Efficient Preference-based Reinforcement Learning

Using Surrogates for Solving Markov Decision Processes with Preferences

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades
Doctor rerum naturalis
(Dr. rer. nat.)

vorgelegt von

Christian Wirth, M.Sc.

geboren in Frankfurt

Tag der Einreichung: 27.04.2017
Tag der Disputation: 21.06.2017

Referenten:	Prof. Dr. Johannes Fürnkranz, TU Darmstadt, Germany
	Prof. Dr. Gerhard Neumann, University of Lincoln, UK

Darmstadt 2017
D17

Acknowledgments

Foremost, I would like to thank my parents, Marianne and Nikolaus Wirth, for supporting me on my way to become a PhD candidate. Without their backing, it would not have been possible to complete my studies.

Second, my gratitude goes to Prof. Dr. Johannes Fürnkranz for offering me the opportunity to obtain a doctorate and to conduct research in a friendly and productive environment. His guidance, suggestions and feedback proved invaluable. The same gratitude I owe to my second supervisor, Prof. Dr. Gerhard Neumann, who especially improved my understanding of various mathematical problems. Furthermore, I want to thank Dr. Riad Akrour, Prof. Dr. Eyke Hüllermeier, Dr. Robert Busa-Fekete and my colleagues at the KE department for the insightful discussions and feedback.

On a personal level, I want to especially thank my wonderful girlfriend Christina Meisl. She had my back in times of stress and insecurity. She also proved substantial to this dissertation by providing me with insight that is not restricted to a narrow, technical view.

Thanks go also to Gabirele Ploch for ensuring that everything runs smoothly in our department, to the DFG and the TU Darmstadt for providing financing and to the Lichtenberg-HLR for providing computational resources.

Abstract

Common reinforcement learning algorithms assume access to a numeric feedback signal. The numeric feedback contains a high amount of information and can be maximized efficiently. However, the definition of a numeric feedback signal can be difficult in practise due to several limitations and badly defined values may lead to an unintended outcome. For humans, it is usually easier to define qualitative feedback signals than quantitative. Hence, we want to solve reinforcement learning problems with a qualitative signal, potentially capable of overcoming several of the limitations of numeric feedback. Preferences have several advantages over other qualitative settings, like ordinal feedback or advice. Preferences are scale-free and do not require assumptions over the optimal outcome. However, preferences are difficult to use for solving sequential decision problems, because it is unknown which decisions are responsible for the observed preference. Hence, we analyze different approaches for learning from preferences and show the design principles that can be used, as well as the advantages and problems that occur. We also survey the field of preference-based reinforcement learning and categorize the algorithms according to the design principles. Efficiency is of special interest in this setting, as it is important to keep the amount of required preferences low, because they depend on human evaluation. Hence, our focus is on efficient use of the preferences. It can be stated that it is important to be able to generalize the obtained preferences, as this keeps the amount of required preferences low. Therefore, we consider methods that are able to generalize the obtained preferences to models not yet evaluated. However, this introduces uncertain feedback and the exploration/exploitation problem already known from classical reinforcement learning has to be considered with the preferences in mind. We show how to efficiently solve this dual exploration problem by interleaving both tasks, in an undirected manner. We use undirected exploration methods, because they scale better to high-dimensional spaces. Furthermore, human feedback has to be assumed to be error-prone and we analyze the problems that arise when using human evaluation. We show that noise is the most substantial problem when dealing with human preferences and present a solution to this problem.

Zusammenfassung

Klassische Algoritmen des verstärkenden Lernens nehmen an dass numerische Bewertungen existieren. Eine numerische Bewertung hat einen hohen Informationsgehalt und kann effizient maximiert werden. Die Definition solche eines numerischen Signals kann aber, auf Grund diverser Limitationen, in der Praxis schwierig sein. Zudem können schlecht definierte Werte zu unerwünschten Ergebnissen führen. Menschen können normalerweise einfacher qualitative als quantitative Bewertungen definieren. Daher wollen wir Probleme des verstärkenden Lernens mit qualitativen Bewertungen lösen um möglicherweise mehrere der Limitationen der numerischen Bewertungen zu überwinden. Präferenzen haben mehrere Vorteile über andere qualitative Bewertungen wie ordinale Signale oder Ratschläge. Präferenzen sind skalenfrei und benötigen keine Annahmen über das optimale Ergebnis. Präferenzen sind aber schwierig zu nutzen um sequentielle Entscheidungsprobleme zu lösen da unbekannt ist welche Entscheidungen zu einer bestimmen Präferenzen geführt haben. Daher analysieren wir verschieden Ansätze die das Lernen von Präferenzen ermöglichen und beschreiben die jeweiligen Design Entscheidungen sowie deren Vorteile und Nachteile. Zudem geben wir ein Überblick über das Forschungsfeld und Kategorisieren die entsprechenden Algorithmen anhand der getroffenen Design Entscheidungen. Effizienz ist in diesem Feld besonders wichtig um die Anzahl der nötigen Präferenzen zu reduzieren, da diese von Beurteilungen durch Menschen abhängen. Daher liegt unser Fokus auf der effizienten Nutzung von Präferenzen. Als besonders wichtig kann es angesehen werden vorhandene Präferenzen zu Generalisieren und Bewertungen zu erhalten, ohne das der Mensch explizit involviert werden muss. Diese Methoden erzeugen aber Unsicherheiten und das "Exploration/Exploitation"-Problem des klassischen, verstärkenden Lernens muss unter dem Aspekt der Präferenzen berücksichtigt werden. Wir zeigen wie dieses duale Erkundungsproblem effizient gelöst werden kann, in dem man beide Aufgaben mit Hilfe von ungerichteten Methoden vereint. Wir verwenden ungerichtete Erkundungsverfahren da diese besser mit der Dimensionalität des Problems skalieren. Des Weiteren muss angenommen werden dass menschliche Bewertungen fehlerhaft sind und wir analysieren die dadurch entstehenden Probleme. Wir zeigen dass es am wichtigsten ist das Problem des Rauschens zu lösen und zeigen eine entsprechende Lösung.

