comparison approaches following the ideas of Busa-Fekete et al. (2013, 2014) can be seen as cases of preference-based multi-armed bandit optimization, where each arm represents one policy. For a survey of such preference-based bandit methods, we refer to Busa-Fekete and Hüllermeier (2014).

3.2.2 Learning a Preference Model

Instead of directly learning a policy, one can also try to learn a model $C(a \succ a' \mid s)$ that predicts the expected preference relation between a and a' for a given state s. The preference relation between actions can be used to obtain a ranking for actions given a state, from which a policy can be derived in turn. The problem of learning $C(a \succ a' \mid s)$ can be phrased as a classification problem trying to correctly predict all observed preferences. As shown in Algorithm 3, *preference-based approximate policy iteration* obtains multiple trajectories for each action in k sampled states (Fürnkranz et al., 2012). This allows to derive multiple action preferences by comparing trajectories, based on the initial actions. Action preferences are directly used as training data $\{s_i, a_{i1}\} \succ \{s_i, a_{i2}\} \Leftrightarrow (a_{i1} \succ a_{i2} \mid s_i)$ for C. From these, a separate classifier $C_{ij}(s)$ is trained, which predicts whether $a_i \succ a_j$ will hold in a state s.

For deriving a ranking over all actions in a given state, the pairwise preference predictions of the trained classifiers C_{ij} may be combined via voting or weighted voting (Hüllermeier et al., 2008), where each prediction issues a vote for its preferred action. The resulting count $k(s, a)$ for each action a in state s

$$k(s,a) = \sum_{\forall a_i \in A(s), a_j \neq a} C(a_i \succ a_j \mid s) = \sum_{\forall a_i \in A(s), a_j \neq a} C_{ij}(s) \tag{8}$$

correlates with $\rho\left((s,a_i) \succ (s,a_j)\right)$ if the classifiers C approximate ρ well. Hence, we can derive a policy

$$\pi^*(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max_{a'} k(s,a') \\ 0 & \text{else} \end{cases},$$

that maximizes the realization probability for the most preferred action, satisfying (2).

Wirth and Fürnkranz (2013a,b) also use action preferences in a similar way. However, they treat C as a continuous function, returning a weighted vote for each action instead of a binary vote. Hence, $k(s,a)$ now also contains the uncertainty of the estimated preference, which allows for more efficient exploration. In this case, the function C is defined by a tabular representation with a single value for each state/action/action triplet, and updated using a gradient-based method. Fürnkranz et al. (2012) use a neural network (Bishop, 1995) that is retrained completely in every iteration, allowing for generalization at the cost of potential approximation errors.

3.2.3 Learning a Utility Function

Utility-based approaches estimate a surrogate utility $U(\boldsymbol{\tau})$ for a given trajectory. In many cases, this trajectory utility can be decomposed into state-action utilities $U(s,a)$, i.e, $U(\boldsymbol{\tau}) = \sum_t U(s_t,a_t)$. Note that this surrogate function is not directly comparable to an approximated reward or return function because it may be subject to concept drift if the estimate of the expert's optimality criterion can change over time. As in the IRL case, the expert may derive the preferences from an unknown, true reward which cannot be reconstructed as it may not be subject to the Markov property. However, in the PbRL case it is sufficient to find a reward function that induces the same, optimal policy as the true reward function. Consequently, we call this function a *utility function* to distinguish it from a fixed reward function[2].

Given the surrogate utility, the policy should then maximize the expected trajectory utility

$$\pi^* = \max_{\pi} \mathbb{E}_{\Pr_\pi(\boldsymbol{\tau})}[U(\boldsymbol{\tau})], \tag{9}$$

which is comparable to the expected return in classic *reinforcement learning*. Besides exploring the policy space, as in classic reinforcement learning, we also need to explore the utility space in preference-based RL algorithms. Estimating a utility function has the advantage that we can evaluate new trajectories without asking the expert for explicit feedback. Yet, utility function approaches may suffer from the approximation errors induced by the estimated utility function. A common utility-based approach is to assume a scalar utility for the trajectories, i.e.,

$$\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2} \Leftrightarrow U(\boldsymbol{\tau}_{i1}) > U(\boldsymbol{\tau}_{i2}).$$

A scalar utility function always exists provided that there are no incomparable trajectory pairs (von Neumann and Morgenstern, 1944).

Algorithm 4 summarizes a general process of utility-based PbRL, which forms the basis of a variety of different realizations. Preference queries are generated by sampling one or

2. The definition of a utility function may differ from the functions encountered in utility theory, but is inline with existing PbRL approaches and the preference learning literature.

Algorithm 4 Utility-based PbRL

Require: initial policy π_0, iteration limit m, state sample limit k, rollout limit n

$\zeta = \emptyset$
for $i = 0$ **to** m **do**
  $\boldsymbol{\Upsilon} = \emptyset$
  for 0 **to** k **do**
    $s \sim \mu(s)$ ▷ Sample k states per iteration
    for 0 **to** n **do**
      $\boldsymbol{\Upsilon} \leftarrow \boldsymbol{\Upsilon} \cup \textsc{rollout}(s, \pi_i)$ ▷ Create trajectory, starting with state s
    end for
  end for
  $(\boldsymbol{\tau}_1, \boldsymbol{\tau}_2) = \textsc{createQuery}(\boldsymbol{\Upsilon})$ ▷ Create a preference query
  $\zeta \leftarrow \zeta \cup \textsc{obtainTrajectoryPreferences}(\boldsymbol{\tau}_1, \boldsymbol{\tau}_2)$ ▷ Query expert
  $U^{\pi_i} = \textsc{computeUtilityFunction}(\zeta)$ ▷ Compute the utility function
  $\pi_{i+1} = \textsc{computePolicy}(U^{\pi_i})$ ▷ Compute a policy
end for
return improved policy π_m

more trajectories from a policy and derive queries by exhaustive or selective comparison (cf. Sec. 3.4.1 & 3.4.2). The preferences are then used to compute a utility function (Friedman and Savage, 1952) which, in turn, can be used to optimize policies according to the expected utility. Different learning algorithms can be used for modeling the utility function. In the following, we primarily discriminate between linear and non-linear utility models.

Linear utility functions. The most common approach is to use utility functions that are linear in a feature vector. We may use state action features $\boldsymbol{\varphi}(s, a)$ resulting in a utility $U(s, a) = \boldsymbol{\theta}^T \varphi(s, a)$, or trajectory features $\boldsymbol{\psi}(\boldsymbol{\tau})$ yielding $U(\boldsymbol{\tau}) = \boldsymbol{\theta}^T \boldsymbol{\psi}(\boldsymbol{\tau})$. In order to find such a linear utility function, we can define a loss function $\mathcal{L}$ which is given by the (weighted) sum of the pairwise disagreement loss L, i.e.,

$$\mathcal{L}(\boldsymbol{\theta}, \zeta) = \sum_{i=1}^{|\zeta|} \alpha_i L(\boldsymbol{\theta}, \zeta_i), \tag{10}$$

as described in Sec. 2.3. Using the linearity of the utility function, we can define the utility difference

$$d(\boldsymbol{\theta}, \zeta_i) = \boldsymbol{\theta}^T (\boldsymbol{\psi}(\boldsymbol{\tau}_{i1}) - \boldsymbol{\psi}(\boldsymbol{\tau}_{i2})) \tag{11}$$

for two trajectories. Different definitions of the pairwise disagreement loss $L(\boldsymbol{\theta}, \zeta_i)$ have been used in the literature and most of them use the utility difference. An intuitive loss directly correlating with the obtained binary feedback is the indicator loss

$$L^{\succ}(\boldsymbol{\theta}, \zeta_i) = \mathbb{I}\left(d(\boldsymbol{\theta}, \zeta_i) \leq \epsilon\right), \tag{12}$$

which states that the utility difference has to be always larger then ϵ, as used by Sugiyama et al. (2012). However, such a binary loss function is typically difficult to optimize and there is no notion of preference violation (or fulfillment) for trading off multiple preferences.

Name	Loss Equation	Reference
ranking SVM	$L^{\succ}(\boldsymbol{\theta}, \zeta_i) = \max(0, 1 - d(\boldsymbol{\theta}, \zeta_i))$	Akrour et al. (2011, 2012) Zucker et al. (2010) Wirth and Fürnkranz (2012, 2015) Runarsson and Lucas (2012, 2014)
IRL	$L^{\succ}(\boldsymbol{\theta}, \zeta_i) = c(d(\boldsymbol{\theta}, \zeta_i))$ $c(x) = \begin{cases} -2x & \textbf{if } x \leq 0 \\ -x & \textbf{otherwise} \end{cases}$	Wirth et al. (2016)
cumulative	$p_{\boldsymbol{\theta}}(\zeta_i) = \Phi\left(\frac{d(\boldsymbol{\theta}, \zeta_i)}{\sqrt{(2)}\sigma_p}\right)$	Kupcsik et al. (2015)
sigmoid	$p_{\text{sig }\boldsymbol{\theta}}(\zeta_i) = \frac{1}{1+\exp(-m \cdot d(\boldsymbol{\theta}, \zeta_i))}$	Christiano et al. (2017)
0/1	$p_{01\ \boldsymbol{\theta}}(\zeta_i) = \mathbb{I}\big(d(\boldsymbol{\theta}, \zeta_i) < 0\big)$	Sugiyama et al. (2012)
combined	$p_{\boldsymbol{\theta}}(\zeta_i) = \frac{\lvert\zeta\rvert - 1}{\lvert\zeta\rvert} p_{01\ \boldsymbol{\theta}}(\zeta_i) + \frac{1}{\lvert\zeta\rvert} p_{\text{sig }\boldsymbol{\theta}}(\zeta_i)$	Wirth et al. (2016)
integrated piecewise	$p_{\boldsymbol{\theta}}(\zeta_i) = \int_0^{\epsilon_{\max}} c(d(\boldsymbol{\theta}, \zeta_i), \epsilon)$ $c(x, \epsilon) = \begin{cases} 0 & \textbf{if } x < -\epsilon \\ 1 & \textbf{if } x > \epsilon \\ \frac{\epsilon + x}{2\epsilon} & \textbf{else}, \end{cases}$	Akrour et al. (2013, 2014)

Table 1: Loss functions for linear utility functions

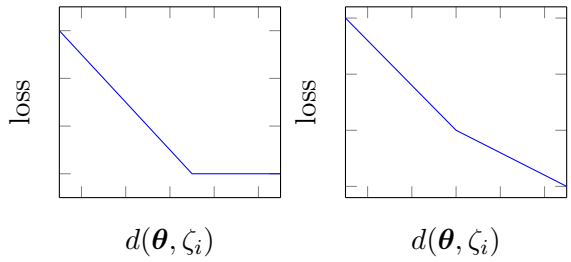
Therefore, continuous loss functions are more often used. A number of continuous loss functions have been proposed in the literature.

Tbl. 1 shows a summary of loss functions and likelihood functions that can be found in the literature. All these approaches try to find a good trade-off between potentially violating preferences and optimizing the utility difference of preferred trajectories. Some algorithms (Akrour et al., 2011, 2012; Zucker et al., 2010; Wirth and Fürnkranz, 2012, 2015; Runarsson and Lucas, 2012, 2014) define the loss function $L^{\succ}(\boldsymbol{\theta}, \zeta_i)$ as hinge loss (Fig. 2a) that can be optimized using SVM ranking algorithms (Joachims, 2002; Herbrich et al., 1999), others (Wirth et al., 2016) as a piecewise linear loss function (Fig. 2b) inspired by inverse reinforcement learning algorithms (Ng and Russell, 2000).

Other algorithms use sigmoidal loss functions to saturate the effect of a high utility differences, in order to overcome the limitations of a linear loss. Such sigmoidal loss functions are often modeled as likelihood functions $p_{\boldsymbol{\theta}}(\zeta_i)$ for the preferences. In this case, we have to optimize the log likelihood, i.e.,

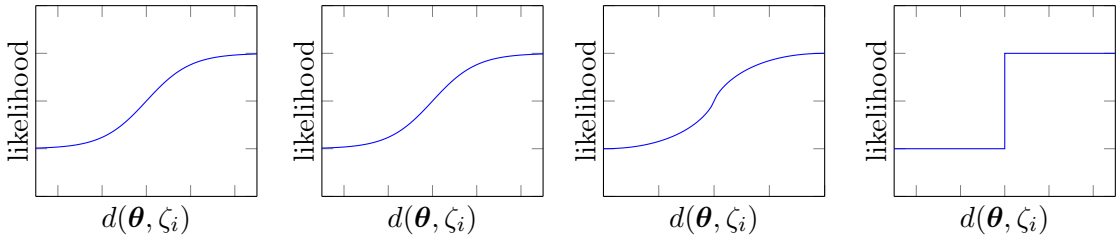
$$\mathcal{L}(\boldsymbol{\theta}, \zeta) = -\log \prod_{i=1}^{|\zeta|} p_{\boldsymbol{\theta}}(\zeta_i) = -\sum_{i=1}^{|\zeta|} \log\left(p_{\boldsymbol{\theta}}(\zeta_i)\right) = \sum_{i=1}^{|\zeta|} \mathcal{L}(\boldsymbol{\theta}, \zeta_i),$$

with $\mathcal{L}(\boldsymbol{\theta}, \zeta) = -\log\left(p_{\boldsymbol{\theta}}(\zeta_i)\right)$. Most algorithms (Akrour et al., 2014; Wirth et al., 2016; Kupcsik et al., 2015) do not directly maximize the likelihood, but obtain the posterior

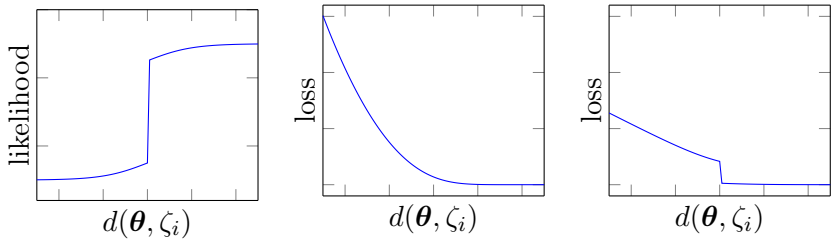


(a) hinge (b) IRL

Figure 2: Shape of loss functions



(a) cum. likelihood (b) sigmoid likelihood (c) int. piecewise likel. (d) 0/1 likelihood



(e) comb. likelihood (f) cum. log loss (g) combined log loss

Figure 3: Shape of likelihood functions and the according negative log likelihood

distribution

$$\Pr(\boldsymbol{\theta} \mid \zeta) \propto \Pr(\boldsymbol{\theta}) \prod_i p_{\boldsymbol{\theta}}(\zeta_i). \tag{13}$$

The expected utility of a new trajectory is then obtained by computing the expectation over the posterior. The posterior is computed using MCMC (Andrieu et al., 2003) by Akrour et al. (2014) and Wirth et al. (2016), whereas the latter uses *elliptic slice sampling* (ELS; Murray et al. 2010). These procedures are sampling based and can only obtain a costly approximation of the posterior. The utility function's posterior is also computed by Kupcsik et al. (2015), but using more efficient *convex optimization*. Wirth et al. (2016) also compare with a direct maximization of the likelihood. The direct maximum likelihood approach is more optimistic in the sense that it disregards the uncertainty.

The only approach that does not use an explicit cost function for obtaining the utility is by Jain et al. (2013, 2015). It uses the preferences to compute a gradient for the parameter vector of the utility function, i.e., $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha\big(\boldsymbol{\psi}(\boldsymbol{\tau}_{k1}) - \boldsymbol{\psi}(\boldsymbol{\tau}_{k2})\big)$, where α is a step-

size. In this case, the parameter vector is updated such that it is correlated with the feature vector of the preferred trajectory. Such a gradient can be mapped to the simple loss function $L(\boldsymbol{\theta}, \zeta_i) = -d(\boldsymbol{\theta}, \zeta_i)$. In general, it is unclear how the aggregated utility loss $\mathcal{L}(\boldsymbol{\theta}, \zeta)$ is related to the policy loss $\mathcal{L}(\pi, \zeta)$ (see Sec. 2.3), as the policy is subject to the system dynamics whereas the utility is not. Nevertheless, it is assumed that maximizing the expected utility (9) yields more preferred trajectories.

Non-Linear Utility Functions. Few approaches allow the use of non-linear utility functions. The utility feature space is usually assumed to be defined in advance, shifting the problem to the user. However, defining such a feature space usually requires expert knowledge. Methods that can use non-linear utilities are easier to apply but may require a higher amount of preferences or trajectory samples, because the learning problem is more complex.

Gritsenko and Berenson (2014) convert the preferences into a ranking and do not operate directly on the preference level. Instead, their algorithm tries to learn a utility distance $d(U, \zeta_i)$, which models the rank difference k_i of a trajectory pair $\boldsymbol{\tau}_{i1}$ and $\boldsymbol{\tau}_{i1}$, i.e.,

$$L^{\succ}(U, \zeta_i) = \|d(U, \zeta_i) - k_i\|_2, \tag{14}$$

This loss function can be minimized by any regression algorithm. For example, Gritsenko and Berenson (2014) use M5' (Wang and Witten, 1997).

The rank values are subject to a somewhat arbitrary scale (e.g., the difference between mispredictions of neighboring ranks is always the same). As an alternative, the rank values can be encoded as classes instead of numeric values (Gritsenko and Berenson, 2014). Such a multi-class problem can be learned by any conventional classification algorithm, Gritsenko and Berenson (2014) use C4.5 (Quinlan, 1993). The classification model is again used as the utility function where the class is used as utility value. The multi-class loss can not be directly mapped to a preference loss $L^{\succ}(\boldsymbol{\theta}, \zeta_i)$ as the classification problem is discrete and disregards the rank or utility difference.

Christiano et al. (2017) use deep neural networks for approximating a non-linear utility function. They directly minimize the negative log likelihood $-\sum_{i=1}^{N} \log\left(p_{\boldsymbol{\theta}}(\zeta_i)\right)$ with a sigmoid likelihood function. The utility function is not linear in the features, but convex. Hence, it is possible to compute a gradient for the parameters $\boldsymbol{\theta}$ and use common backpropagation techniques for minimizing the loss.

Wirth et al. (2016) and Kupcsik et al. (2015) also allow for non-linear utility functions by using kernel-based feature spaces. While the likelihood function is linear in the resulting, infinite feature space, the effective feature space depends non-linearly on the used samples. Wirth et al. (2016) use Gaussian kernel functions whereas Kupcsik et al. (2015) employ Gaussian process preference learning (Chu and Ghahramani, 2005).

In case the utility function should be applicable to states or state/action pairs, the linearity of the MDPs reward has to be considered (Wirth et al., 2016). Hence, the utility must be linear over the state/action pairs within a trajectory and non-linearity has to be achieved by mapping the state/action features itself, as we will discuss in Sec. 3.3.3.

3.3 The Temporal Credit Assignment Problem

As in all sequence learning problems, a key problem is that it is usually not known which states or actions are responsible for the obtained preference. This *temporal credit assignment*

problem is comparable to the delayed reward problem in classic reinforcement learning. It is possible to circumvent it by directly estimating a policy's return in order to approximate the policy value. Methods that provide explicit solutions to the credit assignment typically come with the advantage that standard, reward-based RL methods can be employed. Yet, if we try to solve the credit assignment problem explicitly, we also require the expert to comply with the Markov property. This assumption can easily be violated if we do not use a full state representation, i.e., if the expert has more knowledge about the state than the policy. Depending on how the considered algorithm solves the credit assignment problem, different types of utility functions or preferences can be inferred, which we will discuss in the next subsections.

3.3.1 Value-based Utility

Value-based utility functions estimate the expected long-term utility for a given state, similar to a value function in reinforcement learning. Wirth and Fürnkranz (2012, 2015); Runarsson and Lucas (2012, 2014) use state-preferences $s_{i1} \succ s_{i2}$, which are assumed to originate from the expected utility of the long-term behavior, if we are in the current state and follow an optimal policy. The temporal credit assignment problem is in this case left to the human expert and also not solved explicitly by the algorithm, as we can simply select actions, leading to states with maximal (long-term) utility. To compute the utility $U(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$ from a state preference, we can again make use of the utility difference $d(\boldsymbol{\theta}, \zeta_i) = \boldsymbol{\theta}^T(\boldsymbol{\phi}(s_{i1}) - \boldsymbol{\phi}(s_{i2}))$ using one of the specified loss functions from Tbl. 1. Using the utility function, we can define the policy

$$\pi^*(a \mid s) = \mathbb{I}(a = \arg\max_{a'} \mathbb{E}_\delta[U(s') \mid s, a']), \tag{15}$$

as a greedy policy maximizing the expected value of the next state s'. However, this expectation can only be computed exactly if the transition model δ is known and the state space is discrete. Moreover, value-based utilities always depend on a specific, optimal policy. Hence, it is difficult to transfer the learned value-utility function to other domains.

3.3.2 Return-based Utility

Many approaches that use trajectory preferences circumvent the temporal credit assignment problem by disregarding the temporal structure of the solution. They directly optimize the policy return (Akrour et al., 2011, 2012; Jain et al., 2013, 2015; Gritsenko and Berenson, 2014; Kupcsik et al., 2015) as defined by a return-utility function $U(\boldsymbol{\tau})$. The return utility function $U(\boldsymbol{\tau}) = \boldsymbol{\theta}^T \boldsymbol{\psi}(\boldsymbol{\tau})$ is typically defined by a low dimensional trajectory feature space $\boldsymbol{\psi}(\boldsymbol{\tau})$.

This feature space is usually assumed to be domain-specific (Zucker et al., 2010; Jain et al., 2013, 2015; Gritsenko and Berenson, 2014), however, Akrour et al. (2011, 2012) proposed to cluster the state space using ϵ-means clustering (Duda and Hart, 1973) for defining trajectory features. The trajectory features are given by the relative amount of states in a trajectory belonging to a cluster. Such features can not be directly described as state features as they also depend on the length of the trajectory.