Contents

List of Abbreviations

AC-REPS	actor critic relative entropy policy search
API	approximate policy iteration with rollouts
CMA-ES	covariance matrix adaptation evolution strategy
DNN	deep neural network
EBC	expected belief change
EI	expected improvement
ESS	elliptical slice sampling
EPMC	every-visit preference Monte Carlo
EUS	expected utility of selection
FA	function approximation
FEN	Forsyth-Edwards Notation
GP	Gaussian process
IRL	inverse reinforcement learning
KL	Kullback-Leibler
LSPI	least squares policy iteration
LSTD	least-squares temporal difference learning
MCMC	Markov chain Monte Carlo
MDP	Markov decision process
MDPP	Markov decision process with preferences
NAG	numeric annotation glyph
PBPI	preference-based approximate policy iteration
PBRL	preference-based reinforcement learning
PF	programming by feedback
PGN	portable game notation
PL	preference learning
QBD	query by disagreement
REPS	relative entropy policy search

RL	reinforcement learning
SARSA	state-action-reward-state-action
SDP	sequential decision problem
SVM	support vector machine
UCB	upper confidence bounds

List of Symbols

å	annotation	107
$\mathscr{A}$	set of annotations	107
$\delta(s' \mid s, a)$	transition function	13
γ	discount factor	13
ρ	pairwise preference function	45
$\mathscr{X}$	object space	12
$\mathscr{Y}$	label space	39
$\mu(s)$	initial state distribution	13
$\pi(a \mid s)$	policy	13
$\Pr^{\pi}(\boldsymbol{\tau})$	trajectory distribution	14
$\boldsymbol{\tau}$	trajectory	13
$\boldsymbol{\tau}^{\dashv}$	partial trajectory, second part	87
$\boldsymbol{\tau}^{\vdash}$	partial trajectory, first part	87
$\boldsymbol{\Omega}$	policy parameter space	20
$\boldsymbol{\omega}$	policy parameter vector	20
$\boldsymbol{\phi}(x)$	features	12
$\boldsymbol{\Theta}$	function parameter space	23
$\boldsymbol{\theta}$	function parameter vector	23
$a \in A$	actions	13
R	return	13
$r(s, a, s')$	reward	13
$s \in S$	states	13
s_0	start state	13
U	utility function	40

List of Tables

List of Figures

Chapter I

Introduction

Human problem solving and decision making can be seen as an *optimization* technique. The human has some information about the problem at hand and tries to find an action that maximizes some *reward*. As an example, when searching for a new job, we may want to maximize our net income, based on information about possible employers. This information can include the types and number of positions available in a company, the companies' distance to the residence and possible secondary benefits like a work car. However, we are often not interested in maximizing the immediate *reward*, but a long-term *return*. In the job application example, this would relate to possibilities of promotions or a company retirement plan. Hence, we have to consider a *sequence of decisions*, as problems and opportunities that arise may influence the chance of promotions. Furthermore, we can not be certain about the outcome of an action, e.g., when applying for a position or promotion we may get rejected. We can only maximize an *expectation*. Additionally, when applying for a new job, we will not obtain a work car or the expected salary immediately, but later on, possibly influenced by further decisions in-between. Hence, the reward for applying for a new job can be temporally separated from the decision, it is *delayed*. A multitude of problems can be formalized as such an optimization task, which is subject to *expectations*, *delayed reward* and obtains *sequence* data. Navigation tasks require us to determine the optimal road on each intersection, based on the expectation of a traffic jam but the real travel time is only known on arrival. When playing games, we need to decide on a move at each turn, subject to the expectation of the opponent's move and delayed feedback when finishing the game. The outcome of treating a medical issue may only be known after multiple, different treatments, each related to an expected change.

Humans try to solve such sequence problems based on *experience*. They determine their chances for promotions by considering past promotion possibilities and by experience obtained from other humans in comparable situations. However, this experience must be obtained by interacting with the *environment* and observing the result. We may have an assumption about the outcome, but this may not be certain. Hence, we need to observe the real outcome to increase our certainty. However, it is usually not possible to obtain enough experience to become totally certain about a decision's outcome. Hence, we need to decide if we want to *exploit* our assumed, best action or to *explore* an alternative action to observe an outcome that reduces our uncertainty. As long as we are not certain about our actions, it is possible that our assumed, best action is suboptimal because our expectation is incorrect.

A general framework for maximizing the return in such a setting with an *agent* that obtained experience from interacting with an environment is *reinforcement learning* (RL). The basic idea is to *reinforce* decisions that have led to good outcomes according to the experience, by increasing the chance to perform them again. Classic RL assumes scalar, numeric rewards for evaluating decisions. This enables the computation of scalar return expectations, based on the expected outcomes of the actions we would take in each decision step. In most domains, the effect of an action is not known explicitly, and we have to approximate the expected outcome of a decision by using the observed experience. However, explicit computation of the action effects is usually problematic. After computing the expected return, we increase the selection probability for the actions with the highest, expected return. By changing the action selection probabilities in each step, we change the long-term expectation. Hence, we can only obtain an optimal solution by performing the optimization procedure multiple times.

I.1 Limitations of Numeric Feedback

The assumption of a numeric reward signal limits the applicability of reinforcement learning. The reward is not naturally given in most domains and must be defined manually. Usually, the expert has some unknown, internal utility scale for evaluating problems, but reinforcement learning requires this utility to be explicit and numerical. The translation to an explicit reward signal is error prone, due to multiple reasons. In the following, we will give four examples.

Reward Hacking

The agent may maximize the given reward, without performing the intended task (Amodei et al. 2016). As an 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. In case the reward was only defined for the completion of the task and not for intermediate steps, this solution maximizes the reward. Hence, the expert needs to define the rewards for fine-grained subtasks and re-evaluate the reward based on the obtained policy.

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). Reward shaping can alleviate the problem of reward hacking, but focuses on speeding-up learning. Figure I.1.1 shows the learning progress in a bicycle balance task (Lagoudakis and Parr 2003a). The goal is to keep a bicycle upright for 500 steps (see Section A.2), based on two different reward functions. The graphs shows the mean and quartiles of the accumulated reward obtained in each of 15 iterations of training an *actor critic relative entropy policy search* (AC-REPS) (cf. Section VI.3) RL learning system. The green graph shows learning with a reward of 1 for every step the bicycle is upright. This only defines the intended goal, as the accumulated reward directly relates to the number of time-steps the bicycle was kept upright. The blue graph uses the angle deviation from the upright position as a penalty, guiding the system to a vertically, stable position. We can