Kupcsik et al. (2015) also learn a return-based utility $U(\boldsymbol{\tau})$ but represent a trajectory $\boldsymbol{\tau} = [\boldsymbol{s}, \boldsymbol{\omega}]$ with a context vector $\boldsymbol{s}$ that describes the current task and a parameter vector $\boldsymbol{\omega}$

that specifies the policy. Instead of using a linear feature representation, a Gaussian process is used to model the return utility $U(\boldsymbol{\tau})$, see Sec. 3.2.3.

3.3.3 Reward-based Utility

Many approaches obtain a state-action utility function that resembles a reward function in classical RL methods, i.e., it evaluates the immediate quality of executing action a in state s without considering the long-term behavior. Using reward-based utilities allows us to apply standard RL methods to improve the policy.

Zucker et al. (2010) propose a simple solution by assuming state preferences $s_1 \succ s_2$ that can be directly used for learning a state utility $U(s)$. For learning a reward-based utility from trajectory preferences, Akrour et al. (2013, 2011, 2014) and Wirth et al. (2016) use the *feature average* (Ng and Russell, 2000) along a trajectory

$$\boldsymbol{\psi}(\boldsymbol{\tau}) = \sum_{t=0}^{|\boldsymbol{\tau}|} \gamma^t \boldsymbol{\varphi}(s_t, a_t), \tag{16}$$

to define trajectory features. Hence, the utility return is defined by

$$U(\boldsymbol{\tau}) = \boldsymbol{\theta}^T \boldsymbol{\psi}(\boldsymbol{\tau}) = \sum_{t=0}^{|\boldsymbol{\tau}|} \gamma^t U(s_t, a_t),$$

with $U(s_t, a_t) = \boldsymbol{\theta}^T \boldsymbol{\varphi}(s_t, a_t)$, which coincides with the definition of the return in reinforcement learning. The trajectory utility $U(\boldsymbol{\tau})$ is a linear sum over the state/action utilities but the state/action utilities themselves can be non-linear. As an example, Wirth et al. (2016) use a kernel function to obtain such a non-linear utility by applying the kernel on a state/action level. Sugiyama et al. (2012) proposed an alternative learning method, based on the sum of future utilities

$$\begin{aligned} U(s,a) &= \sum_{s' \in S} \left(\delta(s' \mid s, a) \left(U(s,s') + \max_{a' \in A} \sum_{s'' \in S} \delta(s'', s', a') U(s' \mid s'') \right) \right), \\ U(s,s') &= \boldsymbol{\theta}^T \boldsymbol{\varphi}(s,s'), \end{aligned} \tag{17}$$

with $U(s, s')$ as the utility for the state transition s to s'. However, this idea is only applicable with a known transition function and a discrete action space.

More recently, a method was proposed that does not require the utility to be linear in the state/action features. Christiano et al. (2017) define a learning method for a trajectory utility $U(\boldsymbol{\tau}) = \sum_{t=0}^{|\boldsymbol{\tau}|} U(s_t, a_t)$, that uses a deep neural network for $U(s_t, a_t)$. This method greatly improves on the scalability of the learning process and allows more expressive utility functions, but requires a high amount of preferences.

In contrast to conventional reinforcement learning, a discount factor of $\gamma = 1$ is used for $U(\boldsymbol{\tau})$ in all approaches because the expert should not need to consider the effects of decay. However, using an undiscounted return is not a major problem as all considered trajectories have a finite length. The parameters $\boldsymbol{\theta}$ can now be estimated by using one of the loss functions presented in Sec. 3.2.3, where trajectory preferences are used as constraints for

the utility return $U(\boldsymbol{\tau})$. The underlying assumption of using immediate utility functions is that the expert's evaluation function complies with the Markov property. If this is not the case, the decomposition of the utility return in immediate utilities is error prone. Instead of using state-action utilities $U(s, a)$, some algorithms (Wirth et al., 2016) also disregard action costs and just use a state utility function, i.e., $U(s, a) = U(s)$.

In contrast to value-based utility functions, reward-based utility functions can be easily transferred across domains. Moreover, a reward-based utility is also independent of the system dynamics and, therefore, often has a simpler structure than value based utilities rendering them simpler to learn and generalize.

3.3.4 Intermediate Action Preferences

Wirth and Fürnkranz (2013a) extend the action preference algorithm of Fürnkranz et al. (2012) by deriving additional action preferences for intermediate states in trajectory preferences, defining an approximate solution to the temporal credit assignment problem. As in the original algorithm, preferences are assumed to relate to the long-term expectation. The assignment problem is solved by attributing preferences to an (unknown) subset of action choices made in states occurring in both trajectories, when following a fixed policy in all other states. This enables the computation of the probability that a single state/action pair is responsible for the encountered preference.

This approach may converge faster than the one of Fürnkranz et al. (2012) using a lower amount of preferences. However, the speedup comes at the cost of possible approximation errors due to possibly incorrect, heuristic estimates of the preferences. Wirth and Fürnkranz (2013b) extend this idea further by obtaining more reliable, intermediate preferences based on the assumption that each time step contributed equally to the return of the trajectory. As a result, an even lower amount of preferences is sufficient. However, many domains do not satisfy this uniform reward assumption.

3.4 Trajectory Preference Elicitation

PbRL may be viewed as an interactive process that iteratively queries the expert's preference function to obtain more knowledge about the objectives of the expert while also exploring the system dynamics. In decision making, this process is also known as *preference elicitation*.

Several approaches disregard this problem and only work with an initial set of preferences (Wirth and Fürnkranz, 2012, 2015; Runarsson and Lucas, 2012, 2014; Gritsenko and Berenson, 2014; Zucker et al., 2010; Sugiyama et al., 2012). Fürnkranz et al. (2012) collect new preference feedback, but assume exhaustive samples of the system dynamics and, hence, no additional exploration is needed.

Interactive PbRL algorithms need to decide how to generate new trajectories and how to use these trajectories to query the expert's preference function. Typically, an exploration method is used that reduces the uncertainty of the transition function as well as the expert's preference function. Yet, some approaches apply two distinct methods for exploring the system dynamics and the preference space.

Usually, we want to minimize the amount of expert queries as they require costly, human interaction while generating new trajectories can be a cheaper, mostly autonomous process. Moreover, a trajectory can give us useful information about the system dynamics even if

it is not used for preference generation. Hence, the number of trajectories can be considerably higher than the number of generated preferences. Yet, many algorithms ignore the potentially higher costs of preference generation in comparison to trajectory generation, as we will elaborate in the following discussion.

3.4.1 Trajectory Generation

Interactive PbRL algorithms need to generate diverse trajectories. In order to be informative, the obtained preferences should be different from existing trajectories. Yet, the trajectories should also be close to optimal in order to obtain useful information. Furthermore, the trajectories need to contain sufficient information about the transition function to compute an optimal policy.

Undirected Homogeneous Exploration. *Homogeneous exploration* employs a single exploration to generate diverse solutions, either directed or undirected. Such methods are undirected if they only compute stochastic policies that allow deviations from the optimal strategy while directed methods implement a separate criterion for guiding the exploration in areas of the state space where uncertainty can be reduced. Directed methods usually work better in low-dimensional policy spaces but do not scale well.

Most approaches (Busa-Fekete et al., 2013, 2014; Jain et al., 2013, 2015; Wirth and Fürnkranz, 2013a,b; Kupcsik et al., 2015; Wirth et al., 2016; Christiano et al., 2017) only use undirected exploration for the system dynamics and make the assumption that this will also explore the expert's preference function sufficiently well. Busa-Fekete et al. (2013, 2014) use a *covariance matrix adaptation evolution strategy* (CMA-ES; Hansen and Kern 2004) for optimization and exploration. They implement a criterion for limiting the amount of samples that have to be evaluated to obtain a ranking. The algorithm maintains a collection of candidate policies and discards candidates once they can be considered suboptimal, according to the Hoeffding bound (Hoeffding, 1963). Jain et al. (2013, 2015) use *rapidly-exploring random trees* (LaValle and Kuffner, 1999) whereas Wirth and Fürnkranz (2013a,b) employ approximate policy iteration (Dimitrakakis and Lagoudakis, 2008) with an EXP3-like exploration method (Auer et al., 1995).

Wirth et al. (2016) use *relative entropy policy search* (REPS; Peters et al. 2010) for computing a stochastic policy based on the policy's return, estimated with *least squares temporal difference (LSTD)* learning (Boyan, 1999). Kupcsik et al. (2015) employ *contextual REPS* (Kupcsik et al., 2014), but directly in parameter space instead of state action space. This variant is able to evaluate parametric policies without requiring transition samples due to the black-box character of the algorithm. The result of the policy improvement step is a distribution over parametric policies that is used to sample new trajectories for querying new preferences. In turn, the distribution over possible utility functions is updated for computing an improved estimate of the expected utilities for the next policy update step.

Christiano et al. (2017) use Trust Region Policy Optimization (TRPO; Schulman et al. 2015) and (synchronous) A3C (Mnih et al., 2016), which are policy optimization algorithms for RL with deep neural networks. Both algorithms also ensure exploration by defining stochastic policies, hence they are undirected.

Directed Homogeneous Exploration. In contrast to undirected approaches, Wilson et al. (2012) propose two directed exploration criteria that are derived from the preference

learning method explained in Sec. 3.4.3. However, they also only use a single, homogeneous exploration method. The policy improvement is computed by updating the posterior distribution of a parametric policy space, given a preference-based likelihood function. A new preference query is then created by applying two policies to the MDP, selected by the directed exploration criterion. The computation of these criteria requires multiple trajectories, but they are not used for the queries themselves or for the policy improvement step.

Heterogeneous Exploration. *Heterogeneous exploration* methods use two distinct strategies for exploring the system dynamics and the preference function. For example, Akrour et al. (2011, 2012, 2014) employ different criteria that are well suited for the two tasks. A policy search strategy with an intrinsic exploration method is used to optimize a given utility function. In addition, the utility function implements a criterion for directed exploration of the preference function. Therefore, the resulting policy can be used to sample trajectories for preference queries that reduce the uncertainty over the expert's feedback. Such a strategy comes at the cost of sampling new trajectories for multiple policy update steps before each new preference query. Akrour et al. (2014) propose an alternative version using *least squares policy iteration* (Lagoudakis and Parr, 2003), that does not require additional samples for updating the policy in an off-policy manner. Yet, this approach requires a sufficient amount of samples collected in advance which is unrealistic in many scenarios.

User-Guided Exploration. *User-guided exploration* allows the expert to provide additional trajectories to guide the exploration process. The algorithm by Zucker et al. (2010) assumes a set of initial state preferences and optimize the resulting utility function using a randomized planner (*anytime A**; Likhachev et al. 2003). The planning algorithm samples new trajectories and the expert can select any encountered state to provide additional preferences, i.e., the expert is guiding the exploration of its preference space. In case the expert does not provide additional preferences, no further exploration of the preference function is used. The approach of Jain et al. (2013, 2015) extends the used homogeneous strategy with user-guided exploration. Given a trajectory that has been autonomously generated, the expert may correct the trajectory which simultaneously also provides a preference.

3.4.2 Preference Query Generation

Having generated the trajectories, we need to decide which trajectories to use to query the expert's evaluation function. Typically, the newly generated trajectories are compared to so far undominated trajectories (Akrour et al., 2012, 2011; Wirth et al., 2016; Akrour et al., 2014). Yet, this solution is only feasible if all trajectories are directly comparable, e.g., they start in the same initial state, which is a standard assumption for most PbRL algorithms. Alternatively, both trajectories are explicitly generated by the algorithm (Busa-Fekete et al., 2013, 2014; Jain et al., 2013, 2015; Wirth and Fürnkranz, 2013a,b; Kupcsik et al., 2015; Wilson et al., 2012) or the user selects the samples for a preference query (Zucker et al., 2010).

Exhaustive Preference Query Generation. Some approaches require that each trajectory is evaluated by the expert by defining preferences regarding every possible trajectory pair, either for comparing and selecting policies (Busa-Fekete et al., 2013, 2014) or for evaluating and updating a single policy (Jain et al., 2013, 2015; Wirth and Fürnkranz, 2013a,b;

Kupcsik et al., 2015). These approaches ignore that generating preferences can be more costly than generating trajectories and, hence, typically require a large amount of expert queries.

Greedy Preference Query Generation. In greedy approaches (Akrour et al., 2014), we fully optimize a given objective, described in Sec. 3.4.3, which can be either an undirected or directed exploration objective. After the optimization is finished, we use a trajectory generated from the optimized policy to query the expert's evaluation function. While this approach is typically much more efficient in the number of expert queries, it may need many trajectories to be generated in order to optimize the posed objective. Furthermore, many of these trajectories might have been unnecessarily generated as the posed objective is typically quite error-prone in the beginning of the learning process. Hence, the optimization might explore poor trajectories which could have been avoided if new preferences would have been requested earlier before reaching the maximum of the given objective.

Interleaved Preference Query Generation. The above-mentioned problem was recognized by several authors (Akrour et al., 2012, 2011; Wirth et al., 2016; Wilson et al., 2012; Christiano et al., 2017). A simple solution to alleviate it is to prematurely stop the optimization of the given objective and request new preferences. The preferences are subsequently used to update the given optimization objective (see Sec. 3.4.3). This approach is followed by Akrour et al. (2012, 2011) where the search in parameter space is stopped early, after a given number of iterations. Wirth et al. (2016); Christiano et al. (2017) request new preferences after a single iteration of the used policy optimization algorithm. In each iteration, several trajectories are generated by the current policy but only a subset (typically one) is used for an expert query. Wirth et al. (2016) propose to use the trajectory with the highest expected accumulated utility, whereas Christiano et al. (2017) use a variance estimate, based on an ensemble technique. Wilson et al. (2012) apply the same idea to a parametric policy space, using a selection criterion based on expected belief change. An alternative method selects two policies that are used for creating trajectories for a new preference query. In contrast to Wirth et al. (2016), this algorithm completely discards trajectories that are not used for preference queries, as it is only able to use explicitly evaluated trajectories. Hence, not all trajectories are evaluated, but only evaluated trajectories are used to update the policy.

The resulting algorithms allow to choose the trade-off between the cost of trajectory generation and preference generation by controlling the ratio of generated trajectories to generated preferences. Akrour et al. (2012, 2011); Wirth et al. (2016) can also efficiently explore the trajectory space as well as the preference space as they can use trajectories not evaluated by the expert for policy optimization.

3.4.3 Objectives for Exploring the Preference Function Space

A few algorithms formulate an explicit objective in order to efficiently reduce the uncertainty over the expert's preference function. Different versions have been proposed, but no exhaustive comparison of all exploration methods can be found in the literature. A limited, comparative evaluation was conducted by Wilson et al. (2012) who compared two criteria proposed in the paper.

They suggest to sample multiple policies from the posterior distribution of the policy space and use them to compute trajectories for preference queries. The first method only

considers the expected difference of the trajectories generated by two policies

$$\int_{(\boldsymbol{\tau}_i,\boldsymbol{\tau}_j)} \Pr(\boldsymbol{\tau}_i \mid \pi_i) \Pr(\boldsymbol{\tau}_j \mid \pi_j) d(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j), \tag{18}$$

where $d(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j)$ is a trajectory difference function (see (4)). If two policies disagree greatly on what trajectories to create, it is assumed to be beneficial to rule out one. Therefore, samples from π_i and π_j are evaluated and a query is posed in case the value of the criterion exceeds a given threshold.

The second criterion computes the expected belief change over the policy space that results from adding a preference

$$V(\Pr(\pi \mid \zeta) || \Pr(\pi \mid \zeta \cup \{\boldsymbol{\tau}_{i1}, p_i, \boldsymbol{\tau}_{i2}\}), \tag{19}$$

using potential queries by sampling policies from the current distribution. Different measures could be used as V, e.g. the Kullback-Leibler divergence. Wilson et al. (2012) use an approximation of the variational distance. The query maximizing the criterion, based on a finite trajectory sample set, is then posed to the expert.

Akrour et al. (2011, 2012, 2014) define a utility function that includes an exploration term. The simplest form

$$\begin{aligned} \tilde{U}(\pi) &= U(\pi) + \alpha E(\pi), \\ U(\pi) &= \mathbb{E}_{\Pr_\pi(\boldsymbol{\tau})}[U(\boldsymbol{\tau})], \\ E(\pi) &= \min \Delta(\pi, \pi'), \pi' \in \Pi_t, \\ \Delta(\pi, \pi') &= \frac{|S_1| + |S_2| - |S_3|}{\sqrt{|S_1| + |S_3|}\sqrt{|S_2| + |S_3|}}, \end{aligned} \tag{20}$$

used by Akrour et al. (2011), adds a diversity term to the expected utility of a policy, comparing the states in S_1, only visited by π, the states in S_2, only visited by π' and the states in S_3 visited by both policies. Π_t are all policies obtained up to the current iteration. Candidate policies are generated by an evolutionary strategy and $\mathbb{E}_{\Pr_\pi(\boldsymbol{\tau})}[U(\boldsymbol{\tau})]$ is approximated via samples. The diversity function $\Delta(\pi, \pi')$ rewards differences in the states visited and is only applicable to discrete state spaces, however, a discrete description can be obtained using a clustering algorithm (see Sec. 3.3.2). Akrour et al. (2012, 2014) propose a different selection criterion that maximizes the expected utility of selection

$$\tilde{U}(\boldsymbol{\tau}) = \Pr(\boldsymbol{\tau} \succ \boldsymbol{\tau}^* \mid U) \mathbb{E}_{\zeta \cup \{\boldsymbol{\tau} \succ \boldsymbol{\tau}^*\}}[U(\boldsymbol{\tau})] + \Pr(\boldsymbol{\tau} \prec \boldsymbol{\tau}^* \mid U) \mathbb{E}_{\zeta \cup \{\boldsymbol{\tau} \prec \boldsymbol{\tau}^*\}}[U(\boldsymbol{\tau}^*)], \tag{21}$$

with $\boldsymbol{\tau}^*$ as the currently undominated trajectory. It computes the expected utility for the undominated trajectory, differentiating the cases where the new trajectory $\boldsymbol{\tau}$ is preferred over the current best and where it does not improve. $\Pr(\boldsymbol{\tau} \succ \boldsymbol{\tau}^* \mid U)$ is the probability of improvement, given the current distribution over utility functions. The expected utility $\mathbb{E}_{\zeta \cup \{\boldsymbol{\tau} \succ \boldsymbol{\tau}^*\}}[U(\boldsymbol{\tau})]$ is then computed with the assumption that the new preference is added to the preference set ζ. The two publications (Akrour et al., 2012, 2014) differ in the method used to approximate the criterion.

Wirth et al. (2016) also maintains the current, best trajectory for preference queries. The trajectory maximizing the expected utility

$$\boldsymbol{\tau} = \arg\max_{\boldsymbol{\tau} \in \boldsymbol{\tau}^{[i]}} \mathbb{E}\left[U_\zeta(\boldsymbol{\tau})\right], \tag{22}$$

is selected to obtain the second trajectory for the preference query. $\boldsymbol{\tau}^{[i]}$ is a set of trajectories obtained by sampling from the current policy π_i. Trajectories obtained from sampling former policies are not considered.

Christiano et al. (2017) also use trajectories obtained from the current policy, but evaluate pairs based on the variance of the utility. They approximate the variance by obtaining utility samples from an ensemble of utility functions. Each utility function uses a randomly sampled subset of the available preferences, hence, the learned functions may differ.

3.5 Policy Optimization

A major consideration is how the policy is optimized and how the policy optimization method affects the optimality of the learned policy and the sample requirements of the algorithm.

3.5.1 Direct Policy Search

Direct policy search approaches typically directly search in the parameter space of the policy and do not assume the Markov property or use dynamic programming such as standard reinforcement learning approaches. They maximize the objective defined by (2) and (3), however, this is difficult to compute directly due to the dependence on the transition dynamics. Hence, methods like policy likelihood or ranking (cf. Sec. 3.2.1) are used. Alternatively, common return-based optimization can be employed by defining an utility-based return, as introduced in Sec. 3.3.2.

Direct policy search methods are typically not very sample-efficient, as the temporal structure of the problem is ignored, but they can obtain good results because of their simplicity. These methods can be typically used for policies with a moderate number of parameters, typically less than one hundred.

Wilson et al. (2012) employ *Markov Chain Monte Carlo* (MCMC; Andrieu et al. 2003) sampling to obtain the posterior distribution of a parametric policy space. The sampling is computational costly, but does not require new trajectory samples. A sampling approach is also used by Busa-Fekete et al. (2013, 2014), but within an *evolutionary algorithm* where policies are described with a set of parameters. Akrour et al. (2011, 2012, 2014) also use an evolutionary strategy ($1 + \lambda$ ES; Auger 2005) for optimization in a parametric policy space, which requires a high amount of trajectory samples.

The contextual REPS (Kupcsik et al., 2014) algorithm, as employed by Kupcsik et al. (2015), is also a strategy for optimizing parameters of the policy. In the contextual settings, we can learn to adapt the policy parameters to the context of the task. For example, it can learn to adapt the robot's trajectory if we need to throw a ball for a larger distance (Kupcsik et al., 2014).

3.5.2 Value-based Reinforcement Learning

Value-based approaches exploit the temporal structure of the problem by using the Markov property, which, in principle allows for more sample efficient policy optimization. However, these methods might suffer from approximation errors which can lead to instabilities and divergence. Typically, value-based methods use some variant of policy iteration that is again composed of a policy evaluation step and a policy improvement step. In the policy evaluation step, the estimated reward-based utility (Akrour et al. 2014; Wirth et al. 2016; Christiano et al. 2017; cf. Sec. 3.2.3) is used to compute a value function for the policy. This value function can, for example, be estimated by temporal difference methods such as *least squares temporal difference learning* (LSTD; Boyan 1999). Given the value function, a new policy can be obtained in the policy improvement step. For example, Wirth et al. (2016) uses a variant of the *relative entropy policy search* (REPS) algorithm (Peters et al., 2010) to update the policy. The REPS algorithm performs a soft-greedy policy update that stabilizes the policy iteration process and is also applicable in continuous action spaces. The TRPO (Schulman et al., 2015) algorithm employed by Christiano et al. (2017) works comparably, but uses deep neural networks for approximating the value function and the policy. A separate algorithm for computing the value function is not required.