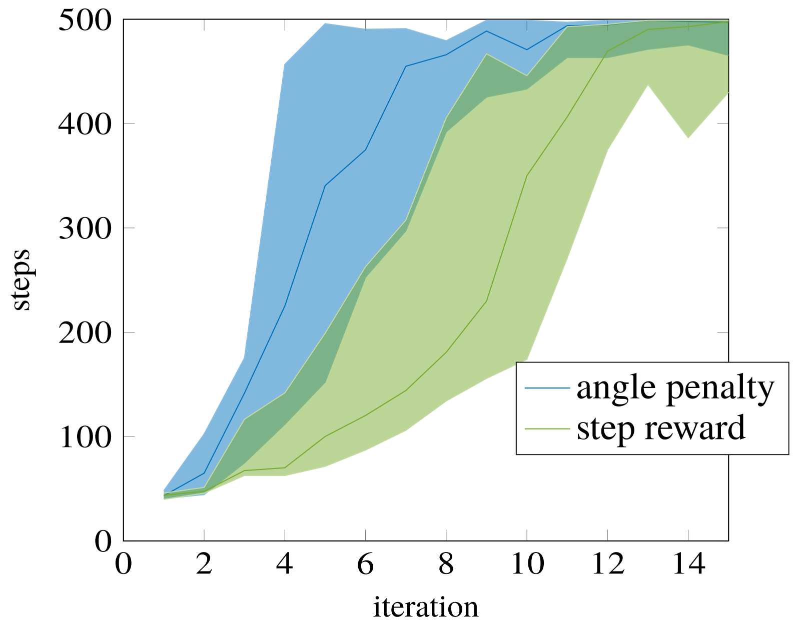


Figure I.1.1.: Two different reward functions for the bicycle balance task

see that the choice of reward function greatly influences the convergence. It seems the angle-based reward is better suited for the task, but it carries the risk of reward hacking. Throwing over the bike instantly will directly end the episode and inflict a penalty, but only for one time-step. This may be better than accumulating many smaller penalties over very long episodes. Striking a balance between the goal definition and the guidance task is often very difficult for the experimenter.

The reward shaping problem is also closely related to the 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 intrinsically 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. 2004; Singh et al. 2009). However, experts 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).

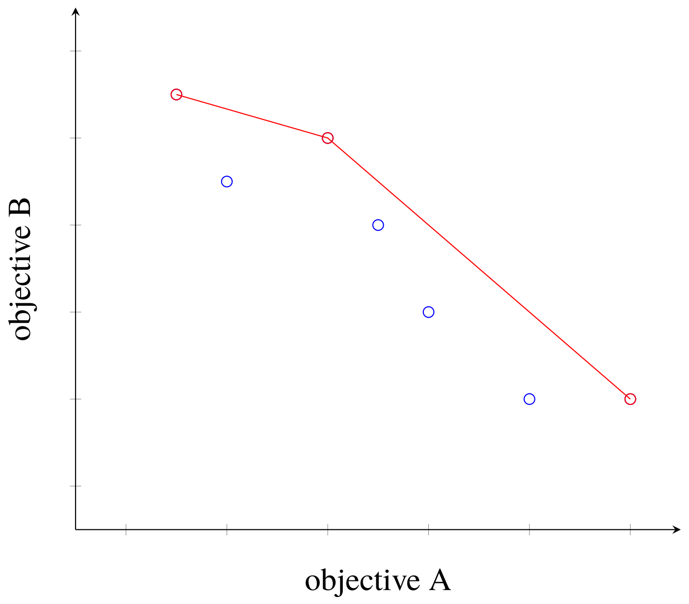


Figure I.1.2.: Multi-objective solutions with the Pareto front in red

Multi-objective tradeoffs

Many domains are subject to multiple reward signals, due to the mentioned reward shaping problem or because the solution needs to be evaluated by different criteria. By applying multi-objective reinforcement learning (Liu et al. 2015), we can obtain a set of Pareto-optimal policies, as shown in Figure I.1.2. However, we have still to pick one specific policy to realize, which implies a tradeoff between the different objectives. It is also possible to scalarize the reward beforehand and run classic reinforcement learning algorithms, but the scalarization also assumes a fixed tradeoff, which may not be known explicitly and is part of the learning problem.

Arbitrary reward values

Some applications require arbitrary-valued rewards, because the true reward values are unknown. For example, consider a recommender system that has only access to the feedback if a recommendation was viewed or not (Golovin and Rahm 2004). Hence, arbitrary values have to be selected for representing the binary outcome. Furthermore, some domains require infinitely-valued rewards, for instance, if certain events should be avoided at any cost. In the cancer treatment problem (Zhao et al. 2009), the death of a patient should be avoided at all times, and therefore give an infinitely low reward. However, an infinite reward breaks classic reinforcement learning algorithms as they depend on numeric methods and arbitrary, finite values have to be selected. However, such a value defines how many patients need to be cured to justify one dead patient.

I.2 Preferences

For many problems, human experts are able to demonstrate good judgment about the quality of certain courses of actions or solution attempts. A particularly well studied form of qualitative knowledge are so-called *pairwise comparisons* or *preferences*. Humans are often not able to determine a precise utility value of an option, but are typically able to compare the quality of two options (e.g., "Treatment **a** is more effective than treatment **b**"). Thurstone's *Law of Comparative Judgment* essentially states that such pairwise comparisons correspond to an internal, unknown utility scale (Thurstone 1927). Hence, such a qualitative approach is a promising replacement for numeric rewards in reinforcement learning, potentially able to overcome the problems stated in Section I.1

The recovery of a hidden utility scale from qualitative preferences is studied in various areas such as ranking theory (Marden 1995), social choice theory (Rossi et al. 2011), voting theory (Coughlin 2008), game theory (Fudenberg 1998), sports (Langville and Meyer 2012), decision theory (Roy and Bouyssou 2002), or marketing research (Rao et al. 2007). Most recently, the emerging field of preference learning (Fürnkranz and Hüllermeier 2010a) studies how such qualitative information can be used in a wide variety of machine learning problems.

However, the available theory on preference learning is mostly restricted to non-sequential problems and can not be applied to RL directly. Preferences in sequential problems are difficult to use, because they are only relative. Numeric feedback can be propagated along the sequence, but preferences are only valid in comparison to another, specific sequence. Hence, we can not propagate preferences easily. Therefore, new methods are required to combined RL with *preference learning* (PL) to obtain *preference-based reinforcement learning* (PBRL) algorithms.