In discrete action spaces, greedy updates using the max-operator have been used (Akrour et al., 2014). However, this approach requires a sufficient amount of transition samples to be obtained beforehand which again stabilizes the policy iteration process.

Other authors (Fürnkranz et al., 2012; Wirth and Fürnkranz, 2012, 2013a,b, 2015; Runarsson and Lucas, 2012, 2014; Sugiyama et al., 2012) assume a utility function directly defining the expected outcome of a state or action. Hence, an optimal policy is derived by directly selecting the action maximizing the utility. In case of state values (Wirth and Fürnkranz, 2012, 2015; Runarsson and Lucas, 2012, 2014), this step requires the transition model to be known.

3.5.3 Planning

Many other approaches (Zucker et al., 2010; Jain et al., 2013, 2015; Gritsenko and Berenson, 2014) use planning algorithms for optimizing policies. Planning-based algorithms use utility-based approaches to derive an evaluation measure for trajectories (or state/actions). Furthermore, a model of the system dynamics is required to determine the effects of actions. It is then possible to plan a sequence of actions that maximizes the given utility. However, a model of the system dynamics is in many cases not available with a sufficient accuracy.

The use of sampling-based *rapidly-exploring random trees* (RRT; LaValle and Kuffner 1999) and their *constrained bi-directional* variant (CBiRRT; Berenson et al. 2009) was explored in (Jain et al., 2013, 2015; Gritsenko and Berenson, 2014). RRTs are guaranteed to find a solution to a planning problem, but provide no guarantees for the optimality of the found solution. Other planning algorithms come with such guarantees, such as *anytime A** (ARA*; Likhachev et al. 2003), as used by Zucker et al. (2010). Planning methods require an accurate model of the system dynamics and inherently suffer from model errors. Small inaccuracies in the model might be exploited by the planning method which may result in solutions that are not feasible on the real system.

3.6 Modelling the Transition Dynamics

The reviewed algorithms also differ with respect to the amount of available model knowledge, which has a big influence on which tasks the approaches are applicable. *Model-based approaches* assume that the system dynamics are known in advance, which allows the use of planning algorithms (Jain et al., 2013, 2015; Gritsenko and Berenson, 2014; Zucker et al., 2010) or to directly derive an optimal policy (Wirth and Fürnkranz, 2012, 2015; Runarsson and Lucas, 2012, 2014; Sugiyama et al., 2012). In addition, some approaches require a model to be able to simulate the system initialized at arbitrary states (Fürnkranz et al., 2012; Wilson et al., 2012).

The transition dynamics of real world problems are usually not known, as they are subject to a wide variate of (unknown) factors. Hence, many model-based approaches try to learn the dynamics based on observed transitions (Grünewälder et al., 2012). However, model learning can be a complicated task in particular for continuous systems and is a research field on its own (Nguyen-Tuong and Peters, 2011). It might require a high amount of samples, we need to choose a suitable model class and the resulting model errors often decrease the quality of the resulting policy.

Alternatively, *model-free approaches* neither assume a model, nor approximate it (Fürnkranz et al., 2012; Wirth and Fürnkranz, 2013a,b; Busa-Fekete et al., 2013, 2014; Wirth et al., 2016; Akrour et al., 2014). For example, Akrour et al. (2014) assume a large database of pre-generated samples which can be subsequently used for model-free, off-policy reinforcement learning algorithms. Most model-free PbRL approaches, with the exception of Wirth et al. (2016), are not able to reuse samples, and therefore require a large amount of samples for comparing or evaluating policies. All algorithms that employ direct policy search strategies are in principle model-free, as the only requirement of such algorithms is that a policy can be executed on the real system. However, as these algorithms often require a lot of policy executions in order to find an optimal policy, it is often assumed that a simulator is available to avoid the costly optimization on the real system. A notable exception is the algorithm by Kupcsik et al. (2015) where the policy optimization is performed without requiring the transition dynamics. Trajectories from the real system are only used to compute an evaluation function for parametric policies.

3.7 Summary of PbRL Algorithms

Tbl. 2 shows a tabular summary of the discussed algorithms, grouped by the different learning problems, introduced in Sec. 3.2. We list the introduced key concepts and the different *requirements*, stated throughout this paper. The *preferences* column is related to the different types of preferences mentioned in Sec. 3.1. The *trajectory generation* and *preference (query) generation* columns are based on the subsections of Sec. 3.4, where we mention how trajectories and preference queries are generated to resolve the exploration problem. The various approaches use different optimization strategies for performing policy optimization and surrogate loss minimization, if applicable. The columns *surrogate optimization* and *policy optimization* show the strategies mentioned in Sec. 3.2 and Sec. 3.5 explicitly.

Learning a Policy	**Preferences**	**Traj. Gen.**	**Pref. Gen.**	**Surrogate Opt.**	**Policy Opt.**	**Require.**
Busa-Fekete et al. (2013, 2014)	Trajectory	Homogen	Exhaustive	-	CMA-ES	PP
Wilson et al. (2012)	Trajectory	Homogen	Interleaved	-	MCMC	PP,M/S_0
Learning a Preference Model	**Preferences**	**Traj. Gen.**	**Pref. Gen.**	**Surrogate Opt.**	**Policy Opt.**	**Require.**
Fürnkranz et al. (2012)	Action	-	Exhaustive	MLP	Direct Max.	M/S_0, DA
Wirth and Fürnkranz (2013a,b)	Trajectory	Homogen	Exhaustive	Gradient	Direct Max.	DS, DA
Learning a Utility Function	**Preferences**	**Traj. Gen.**	**Pref. Gen.**	**Surrogate Opt.**	**Policy Opt.**	**Require.**
Akrour et al. (2011, 2012)	Trajectory	Heterogen	Interleaved	SVM	$1+\lambda$-ES	PP
Akrour et al. (2014)	Trajectory	Heterogen	Greedy	MCMC	LSTD, CMA-ES	MA or PP
Christiano et al. (2017)	Trajectory	Homogen	Interleaved	Gradient	TRPO, A3C	
Gritsenko and Berenson (2014)	Trajectory	-	-	C4.5, M5P	CBiRRT	M
Jain et al. (2013, 2015)	Trajectory	Homogen+Guided	Exhaustive	Gradient	RRT	M
Kupcsik et al. (2015)	Trajectory	Homogen	Exhaustive	convex solver	C-REPS	PP
Runarsson and Lucas (2012, 2014)	State	-	-	SVM	Direct Max.	M
Sugiyama et al. (2012)	Trajectory	-	-	convex solver	Direct Max.	M, DS, DA
Wirth and Fürnkranz (2012, 2015)	State	-	-	SVM, LP	Direct Max.	M
Wirth et al. (2016)	Trajectory	Homogen	Interleaved	LP, ELS	LSTD, REPS	
Zucker et al. (2010)	State	Homogen	Guided	SVM	ARA*	M

Table 2: Overview of PbRL approaches by design principle

Algorithms: $1+\lambda$-*ES*: $1+\lambda$ Evolution Strategy (Auger, 2005), *A3C*: Asynchronous Advantage Actor-Critic (Mnih et al., 2016), *ARA**: Anytime A* (Likhachev et al., 2003), *C-REPS*: Contextual Relative Entropy Policy Search (Kupcsik et al., 2014), *C4.5*: C4.5 Classifier (Quinlan, 1993), *CBiRRT*: Constrained Bi-directional Rapidly-Exploring Random Tree (Berenson et al., 2009), *CMA-ES*: Covariance Matrix Adaptation Evolution Strategy (Hansen and Kern, 2004), *ELS*: Elliptic Slice Sampling (Murray et al., 2010), *LP*: Linear Programming, *LSTD*: Least Squares Temporal Difference Learning (Boyan, 1999), *M5P*: M5P Regression Model Trees (Wang and Witten, 1997), *MCMC*:Markov Chain Monte Carlo (Andrieu et al., 2003), *MLP*: Multi-layer Perceptron (Bishop, 1995), *REPS*: Relative Entropy Policy Search (Peters et al., 2010), *RRT*: Rapidly-exploring Random Tree (LaValle and Kuffner, 1999), *SVM*: Ranking Support Vector Machine (Joachims, 2002; Herbrich et al., 1999), *TRPO*: Trust Region Policy Optimization (Schulman et al., 2015)

Requirements: *DA*: discrete action space, *DS*: discrete state space, *M*: transition model is known, *MA*: pre-generated samples required, *PP*: parametric policy, S_0: all states can be used as initial state

4. Experiment Domains

Most practical applications for PbRL focus on robot teaching tasks and on board game domains. These domains can be subject to complex reward shaping problems (Ng et al., 1999) as it is required to not only solve the task itself, but also consider constraints like motor limitations or obstacle avoidance. Furthermore, a robot acting in uncontrolled environments needs to cope with changing environments, interact with non expert users and adapt to new tasks. For such tasks, PbRL is a suitable tool as it allows robots to learn without expert knowledge or predefined rewards. In the following, we highlight a few domains where preference-based approaches have been successfully employed in the past.

4.1 Object Handover

Kupcsik et al. (2015) use a preference-based approach to learn a robot handover task where a robot has to pass a bottle to a person sitting, walking or running, as shown in Fig. 4. The handover should be performed as comfortably as possible for the human which approaches at different speed. Hence, the stiffness as well as the trajectory of the robot's end-effector need to be adapted by the learning procedure.

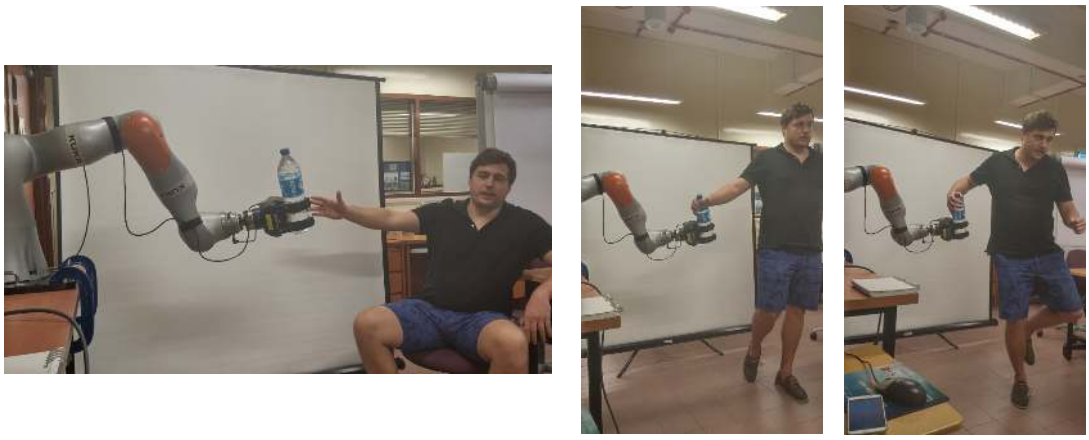


Figure 4: The robot handover task with a sitting, standing and running person

As described in Strabala et al. (2013), this task consists of three stages: approach, signal and transfer. First, the robot needs to approach the human and get ready for the actual handover. In the signaling phase, the robot and the human need to exchange information for determining the intended joint position for the handover. This information exchange is often non-verbal. Lastly, the joint contact needs to be established. This physical transfer should be dynamic and should not require a stationary contact phase. The focus of Kupcsik et al. (2015) is on this last phase. Experiments are performed with a real *7-DoF KUKA LBR arm* and a *Robotiq 3-finger hand* in a laboratory setup, as shown in Fig. 5. The robot performs several handover trials with a human receiving the object. Feedback is obtained in terms of trajectory preferences, but can be enhanced with an ordinal evaluation on a 1–10 scale. The trajectories are considered to be samples of a parametric policy and the related feedback is used to learn an evaluation function for policies, as described in Sec. 3.2.3 and 3.4.1. The evaluation function uses the policy parameters as input, allowing to define a