I.3 Research Goal

The aim of this thesis is to allow the use of preference-based feedback signals in reinforcement learning to overcome the limitations stated in Section I.1, with the additional focus on reducing the cognitive load for the expert. In contrast to classic preference learning problems, reinforcement learning aims at solving sequential problems. Hence, it is required to relate the obtained preferences to specific choices in the action sequence. Furthermore, the state-action space in reinforcement learning problems is usually quite large, making it necessary to generalize the obtained feedback to states, actions or trajectories that have not been evaluated. Our aim is therefore to introduce techniques that are able to efficiently solve this generalization task in a reinforcement learning setting. To be able to achieve the goal of reducing the cognitive demand, we need to keep the mount of required preferences low. Sample efficiency and generalization are also a prominent topic in classic reinforcement learning, but we target the novel problems introduced by the preference aspect. We consider the following problems in detail:

1. *What problems occur when dealing with preference-based feedback signals, and how can we resolve them?* There are several differences between a numeric reward signal and a trajectory preference. The differences have to be made explicit and analyzed to determine the relevant problems. In particular, the problems with the highest impact on the efficiency have to be resolved.

2. *How can we improve the efficiency of PBRL algorithms, in terms of required preference feedback?* For reducing the expert's cognitive load, we need to reduce the amount of required queries. An efficient PBRL algorithm should generalize the obtained information and only pose the most relevant preference queries.

3. *What problems occur when using preferences defined by humans, and how to deal with them?* Human feedback is not perfectly reliable. As an example, they can make mistakes or induce a bias. The relevant problems have to be determined and appropriate methods should be developed.

4. *What are the similarities and differences between available PBRL algorithms?* Several approaches to PBRL are available, but the methods differ greatly. A unified framework should be introduced that allows to determine the reason, advantage and disadvantages of certain design choices.

We also focus on methods that do not require explicit information about effects of each action (model-free) for allowing a wide range of applications. Our exploration methods use stochastic polices to select actions randomly, based on the expected long-term outcome. They are undirected. Directed methods use global information to compute specific sequences or actions that should be evaluated. Undirected methods usually work better in large state-action spaces as it can be costly to obtain sufficient, global information. However, it is often difficult to prove global convergence. For further information, see Section II.4.1.

I.4 Contributions and Organization of the Work

In light of the state research goal, this work makes the following main contributions:

A A framework is provided, that formally defines the problem and objective, unifying all algorithms for PBRL. The required subtasks are made explicit and different approaches are categorized accordingly. This includes a discussion of related problem settings.

B We survey the field and present the available algorithms in context of the given framework, also considering possible advantages and disadvantages. Implicit requirements and limitations of the algorithms are described explicitly.

C Two different studies concerning human preferences are provided. We discuss the forms of preference feedback that are possible and point out the variant that requires the least amount of expert knowledge. Furthermore, we show the problems that arise when using human preference feedback.

D We show several techniques that enable the generalization of obtained trajectory preference feedback to states and actions for reducing the number of preferences that need to be evaluated by the expert. The techniques also provide us with insight into how to improve the generalization capabilities further.

E We analyze exploration/exploitation techniques for the PBRL setting. In contrast to classic RL, two different exploration tasks have to be considered, but it is unclear if it is beneficial to unify them. We consider unified solutions, but also enhancements that view both tasks explicitly.

F Intelligent preference selection methods are provided that only query the expert for the most informative preferences, concerning the given policy optimization problem. These methods reduce the number of required preferences, but should consider the exploration task mentioned in Contribution E.

G We present three new PBRL algorithms. Two for a model-free setting and one model-based. The latest algorithm defines the current state of the art in PBRL, as far as known to the author. It is also the first PBRL algorithm that is applicable in a model-free, non-parametric setting.

The following listing shows the organization of this thesis. A short description of each chapter is provided that relates it to the stated research goals and contributions. The specific research questions solved in each chapter depend on understanding the general problems of PBRL, stated in Chapter III. Hence, detailed research questions are presented at the beginning of each chapter, after the according topics have been described. For a detailed overview, we point the interested reader to Section VII.1, which includes a tabular overview over all specific research questions. Most of the chapters are based on prior publications of the author which are also indicated.

- **Chapter II**
  The foundations chapter introduces the fundamental concepts required for understanding this thesis. We present the commonly used notation and formalization for RL problems and discuss three common problems encountered in reinforcement learning as well as the basic solution principles that we use. Furthermore, we show recent advances in RL that are used throughout this thesis, as well as some alternatives that could be used. We also discuss preference learning and its basic solution techniques as well as how to solve approximation problems using *function approximation* (FA).

- **Chapter III** (Research Goal 4; Contributions A, B, C)
  This chapter is based on ***Wirth and Fürnkranz (2013d) and Wirth et al. (2017)****.*
  In this chapter, we present the *Markov decision process with preferences* (MDPP) and the formal objective that has to be solved. Furthermore, a successful PBRL algorithm must solve multiple issues that we state explicitly as well as the solutions that we encountered while surveying the field. This overview also includes the novel algorithms presented throughout this thesis. We also discuss the relation to other RL approaches that do not use numeric feedback.

- **Chapter IV** (Research Goal 1 & 2; Contribution D, E, G)
  This chapter is based on ***Wirth and Fürnkranz (2013a, 2013b, 2013c)****.*
  Only few approaches to PBRL have been known prior to this thesis and we selected the approach by Fürnkranz et al. (2012) as a starting point for our research. We discuss the reasons for this selection and explain the approach as well as the unsolved generalization problems. We also demonstrate two solutions that are able to implement a generalization method for this algorithm. The methods are then used to analyze the effects of the different simplifications implied by trajectory preferences. Besides the generalization-based efficiency improvements, we also apply additional techniques for improving sample efficiency.

- **Chapter V** (Research Goal 3; Contribution C, G)
  This chapter is based on ***Wirth and Fürnkranz (2012, 2015)****.*
  In this chapter, we analyze a different preference learning paradigm, that is possibly able to provide better generalizations, but at the cost of higher approximation errors. Due to our focus on human preferences, errors in the feedback signal have to be assumed, possibly potentiating the problems of approximation errors. Therefore, we turn to a specific domain that allows us to consider the preference learning problem in a sequential setting while disregarding the reinforcement learning problem itself. The chess domain supplies

us with a high number of preferences and a predefined policy approximator, enabling us to use a batch version of reinforcement learning for computing a policy. In turn, we analyze the effects of the human preferences for our policy and derive a second, improved method.

- **Chapter VI** (Research Goal 2 & 3; Contributions D, E, F, G)
  This chapter is based on ***Wirth et al. (2016)****.*
  Finally, we bring together all obtained information to create a state-of-the-art PBRL algorithm, capable of learning from human preferences in a model-free, non-parametric setting. It solves all subproblems stated in Chapter III explicitly and uses state-of-the-art reinforcement learning methods.

- **Chapter VII**
  In this final chapter, we wrap up all research questions and the obtained answers. We also relate the analyzed approaches to the problems and PBRL characteristics mentioned in Chapter III. Furthermore, we discuss the obtained results in relation to other publications, further supporting most of our claims. The conclusions from this thesis are presented in a compact manner and possible future research directions are suggested.

Chapter II
Foundations

II.1 Preliminaries

Machine learning is about learning a model that predicts an output, given some input. The input objects $x \in \mathscr{X}$ can differ greatly, depending on the domain. As an example, they can be documents in a document labeling task, financial data in a stock market forecast task, or the properties of a robot when trying to predict the next action the robot should perform. The output possibilities range from a set of distinct objects over numeric values to structures. For learning such a model in a supervised manner, evaluative feedback of the current model is assumed. Supervised learning uses a *training set* where the expected output is known. The model should then minimize the difference between the expected output and the predicted output. However, the performance evaluation has not to be performed on the training set, but on a separate *test set*, as we are interested in models that are able to generalize to unseen data. The expected output can be directly given as binary, numeric or ordinal output values or indirectly as ranks or relations between objects. Besides the output, the model often requires some information about the properties of the object, like size, color or position. The function $\boldsymbol{\phi}(x)$ maps objects x to features of x. Depending on the objects, the features can again differ greatly, but are usually binary, nominal, ordinal or numeric.

The objects relevant to this thesis are defined by an *Markov decision process* (MDP), as we explain in the following Section II.2.1. Finding solutions to MDPs is difficult; we show the problems that will arise in Section II.2.2 and general solution principles in Section II.2.3. When solving MDPs, it is often required to compute models or functions able to generalize and approximate from a given sample set via function approximation, as we show in Section II.3. Section II.3.1 introduces approximators that are linear in the features whereas Section II.3.3 and Section II.3.4 show how to define complex, non-linear functions. Section II.4 discusses recent advances in *reinforcement learning* (RL), the principle solution technique for MDPs.

The second problem type relevant to this thesis is *preference learning* (PL), which we introduce in Section II.5, starting with the formal definition of preferences. Section II.5.2 shows the assumptions made in different PL settings, followed by the two most common solution principles in Section II.5.3.

.

II.2 Reinforcement Learning

Reinforcement learning (RL) is a framework for finding a solution for a *sequential decision problem* (SDP). More exactly, it assumes a specific version of a SDP, an *Markov decision process* (MDP).

II.2.1 Markov Decision Processes[2]

An MDP is defined by a sextuple $(S, A, \mu, R, \delta, \gamma)$. We are given a *state space* S and an *action space* A, represented by feature vectors $\boldsymbol{\phi}(s)$ and joint state-action features $\boldsymbol{\phi}(s, a)$. The dimensionality of these feature vectors is defined by $D_{\boldsymbol{\phi}(s)}$ and $D_{\boldsymbol{\phi}(s,a)}$. The state-action spaces and their feature spaces can either be discrete, $\boldsymbol{\phi}(s) \in \mathbb{N}^{D_{\boldsymbol{\phi}(s)}}$, or continuous, $\boldsymbol{\phi}(s) \in \mathbb{R}^{D_{\boldsymbol{\phi}(s)}}$. $A(s)$ defines the set of actions available in state s and $\mu(s)$ is the distribution of possible initial states s_0. $r(s, a, s') \in \mathbb{R}$ defines the *reward* obtained by invoking action a in state s and observing next state s'. However, in most domains it is sufficient to consider state-action rewards $r(s, a)$. The *transition function* $\delta(s' \mid s, a)$ is assumed to be stochastic and defines the distribution of possible follow-up states s' when applying action a to state s. A *policy* $\pi(a \mid s)$ is a distribution that assigns probabilities to actions choices based on the current state. Alternatively, a policy can be formulate deterministically as $\pi(s) = a$. A policy induces *trajectories* $\boldsymbol{\tau} = \{s_0, a_0, s_1, a_1, \dots, s_{n-1}, a_{n-1}, s_n\}$ as an alternating sequence of states and actions that gets realized by selecting actions for the current state, according to the policy, and repeat with the state returned by the transition function. A single state-action pair of a trajectory is index by $\boldsymbol{\tau}(i) = (s_i, a_i)$. We use the notation $\boldsymbol{\tau}^{[i]}$ to describe that $\boldsymbol{\tau}$ was obtained by applying policy $\pi_i(a \mid s)$. Equivalently, $(s, a)^{[i]}$ defines that the state-action sample was obtained by applying $\pi_i(a \mid s)$. $\Pr^{[i]}(s, a, s')$ is the probability of observing (s, a, s') when applying $\pi_i(a \mid s)$ and $\delta(s' \mid s, a)$ to state s.

A trajectory can be evaluated by its accumulated reward, or return

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{|\boldsymbol{\tau}|-1} r(s_t, a_t, s_{t+1}), \tag{II.2.1}$$

with $|\boldsymbol{\tau}| \in \mathbb{N}^+$ as the time steps of the trajectory. In the infinite-horizon, it is required to consider the discounted return

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}). \tag{II.2.2}$$

where the *discount factor* $\gamma \in [0, 1)$ is mathematically required to ensure that $R(\boldsymbol{\tau})$ can not become infinite. Additionally, intermediate rewards become more important than long-term effects. The states are assumed to be subject to the Markov property (Sutton and Barto 1998, Chapter 3.5) which states that the given state (and action) information is sufficient to determine the next state. This implies that the policy is independent of the history of the state sequence, but this assumption can be relaxed using methods like partial-observable MDPs (Spaan 2012, Chapter 12).

[2] This section is based on van Otterlo and Wiering (2012), van Hasselt (2012) and Deisenroth et al. (2013).