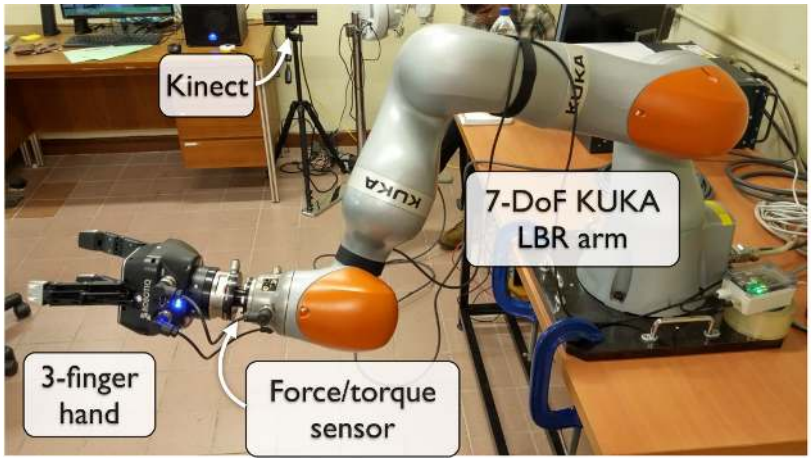


Figure 5: The robot KUKA robot with a 3-finger hand

likelihood function for the feedback, given a parametric policy. Using Bayesian inference and a Gaussian process prior, it is possible to infer the policy evaluation function explicitly. Policies can be evaluated without obtaining trajectory samples by directly evaluating the policy parameters. Hence, no costly environment calls are needed for optimizing the policy.

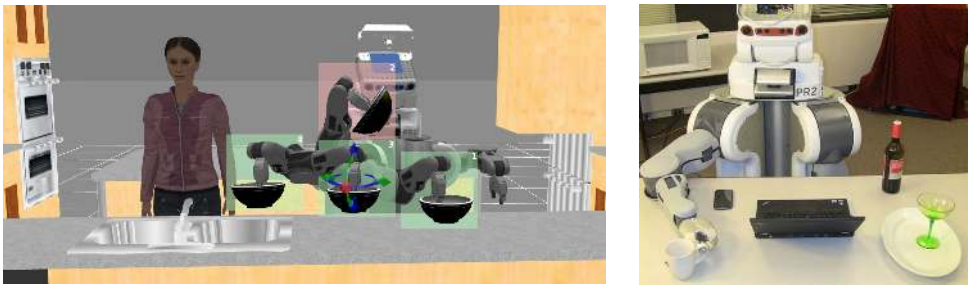
The performed evaluation shows a steady improvement, averaged over 5 trials. Furthermore, the return of the policy approaches a plateau after 80 policy updates. However, the exact amount of used preferences as well as the performance difference to an optimal policy remain unclear.

In general, the approach is very interesting for domains where (low-dimensional) parametric policies are available and trajectory generation is expensive. Only learning the utility function requires trajectory sampling. Furthermore, the process can probably be improved further by considering specific preference function exploration criteria, as introduced in Sec. 3.4.3. However, the approach is most likely not suited for high-dimensional policy spaces because of the Gaussian process. Gaussian process-based methods are computational expensive and require a high number of samples to sufficiently capture complex policies. Hence, a high number of trajectory samples and preferences would be required.

4.2 Path Planning

Jain et al. (2013, 2015) also tries to solve robotic tasks, but use a simulated setup, shown in Fig. 6a. Due to the simulated setup, the authors were able to evaluate their system on a wide range of path planning tasks, namely:

- pick and place, e.g., moving a glass of water without spilling water;
- inserting, e.g., placing or knife inside a holder;
- pouring, e.g., pouring water into a cup;
- heavy-fragile movement, e.g., moving a heavy object with a fragile object close by;
- liquid-electronic movement, e.g., moving a glass of water with a laptop in vicinity;



(a) The simulation environment (b) The real robot

Figure 6: The experiment setup of Jain et al. (2013, 2015)

- sharp-object movement, e.g., moving a knife with a human in range;
- hot-object movement, e.g., serving a hot drink.

Overall, for evaluating their approach, they defined 35 tasks by changing the objects and/or environment. The robot demonstrates a set of trajectories, which are ranked by the algorithm. The user can then define preferences by re-ranking them or by supplying new trajectories, which are assumed to be an improvement over all trajectories demonstrated by the robot. Hence, they are added to the ranking as the top-ranked element. Preferences are then obtained by computing all pairwise preferences contained in the ranking.

For evaluating the approach, an expert scored 2100 trajectories in total, based on a 1–5 scale. From these scores, the authors derive a ranking and compare it with a ranking based on the learned utility. They achieve an nDCG@1(nDCG@3) ranking error (Manning et al., 2008) of 0.78(0.66) to 0.93(0.92) after obtaining feedback 20 times. The ranking error is averaged over multiple tasks, grouped into three categories. The used feedback is only a re-ranking of 5 obtained trajectories by picking the most preferred one. The exact amount of derived preferences is unclear.

The work of Jain et al. (2013, 2015) is especially interesting because it was demonstrated to be applicable in a wide range of domains. In contrast to Kupcsik et al. (2015), a low-dimensional policy space is not required, but a predefined trajectory feature space. Arguably, this requirement is easier to satisfy and apply. However, the utility function has to be linear in the given feature space but the method could be combined with non-linear utility functions, as introduced by Wirth et al. (2016).

4.3 Chess Agent

An application of PbRL to board games was demonstrated by Wirth and Fürnkranz (2012, 2015). They use already available data from the chess domain to learn an artificial chess agent. Professional chess games are commonly stored in large databases, where many games annotate move and position with a unified symbol set, as shown in Fig. 7. The annotation symbols define the quality of a position or move, e.g., the white/black player has a decisive/moderate/slight advantage over her opponent[3]. In the given example, the annotator

3. http://portablegamenotation.com/Symbols.html

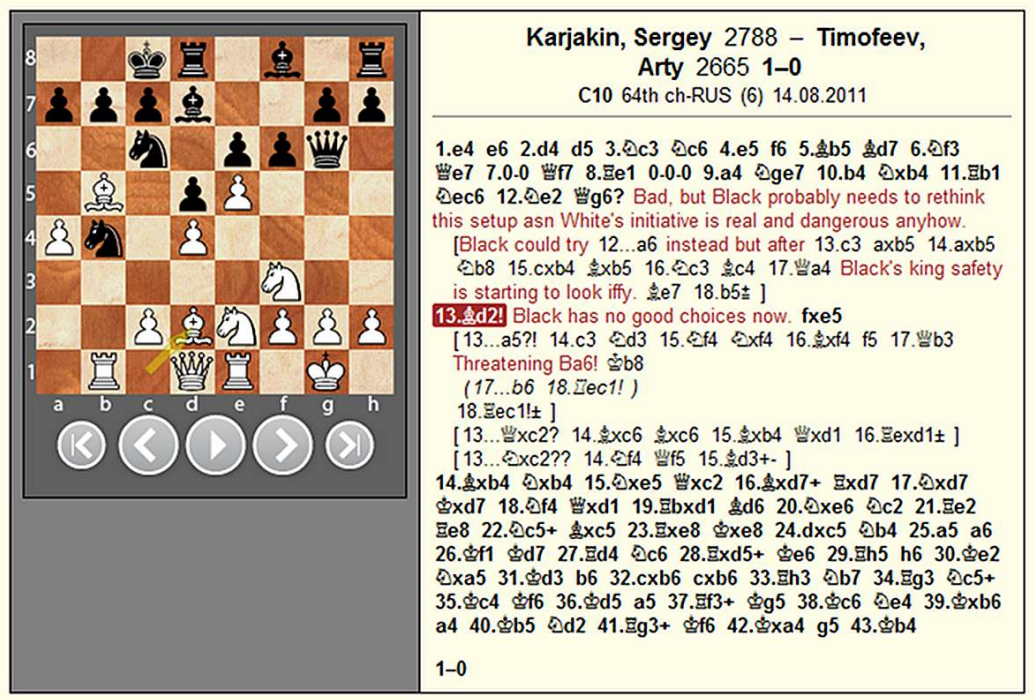


Figure 7: A chess position with move history and annotations.

remarked that White's move 13.♗d2! was a very good move (indicated by the exclamation mark at the end of the move), and provides three alternative replies for the move 13...f×e5 that was played in the game: 13...a5?! is dubious, 13...♕×c2? is a mistake, and 13...♘×c2?? is a losing move. The positions at the end of these variations are also annotated with symbols like ±, indicating a small advantage for White, or +− indicating a decisive advantage for White. It is straight-forward to derive preferences over moves and positions from such annotations. The preferences can then be used to learn a state utility function which estimates the expected outcome of a position. A policy is then derived by maximizing the utility, using the already known chess transition function.

The results in this domain show that it is possible to approach the performance of a near-optimal handcrafted utility function. However, the best results require 2.12M preferences and use a pre-defined utility feature space with only 15 dimensions.

It can be concluded that this approach is interesting for domains where it is easily possible to derive high number of (noisy) preferences. However, if optimal policies are of importance, it should probably only be used for computing an initial solution that can be further refined by applying one of the other PbRL approaches.

4.4 Other Domains

Nearly all other approaches are evaluated on classical reinforcement learning toy domains, where qualitative preferences are derived from the quantitative returns of the trajectories. Commonly used domains are the *mountain car* (Sutton and Barto, 1998), *pendulum swing-up* (Åström and Furuta, 2000), *cart-pole balancing* (Wang et al., 1996), *acrobot* (Spong, 1994),

and the *bicycle balance* (Randløv and Alstrøm, 1998) domains. Two approaches also use a simple grid world defined by Akrour et al. (2014). This *grid world* is easy to solve from a reinforcement learning point of view, but difficult for preference learning as the ground-truth reward values are very similar for some states. Typical state-of-the-art approaches require $5 - 40$ preferences to compute an optimal policy in these domains. Several algorithms are also evaluated on a *cancer treatment* task, derived from a reward based setting by Zhao et al. (2009). More complex toy domains have been used by Christiano et al. (2017), as they employ the ATARI learning environment (Bellemare et al., 2013) and the MuJoCo framework (Todorov et al., 2012) for learning movement and other robotic tasks. Interestingly, they have been able to outperform state-of-the-art RL approaches with numeric rewards in several domains, however they required several hundred preferences.