A trajectory $\boldsymbol{\tau}$ is an observation of an agents behavior and we want to find an agent that creates trajectories with maximal, cumulative reward. Hence, our learning goal is to find a policy π^* maximizing the expected return

$$\begin{aligned} R(\pi) &= \mathbb{E}_\pi \left[\int R(\boldsymbol{\tau}) \Pr^\pi(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \right], \\ \pi^* &= \arg\max_\pi R(\pi), \end{aligned} \tag{II.2.3}$$

over the distribution of trajectories $\Pr^\pi(\boldsymbol{\tau})$, as induced by the policy and the transition function. We also use the notation $Q^\pi(s, a)$ as the *Q-function*

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\int R(\boldsymbol{\tau}) \Pr^\pi(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \,|\, s_0 = s, a_0 = a) \right], \tag{II.2.4}$$

which is defined as the expected return when starting in state s, selecting action a first and following policy π afterwards. The according, optimal Q-function is denoted as

$$Q^*(s,a) = \mathbb{E}_{\pi^*} \left[\int R(\boldsymbol{\tau}) \Pr^{\pi^*}(\boldsymbol{\tau}) \mathrm{d}\boldsymbol{\tau} \,|\, s_0 = s, a_0 = a) \right]. \tag{II.2.5}$$

We can also define a state value function

$$V^\pi(s) = \mathbb{E}_\pi \left[\int \pi(a \,|\, s) Q^\pi(s,a) \, \mathrm{d}a \right]. \tag{II.2.6}$$

When using deterministic policies, the trajectory distribution is given by

$$\Pr^\pi(\boldsymbol{\tau}) = \mu(s) \prod_{t=0}^{|\boldsymbol{\tau}|-1} \delta(s_{t+1} \,|\, s_t, \pi(s_t)). \tag{II.2.7}$$

For stochastic policies, the distribution is defined by

$$\Pr^\pi(\boldsymbol{\tau}) = \mu(s) \prod_{t=0}^{|\boldsymbol{\tau}|-1} \delta(s_{t+1} \,|\, s_t, a_t) \pi(a_t \,|\, s_t). \tag{II.2.8}$$

In the following, we will assume the more general, stochastic representation (II.2.8) unless mentioned explicitly.

II.2.2 Problems in Reinforcement Learning

Determining an optimal solution requires computation of the expected return for a policy and maximizing it. Usually, it is not possible to solve this problem in closed form because of the complex form of $\delta(s_{t+1} \mid s_t, a_t)$. Therefore, we can only draw samples from the MDP and perform sample-based calculations. This results in several problems we discuss in the following subsections.

II.2.2.a Dependence on the Transition Function

The expected return of a policy can be computed exactly in case $\mu(s)$, $\delta(s' \mid s, a)$ and $r(s, a, s')$ are known, as shown by Equation II.2.3. In practical settings, $\mu(s)$ and $r(s, a, s')$ may be known, but $\delta(s' \mid s, a)$ is usually not available. This may have multiple reasons. In many robotics domains, the transition function is theoretically known as it is the function of the robots mechanics, but real world robots are subject to inexact motor controls or the exact environment properties are unknown. In other domains, this function is completely unknown. Consider a medical treatment plan where it is not possible to forecast the next state of the patient based on its current condition and the applied treatment. Therefore, availability of $\delta(s' \mid s, a)$ can not be assumed in practice.

It is possible to approximate $\delta(s' \mid s, a)$ based on environment samples $\Pr(s, a, s')$, e.g.,

$$\delta(s' \mid s, a) = \frac{\Pr(s, a, s')}{\Pr(a \mid s) \Pr(s)} = \frac{\Pr(s, a, s')}{\pi(a \mid s) \sum_{a,s'} \Pr(s, a, s')}, \tag{II.2.9}$$

would be a very simple and basic approximation. Hence, we can use an approximate transition model to compute an optimal policy (Brafman and Tennenholtz 2001). However, this may introduce an approximation error, as we only have access to a finite number of samples. Therefore, using an approximation of $\delta(s' \mid s, a)$ for computations can be dangerous but it can simplify the computation of a policy's expected return. Hence, approaches that assume knowledge of $\delta(s' \mid s, a)$ are reasonable, but alternatives not using $\delta(s' \mid s, a)$ are also required. Approaches with knowledge of the transition function are called *model-based* whereas other approaches are *model-free*.

II.2.2.b Exploration of the Transition Function

Collecting transition samples is often very expensive. This can be in terms of time, e.g., in a real world robotics domain, it is required to set up the robot. This can also be in terms of cost, e.g., consider collecting samples of a trading system where it is required to perform an actual trade. Therefore, it is of significant relevance when and where to request new transition samples. Usually, it is most beneficial to request samples for parts of the state space where it is expected that we will observe high returns. This potentially allows us to improve on the current, optimal policy, as it operates in this part of the state space. However, our expectation may underestimate the real return because of large approximation errors due to a low sample count. Hence, it is required to balance improving the current best estimate (*exploitation*) and reducing the uncertainty in other parts of the state space (*exploration*). This is also relevant for model-free approaches as they also compute an expectation of an policy's return based on transition samples. Therefore, the expected return is subject to approximation errors comparable to the approximated transition function of model-based approaches. Resolving the exploration/exploitation dilemma is subject of ongoing research with plenty of different methods available. In the following, we will present three, common baseline methods that derive a policy, which maximizes the expectation (II.2.3), while still maintaining exploration. All three methods are subject to a hyper-parameter that needs to be tuned and decayed over time.

ϵ-greedy

The most basic method for resolving this exploration/exploitation tradeoff is an ϵ*-greedy* policy (Watkins 1989). Consider a policy (II.2.3), trying to maximize the expected return, as defined by Equation II.2.4. We want to derive a policy $\tilde{\pi}(a \mid s)$ that maintains exploration.

$$\tilde{\pi}(a \mid s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|} & \textbf{if } a = \arg\max_{a \in A(s)} Q^{\pi}(s, a) \\ \frac{\epsilon}{|A(s)|} & \textbf{otherwise} \end{cases}, \tag{II.2.10}$$

defines an ϵ-greedy policy with $\epsilon \in [0, 1]$ as the probability for selecting a random action. This method is easy to implement, but it does not consider the expected return for suboptimal values. Furthermore, it is only applicable in domains with discrete action spaces $A(s)$ as it is required to compute the number of available actions $|A(s)|$.

Softmax

A softmax policy (Sutton and Barto 1998, Chapter 2.3) weights all actions by their expected return. Therefore, a distribution is introduced that maps the the expected return (II.2.4), to action selection probabilities. Most commonly, a Gibbs distribution is used for such an exploration strategy, resulting in

$$\tilde{\pi}(a \mid s) = \frac{e^{Q^{\pi}(s,a)/t}}{\int e^{Q^{\pi}(s,a)/t} \, \mathrm{d}a}, \tag{II.2.11}$$

as the according policy. $t \in \mathbb{R}_0^+$ is the temperature parameter with defines the tradeoff between pure exploration ($t = \infty$) and pure exploitation ($t = 0$). This policy is better suited for maintaining exploration as it also respects the expected return of an action. However, the parameter t is difficult to tune because it depends on the range of possible returns, as induced by the MDP. In theory, this method is applicable to continuous action spaces, but $\int e^{Q^\pi(s,a)}\,\mathrm{d}a$ is usually difficult to calculate. Hence, softmax is mostly used in discrete action spaces or for sample-based approximations, where it simplifies to

$$\tilde{\pi}(a \mid s) = \frac{e^{Q^\pi(s,a)/t}}{\sum_{a' \in A(s)} e^{Q^\pi(s,a')/t}}. \tag{II.2.12}$$

Gaussian Exploration

In case of continuous action spaces, it is possible to simply define a Gaussian policy

$$\tilde{\pi}(a \mid s) = \mathcal{N}(\arg\max_{a \in A(s)} Q^\pi(s,a), \sigma^2), \tag{II.2.13}$$

with the return-maximizing action as the mean, and variance σ^2 as the exploration parameter (van Hasselt and Wiering 2007). This method requires an ordering of the action space where actions close together result in similar outcomes. Continuous action spaces satisfy this requirement implicitly, but discrete action spaces usually not. In theory, it could be applied to a well-defined discrete action space, although in practice softmax is easier to apply.

II.2.2.c Off-Policy Learning[4]

The expected return of a policy, as defined by Equation II.2.3, depends on the distribution $\pi(a \mid s)$, but it can be difficult or dangerous to draw samples by applying the current policy $\pi(a \mid s)$. In a medical domain, one should not request samples that may harm the patient, or it may be too expensive to obtain a significant number of samples from $\pi(a \mid s)$. Hence, it is important to be able to (re-) use samples obtained from a different distribution. Sample reuse is especially relevant as most algorithms iterate over multiple policies π_i before converging to the optimal policy π^*.

So-called *off-policy* algorithms can compute the policy's expected return based on samples from any other distribution, but this is not the case for *on-policy* algorithms. On-policy algorithms require samples from a policy that is greedily derived; a policy where the most probable action maximizes the expected return. Hence, mechanisms are required that allow us to use sam-

[4] This section is based on Sutton and Barto (1998, Chapter 5&6).

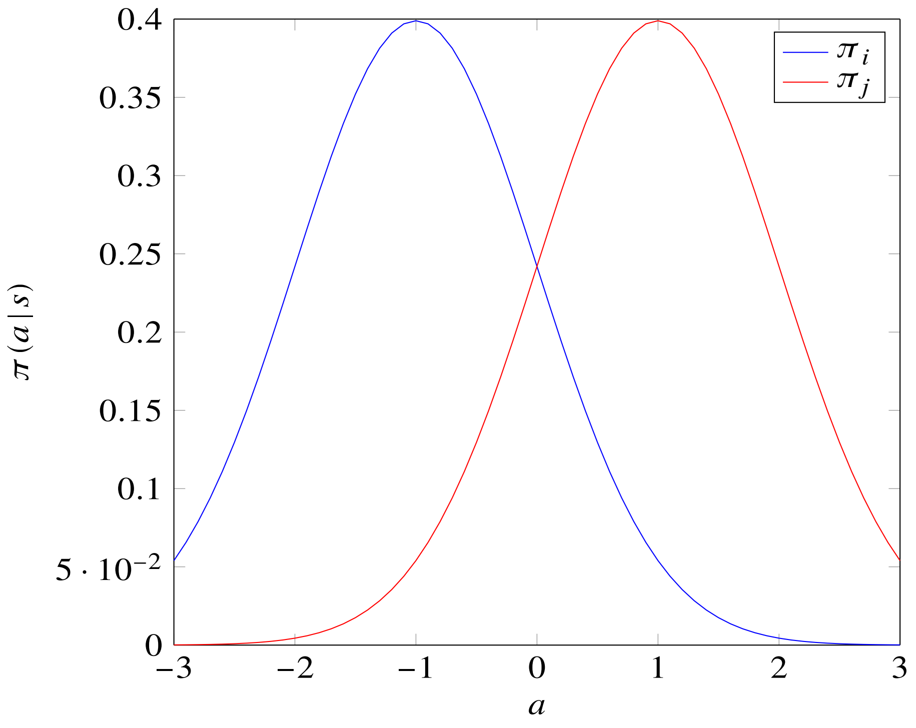


Figure II.2.1.: Two differing distributions

ples from any distribution $\pi_i(a \mid s)$ for computing expectations for a different policy $\pi_j(a \mid s)$. It is impossible to compute a sufficient approximation in case the two distributions differ greatly, as the relevant data is missing. For example, consider the range $a \in [1, 3]$ in Figure II.2.1, where it is problematic to correctly approximate π_j as the probability of obtaining any samples from in π_i this range is fairly low. Therefore, it is usually best to reuse old samples for improving the estimate in areas common to both distributions while also obtaining new samples. This ensures efficiency while also allowing to compute good approximations in areas of low probability within the sampling distribution.

In the following, we will present two methods for reusing samples from a single distribution $\pi_i(a \mid s)$. For reusing samples from multiple iterations, it is also possible to phrase the sampling distribution as a mixture of all used policies.

Rejection Sampling

One basic procedure for reusing samples from a different distribution is *rejection sampling* (Devroye 1986). Assume we have drawn transition samples (s, a, s') with $\Pr^{[i]}(s, a, s')$ as the occurrence probability under policy $\pi_i(a \mid s)$. We now want to use them for computations underlying distribution $\pi_j(a \mid s)$. Therefore, we need to estimate the sample probability

$$\Pr^{[j]}(s, a, s') = \delta(s' \mid s, a)\pi_j(a \mid s), \tag{II.2.14}$$

but we have to consider that the occurrence probability of the sample is dependent on $\pi_i(a \mid s)$. Rejection sampling defines an acceptance probability

$$\Pr_{\text{accept}}(s, a, s') = \frac{\Pr^{[j]}(s, a, s')}{k \Pr^{[i]}(s, a, s')} = \frac{\pi_j(a \mid s)}{k\pi_i(a \mid s)}, \tag{II.2.15}$$

with $k \in \mathbb{R}^+$ as a constant satisfying $k \geq \min \frac{\Pr^{[j]}(s,a,s')}{\Pr^{[i]}(s,a,s')}$. It is now required to draw a uniformly distributed random number $u_n \in [0, 1]$ for each sample and add the sample to the calculations if $\Pr_{\text{accept}}(s, a, s') > u_n$. Therefore, we can now use a subset of all samples $\Pr^{[i]}(s, a, s')$ to be used as $\Pr^{[j]}(s, a, s')$.