Runarsson and Lucas (2012, 2014) use the board game Othello, in a setting that is similar to the experiments by Wirth and Fürnkranz (2012, 2015). However, preferences are not derived from human annotations, but by assuming that observed state/action choices are preferable to unseen alternatives. This relates the approach closely to inverse reinforcement learning, although, the system directly learns a value function instead of a reward function. Sugiyama et al. (2012) derive preferences from numeric ratings and also learn a value function, however, in a human dialog setting. They show improvements over classic RL and IRL approaches. A detailed overview of experiment domains and evaluated approaches is given in Tbl. 3. Domain setups are not always identical and may differ in details such as the noise level, used feature space or action limits. Hence, even in the few cases where two papers do publish results of the same evaluation domain, the results are usually not directly comparable.

5. Discussion

While there has been much progress in the recent years in PbRL, many open questions and shortcomings of the approaches remain. For example, none of the utility-based approaches are able to produce a set of Pareto-optimal policies in case of incomparabilities. Only the *direct policy learning* method of Busa-Fekete et al. (2014) can handle incomparabilities by assuming that the incomparable trajectories should be evaluated equally, i.e., that both trajectories receive the same probability. A potential solution to cope with incomparabilities would be to learn a non-scalar utility function, allowing the utility functions to differ in multiple dimensions. A set of Pareto-optimal policies could then be learned using *multi-objective reinforcement learning* (Liu et al., 2015).

The question of preference-space exploration has also not yet been addressed satisfactorily. While a few approaches for preference-space exploration have been proposed (Sec. 3.4.3), a principled integration into the policy-space exploration strategy is still missing. Both tasks should be combined efficiently to reduce the amount of trajectory samples and preferences that are required. In addition, very few utility-based approaches are able to learn non-linear utility functions. Gritsenko and Berenson (2014) learn non-linear utilities but require that the PbRL problem is turned into an ordinal ranking problem first (Sec. 3.2.3). Ordinal rankings are problematic in that arbitrary distances between ranks are introduced when they are treated as regression problems, whereas the classification case suffers from a severe loss of information. Wirth et al. (2016); Kupcsik et al. (2015) employ

	classic discrete			classic continuous					robotics						games				
	End of Maze	**Grid world**	**River Swim**	**Acrobot**	**Bicycle Balance**	**Inverted Pendulum**	**Mountain Car**	**Swing Up**	**MuJoCo**	**NAO Robot**	**path planning**	**unhang wheel**	**terrain locomotion**	**object handover**	**ATARI**	**Chess**	**Othello**	**Cancer**	**Dialog Data**
Akrour et al. (2011)	✗		✗																
Akrour et al. (2012)							✗											✗	
Akrour et al. (2014)		✗			✗	✗				✗									
Busa-Fekete et al. (2013, 2014)																		✗	
Christiano et al. (2017)									✗						✗				
Fürnkranz et al. (2012)						✗	✗											✗	
Gritsenko and Berenson (2014)												✗							
Jain et al. (2013, 2015)											✗								
Kupcsik et al. (2015)														✗					
Runarsson and Lucas (2012, 2014)																	✗		
Sugiyama et al. (2012)																			✗
Wilson et al. (2012)				✗	✗	✗	✗												
Wirth and Fürnkranz (2013a,b)				✗	✗	✗													
Wirth and Fürnkranz (2012, 2015)																✗			
Wirth et al. (2016)		✗		✗	✗			✗											
Zucker et al. (2010)													✗						

Table 3: Overview of evaluation domains

kernels for overcoming this problem, an idea applicable to all linear utility function learning problems. However, this results in a high-dimensional utility space and is computational costly. Hence, the approaches do not scale well to high dimensional spaces. The deep neural network based approach by Christiano et al. (2017) overcomes the scalability issues of kernel-based methods but requires a high amount of preferences. Further research is required to either scale preference-based learning with kernels to larger problems spaces or to improve the results from DNN-based approaches with low amounts of training data.

Another important research question is to find principled ways for combining various types of feedback from the human expert, such as preferences, ordinal or numeric feedback, and demonstrations. In particular, inverse reinforcement learning from demonstrated trajectories could be used to provide good initializations for learning a utility function. This utility could then be refined by preferences of the experts.

Furthermore, PbRL can currently not be used to perform risk-averse optimization (Coraluppi and Marcus, 1999; Geibel, 2001). Risk-adverse optimization guarantees that

the worst case performance of the algorithm is bounded. In many cases, it is required that certain trajectories are not realized, for example, trajectories that would damage the robot or its environment. Such risk-adverse behavior could be realized with weighted preference loss functions such as (3), i.e., by substantially increasing the weights of preferences that concern risky trajectories.

Another important open issue, which would substantially extend the range of possible applications of PbRL, is to develop techniques that are able to use preferences between trajectories starting from different initial states. A potential solution could be to take the expected value of initial states into account and compute the preference based on the advantage function.

As mentioned in Secs. 3.4 and 3.5, most approaches are lacking in terms of sample efficiency. While sample efficiency could already be improved by using reward-based utilities (see Sec. 3.2.3) which allow sample reuse for policy optimization (Wirth et al., 2016), more efficient model-based policy search strategies such as the PILCO algorithm (Deisenroth and Rasmussen, 2011) or guided policy search (Levine and Koltun, 2013) could be employed that learn a system dynamics model from the data and employ it for policy optimization. Moreover, we have seen that the complexity of the tasks that have been used for learning is limited. More complex policies, which, e.g., can directly use sensory feedback as inputs, could be acquired by incorporating recent successes from deep reinforcement learning (Mnih et al., 2015, 2016).

Furthermore, PbRL is still lacking a unified evaluation framework. The different publications use a wide variety of testing domains, usually derived from classic RL problems. However, the specific setups depend on the publication and usually modified the basic configuration by adding noise or removing the terminal conditions. Moreover, many algorithms disregard specific subproblems like policy generalization or utility exploration. Hence, a satisfactory comparison of algorithms is currently not possible.

6. Conclusion

Preference-based reinforcement learning (PbRL) is a suitable tool for learning from qualitative, non-numeric rewards. On the one hand, it can provide solutions in domains where numeric feedback is not readily available, and, on the other hand, it may reduce the demands and prior knowledge that is needed for applying RL to real-world tasks where a numeric feedback signal has to be shaped. This survey provided a first overview over the design choices that have to be considered for PbRL and presented current work in a coherent framework, shedding light on the advantages and limitations of various approaches.

The most substantial problem seems to be the complexity of the learning problem. The presented approaches require a high amount of preferences or well defined feature spaces, either on a policy or trajectory level. The definition of suitable, low-dimensional feature spaces requires expert knowledge, if possible at all. In many domains, high-dimensional feature spaces are required to approximate optimal policies to an acceptable degree. Furthermore, preferences are costly to obtain in an incremental setting with human evaluations. Even in cases where it is possible to obtain preferences from non-expert users, human labor is usually quite expensive. Hence, it is very important to use the available information as efficiently as possible. Utility-based approaches allow to generalize the obtained preference

feedback and are an important step towards practical algorithms. Especially reward-based utilities seem to be appropriate for domains without well defined policy spaces, because they allow the application of efficient value-based reinforcement learning methods. Hence, it is possible to use the substantial progress that has already been achieved in this field. However, it is also important to consider the novel problems introduced by using preference-based feedback. Foremost, efficient methods for exploring the preference function space are required.

Recently, interesting real-world applications could be solved by preference-based methods. Yet, several open questions remain. Foremost, sample efficiency and scalibility issues still prevent applications to more complex tasks. DNN-based approaches can tackle the scalibility issues to some degree, but improved solutions to the preference-space exploration problem are required to limited the amount of required preferences. Furthermore, PbRL is still limited by several assumptions like the total order assumption, the uniform preference weighting and the single starting state. Without these assumptions, it would be possible to apply PbRL in multi-objective domains, for risk-adverse problems and in settings that can not be simulated. Methods that can use multiple feedback forms, like preferences, demonstrations or rewards, are also of interested as they are possible able to reduce the cognitive demand even further.

We believe that this line of research could provide the foundation to make reinforcement learning applicable to the non-scientific community because despite many recent successes in reinforcement learning, we believe that the required expertise in machine learning is still a key limitation when it comes to applying RL to real-world tasks.

Acknowledgments

This work was supported by the German Research Foundation (DFG) as part of the Priority Programme 1527 "Autonomous Learning", the research project FU 580/10, and under the EU H2020 project 'RoMaNS', Ref. 645582. We would like to thank Eyke Hüllermeier, Robert Busa-Fekete and Stefan Riezler for interesting and stimulating discussions that greatly influenced this paper.

References

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine learning (ICML-04)*, volume 69. 2004.

A. Agarwal, O. Dekel, and L. Xiao. Optimal algorithms for online convex optimization with multi-point bandit feedback. In *Proceedings of the 23rd Conference on Learning Theory (COLT-10)*, pages 28–40, 2010.

R. Akrour, M. Schoenauer, and M. Sebag. Preference-based policy learning. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-11)*, volume 6911 of *LNCS*, pages 12–27. 2011.

R. Akrour, M. Schoenauer, and M. Sebag. APRIL: Active preference learning-based reinforcement learning. In *Proceedings of the European Conference on Machine Learning*

and Knowledge Discovery in Databases (ECML-PKDD-12), volume 7524 of *LNCS*, pages 116–131. 2012.

R. Akrour, M. Schoenauer, and M. Sebag. Interactive Robot Education. In *Proceedings of the ECML/PKDD Workshop on Reinforcement Learning with Generalized Feedback: Beyond Numeric Rewards*, 2013.

R. Akrour, M. Schoenauer, M. Sebag, and J.-C. Souplet. Programming by Feedback. In *Proceedings of the 31nd International Conference on Machine Learning (ICML-14)*, volume 32, pages 1503–1511. 2014.

D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016.

C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1-2):5–43, 2003.

K. J. Åström and K. Furuta. Swinging up a pendulum by energy control. *Automatica*, 36 (2):287–295, 2000.

P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. Gambling in a rigged casino: The adversarial multi-arm bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS-95)*, pages 322–331. 1995.

A. Auger. Convergence Results for the (1,λ)-SA-ES using the Theory of φ-irreducible Markov Chains. *Theoretical Computer Science*, 334(1-3):35–69, 2005.

A. G. Barto. Intrinsic motivation and reinforcement learning. In G. Baldassarre and M. Mirolli, editors, *Intrinsically Motivated Learning in Natural and Artificial Systems*, pages 17–47. Springer Berlin Heidelberg, 2013.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47 (1):253–279, 2013.

D. Berenson, S. Srinivasa, D. Ferguson, and J. Kuffner, Jr. Manipulation planning on constraint manifolds. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-09)*, pages 625–632, 2009.

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

J. Boyan. Least-Squares Temporal Difference Learning. In *Proceedings of the 16th International Conference on Machine Learning (ICML-99)*, pages 49–56. 1999.

R. Busa-Fekete and E. Hüllermeier. A survey of preference-based online learning with bandit algorithms. In *Proceedings of the 25th Algorithmic Learning Theory International Conference (ALT-14)*, volume 8776 of *LNCS*, pages 18–39. 2014.

R. Busa-Fekete, B. Szörényi, P. Weng, W. Cheng, and E. Hüllermeier. Preference-based evolutionary direct policy search. In *Proceedings of the ICRA Workshop on Autonomous Learning*, 2013.

R. Busa-Fekete, B. Szörényi, P. Weng, W. Cheng, and E. Hüllermeier. Preference-based reinforcement learning: evolutionary direct policy search using a preference-based racing algorithm. *Machine Learning*, 97(3):327–351, 2014.

P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *CoRR*, abs/1706.03741, 2017.

W. Chu and Z. Ghahramani. Preference learning with Gaussian processes. In *Proceedings of the 22nd International Conference on Machine Learning (ICML-05)*, pages 137–144. ACM, 2005.

S. P. Coraluppi and S. I. Marcus. Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes. *Automatica*, 35(2):301 – 309, 1999.

C. Daniel, M. Viering, J. Metz, O. Kroemer, and J. Peters. Active reward learning. In *Proceedings of Robotics: Science & Systems X (R:SS-14)*, 2014.

M. P. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 465–472. 2011.

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

J. Duchi, L. W. Mackey, and M. I. Jordan. On the consistency of ranking algorithms. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 327–334, 2010.

R. O. Duda and P. E. Hart. *Pattern Recognition and Scene Analysis*. John Wiley and Sons, 1973.

A. Faust, H.-T. Chiang, N. Rackley, and L. Tapia. Avoiding moving obstacles with stochastic hybrid dynamics using PEARL: Preference appraisal reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-16)*, pages 484–490. 2016.

M. Friedman and L. J. Savage. The expected-utility hypothesis and the measurability of utility. *Journal of Political Economy*, 60, 1952.

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

J. Fürnkranz, E. Hüllermeier, W. Cheng, and S.-H. Park. Preference-based reinforcement learning: A formal framework and a policy iteration algorithm. *Machine Learning*, 89 (1-2):123–156, 2012. Special Issue of Selected Papers from ECML/PKDD-11.

P. Geibel. Reinforcement learning with bounded risk. In *In Proceedings of the 18th International Conference on Machine Learning (ICML-01)*, pages 162–169. 2001.

S. Griffith, K. Subramanian, J. Scholz, C. Isbell, and A. L. Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Advances in Neural Information Processing Systems 26 (NIPS-13)*, pages 2625–2633, 2013.

A. Gritsenko and D. Berenson. Learning cost functions for motion planning from human preferences. In *Proceedings of the IROS 2014 Workshop on Machine Learning in Planning and Control of Robot Motion*, volume 1, pages 48–6, 2014.

S. Grünewälder, G. Lever, L. Baldassarre, M. Pontil, and A. Gretton. Modelling transition dynamics in MDPs with RKHS embeddings. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 535–542, 2012.

N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN-04)*, volume 3242 of *LNCS*, pages 282–291. 2004.

V. Heidrich-Meisner and C. Igel. Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. In *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, pages 401–408. 2009.

R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression. In *Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN-99)*, volume 1, pages 97–102, 1999.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

E. Hüllermeier, J. Fürnkranz, W. Cheng, and K. Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16–17):1897–1916, 2008.

A. Jain, B. Wojcik, T. Joachims, and A. Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *Advances in Neural Information Processing Systems 26 (NIPS-13)*, pages 575–583, 2013.

A. Jain, S. Sharma, T. Joachims, and A. Saxena. Learning preferences for manipulation tasks from online coactive feedback. *International Journal of Robotics Research*, 34(10): 1296–1313, 2015.

T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 133–142. 2002.

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

W. B. Knox and P. Stone. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP-09)*, pages 9–16, 2009.

W. B. Knox and P. Stone. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)*, pages 5–12. 2010.

W. B. Knox and P. Stone. Reinforcement learning from simultaneous human and MDP reward. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-12)*, pages 475–482. 2012.

J. Kober, J. A. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

D. Kreps. *Notes on the theory of choice*. Underground classics in economics. Westview Press, 1988.

A. Kupcsik, M. P. Deisenroth, J. Peters, A. P. Loh, P. Vadakkepat, and G. Neumann. Model-based contextual policy search for data-efficient generalization of robot skills. *Artificial Intelligence*, 247:415–439, 2014.

A. Kupcsik, D. Hsu, and W. S. Lee. Learning dynamic robot-to-human object handover from human feedback. In *Proceedings of the 17th International Symposium on Robotics Research (ISRR-15)*, 2015.

M. G. Lagoudakis and R. Parr. Least-Squares Policy Iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

S. M. LaValle and J. Kuffner, Jr. Randomized kinodynamic planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-99)*, pages 473–479, 1999.

S. Levine and V. Koltun. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1–9, 2013.

M. Likhachev, G. Gordon, and S. Thrun. ARA*: Anytime A* search with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems 16 (NIPS-03)*, pages 767–774. 2003.

C. Liu, X. Xu, and D. Hu. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398, 2015.

C. X. Lu, Z. Y. Sun, Z. Z. Shi, and B. X. Cao. Using emotions as intrinsic motivation to accelerate classic reinforcement learning. In *Proceedings of the International Conference on Information System and Artificial Intelligence (ISAI-16)*, pages 332–337, 2016.

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

C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

B. Mayeur, R. Akrour, and M. Sebag. Direct value learning: a rank-invariant approach to reinforcement learning. In *Proceedings of the NIPS Workshop on Autonomously Learning Robots*, 2014.

D. Meunier, Y. Deguchi, R. Akrour, E. Suzuki, M. Schoenauer, and M. Sebag. Direct value learning: A preference-based approach to reinforcement learning. In *Proceedings of the ECAI Workshop on Preference Learning: Problems and Applications in AI*, pages 42–47, 2012.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML-16)*, pages 1928–1937, 2016.

I. Murray, R. P. Adams, and D. J. C. MacKay. Elliptical slice sampling. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS-10)*, volume 9, pages 541–548. 2010.

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

A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning (ICML-99)*, pages 278–287, 1999.

D. Nguyen-Tuong and J. Peters. Model learning for robot control: a survey. *Cognitive Processing*, 12(4):319–340, 2011.

J. Peters, K. Muelling, and Y. Altun. Relative entropy policy search. In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI-10)*, pages 1607–1612. 2010.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning (ICML-98)*, pages 463–471, 1998.

C. A. Rothkopf and C. Dimitrakakis. Preference elicitation and inverse reinforcement learning. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-11), Part III*, volume 6913 of *Lecture Notes in Computer Science*, pages 34–48, 2011.

T. P. Runarsson and S. M. Lucas. Preference learning for move prediction and evaluation function approximation in othello. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):300–313, 2014.

T. P. Runarsson and S. M. Lucas. Imitating play from game trajectories: Temporal difference learning versus preference learning. In *IEEE Conference on Computational Intelligence and Games (CIG-12)*, pages 79–82. IEEE, 2012.

J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

O. Shamir. An optimal algorithm for bandit and zero-order convex optimization with two-point feedback. *Journal of Machine Learning Research*, 18:1–11, 2017.

S. P. Singh, A. G. Barto, and N. Chentanez. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems 17 (NIPS-04)*, volume 17, pages 1281–1288, 2004.

S. P. Singh, R. L. Lewis, and A. G. Barto. Where do rewards come from? In *Proceedings of the 31th Conference of the Cognitive Science Society (CogSci-09)*, pages 2601–2606, 2009.

A. Sokolov, J. Kreutzer, C. Lo, and S. Riezler. Learning structured predictors from bandit feedback for interactive NLP. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL-16)*, 2016.

M. W. Spong. Swing up control of the acrobot. In *IEEE International Conference on Robotics and Automation (ICRA-94)*, pages 2356–2361. IEEE, 1994.

K. Strabala, M. K. Lee, A. Dragan, J. Forlizzi, S. Srinivasa, M. Cakmak, and V. Micelli. Towards seamless human-robot handovers. *Journal of Human-Robot Interaction*, 2013.

H. Sugiyama, T. Meguro, and Y. Minami. Preference-learning based inverse reinforcement learning for dialog control. In *Proceedings of the 13th Annual Conference of the International Speech Communication Association (INTERSPEECH-12)*, pages 222–225, 2012.

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

G. Tesauro. Temporal Difference Learning and TD-gammon. *Communications ACM*, 38(3): 58–68, 1995.

A. L. Thomaz and C. Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI-06)*, pages 1000–1006. 2006.

E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-12)*, pages 5026–5033, 2012.

L. Torrey and M. E. Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-13)*, pages 1053–1060. 2013.

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

J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

H. O. Wang, K. Tanaka, and M. F. Griffin. An approach to fuzzy control of nonlinear systems: stability and design issues. *IEEE Transactions on Fuzzy Systems*, 4(1):14–23, 1996.

Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *Poster papers of the 9th European Conference on Machine Learning (ECML-97)*. 1997.

Y.-C. Wang and J. M. Usher. Application of reinforcement learning for agent-based production scheduling. *Engineering Applications of Artificial Intelligence*, 18(1):73–82, 2005.

P. Weng. Markov decision processes with ordinal rewards: Reference point-based preferences. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-11)*, 2011.

A. Wilson, A. Fern, and P. Tadepalli. A bayesian approach for policy learning from trajectory preference queries. In *Advances in Neural Information Processing Systems 25 (NIPS-12)*, pages 1142–1150. 2012.

C. Wirth and J. Fürnkranz. First steps towards learning from game annotations. In *Proceedings of the ECAI Workshop on Preference Learning: Problems and Applications in AI*, pages 53–58, 2012.

C. Wirth and J. Fürnkranz. A policy iteration algorithm for learning from preference-based feedback. In *Advances in Intelligent Data Analysis XII: 12th International Symposium (IDA-13)*, volume 8207 of *LNCS*, pages 427–437. 2013a.

C. Wirth and J. Fürnkranz. EPMC: Every visit preference Monte Carlo for reinforcement learning. In *Proceedings of the 5th Asian Conference on Machine Learning, (ACML-13)*, volume 29 of *JMLR Proceedings*, pages 483–497. 2013b.

C. Wirth and J. Fürnkranz. Preference-based reinforcement learning: A preliminary survey. In *Proceedings of the ECML/PKDD-13 Workshop on Reinforcement Learning from Generalized Feedback: Beyond Numeric Rewards*, 2013c.

C. Wirth and J. Fürnkranz. On learning from game annotations. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(3):304–316, 2015.

C. Wirth, J. Fürnkranz, and G. Neumann. Model-free preference-based reinforcement learning. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 2222–2228, 2016.

Y. Yue, J. Broder, R. Kleinberg, and T. Joachims. The k-armed dueling bandits problem. *Journal of Computer System Sciences*, 78(5):1538–1556, 2012.

Y. Zhao, M. Kosorok, and D. Zeng. Reinforcement learning design for cancer clinical trials. *Statistics in Medicine*, 28(26):3294–3315, 2009.

Y. Zhao, D. Zeng, M. A. Socinski, and M. Kosorok. Reinforcement learning strategies for clinical trials in nonsmall cell lung cancer. *Biometrics*, 67(4):1422–1433, 2011.

S. Zhifei and E. Meng Joo. A survey of inverse reinforcement learning techniques. *International Journal of Intelligent Computing and Cybernetics*, 5(3):293–311, 2012.

B. D. Ziebart, A. Maas, J. A. Bagnell, and A. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 1433–1438. 2008.

M. Zucker, J. A. Bagnell, C. Atkeson, and J. Kuffner, Jr. An optimization approach to rough terrain locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-10)*, pages 3589–3595. 2010.