Importance Sampling

The idea of *importance sampling* (Shacter and Peot 1989) is to calculate a (normalized) weighting, or importance factor, for each obtained sample. This limits its application to calculating the expectation of a distribution, but this is usually sufficient for sample reuse in RL. Importance sampling computes a *likelihood ratio*

$$\frac{\Pr^{[j]}(s, a, s')}{\Pr^{[i]}(s, a, s')} = \frac{1}{\sum_{(s,a,s')} \big(\pi_j(a \mid s)/\pi_i(a \mid s)\big)} \frac{\pi_j(a \mid s)}{\pi_i(a \mid s)}, \tag{II.2.16}$$

with $\frac{1}{\sum_{(s,a,s')}\left(\pi_j(a \mid s)/\pi_i(a \mid s)\right)}$ as the normalizer, summing over all (sampled) triples. This ratio is now the weighting factor required for aggregating samples. The expected return of a policy π_j is then

$$\mathbb{R}(\pi_j) = \frac{1}{\sum_{(s,a,s')} \big(\pi_j(a \mid s)/\pi_i(a \mid s)\big)} \sum_{(s,a,s')} r(s, a, s') \frac{\pi_j(a \mid s)}{\pi_i(a \mid s)}. \tag{II.2.17}$$

Importance sampling is superior to rejection sampling for calculating expectations as it is able to reuse all samples and not just a subset.

II.2.3 Policy Learning[6]

Finding the optimal policy (II.2.3) usually requires iterating over repeated steps of exploration, evaluation and policy update.

1. Exploration: Create new trajectories $\boldsymbol{\tau}^{[i]}$ based on the current policy π_i.
2. Evaluate: Asses quality of the observed trajectories $\boldsymbol{\tau}^{[i]}$ or state-action pairs $(s, a)^{[i]}$.
3. Update: Compute policy π_{i+1} based on the evaluation, increasing the chance to generate high quality trajectories.

The exploration step assumes a given policy and applies it to the MDP for generating new trajectories, as explained in Section II.2.1.

In the policy evaluation step, the quality of the trajectories is assessed. Either by directly computing the return $R(\boldsymbol{\tau})$ (II.2.1) of each trajectory or by computing the Q-function $Q^{\pi}(s, a)$ (II.2.4). This can either be achieved by computing Monte Carlo estimates as we describe in Section II.2.3.a or by learning a value function as we explain in Section II.2.3.b.

In the update step, the algorithm creates a new policy π_{i+1} that improves on the policy's expected return, moving it further into the direction of the optimum defined by (II.2.3). The update should not directly maximize the defined quality criterion but still maintain exploration, as explained in Section II.2.2.b. In the trajectory evaluation case, this usually boils down to increasing the chance of realizing trajectories with high return. In the state-action evaluation setting, it is also possible to create new policies by increasing the chance to select the action with the highest, expected return in each state. The according optimality task can be defined as

$$\pi^* = \arg\max_{\pi} \int\int \Pr^{\pi}(s)\pi(a \mid s)Q^{\pi}(s, a)\,\mathrm{d}s\,\mathrm{d}a, \tag{II.2.18}$$

with $\Pr^{\pi}(s)$ as the state distribution induced by the policy π. In both, the evaluation and update step, it is possible to reuse samples from prior iterations, as explained in Section II.2.2.c.

Some methods assume a *parametric policy*

$$\pi(a \mid s, \boldsymbol{\omega}) = f(\boldsymbol{\phi}(s, a), \boldsymbol{\omega}), \tag{II.2.19}$$

with $\boldsymbol{\omega} \in \boldsymbol{\Omega}$ as a parameter vector. This reduces the learning problem to learning the parameter vector $\boldsymbol{\omega}$. A parametric policy may reduce the dimensionality of the learning problem substantially, but can also introduce approximation errors, as will be mentioned in Section II.3. Furthermore, the design of parametric policies usually requires specific domain knowledge. Hence, we disregard this more demanding setting, and refer the interested reader to Deisenroth et al. (2013).

[6] This section is based on Deisenroth et al. (2013).

II.2.3.a Monte Carlo Estimates[8]

Monte Carlo sampling computes an estimate of the expectation $Q^{\pi_i}(s,a)$ (II.2.4) by sampling k new trajectories, starting with (s_0, a_0) and applying policy π_i afterwards.The occurrence probability of a trajectory is then $\Pr^{\pi_i}(\boldsymbol{\tau})$, yielding

$$Q^{\pi_i}(s_0, a_0) = \frac{1}{k}\sum_{j=1}^{k}\sum_{t=0}^{|\boldsymbol{\tau}_j|-1} \gamma^t r(s_t^{[i]}, a_t^{[i]}, s_{t+1}^{[i]}), \tag{II.2.20}$$

as a sample-based estimate for the expectation $Q^{\pi_i}(s,a)$. Equation II.2.20 is a *first-visit* Monte Carlo method, as the samples are only used to approximate the value of the first state-action pair in each trajectory. By using trajectories to approximate the expectation for each encountered state within the time-horizon T, we obtain the more efficient *every-visit* Monte Carlo method

$$\forall t \in [0, T) : Q^{\pi_i}(s_t, a_t) = \frac{1}{k}\sum_{j=1}^{k}\sum_{\tilde{t}=t}^{|\boldsymbol{\tau}_j|-1} \gamma^{t-\tilde{t}} r(s_{\tilde{t}}^{[i]}, a_{\tilde{t}}^{[i]}, s_{\tilde{t}+1}^{[i]}). \tag{II.2.21}$$

In general, this technique is an unbiased estimator, but will often be subject to a high variance. Additionally, a large number of trajectories are usually required for obtaining useful estimates. A well-known example for Monte Carlo-based reinforcement learning is the class of REINFORCE algorithms (Williams 1987, 1992).

II.2.3.b Temporal Difference Learning[10]

An alternative to Monte Carlo sampling is the use of temporal difference learning (Sutton 1988). Equation II.2.4 can be defined recursively using the *Bellmann operator* $Q_\pi = T_\pi Q_\pi$ where T_π is defined as

$$(T_\pi Q^\pi)(s,a) = \int_S \delta(s' \mid s, a)\left(r(s,a,s') + \gamma \int_{A(s')} \pi(a' \mid s') Q^\pi(s', a')\,\mathrm{d}a'\right)\mathrm{d}s'. \tag{II.2.22}$$

8 This section is based on Deisenroth et al. (2013) and Sutton and Barto (1998, Chapter 2.3).

10 This section is based on van Hasselt (2012) and Hoffman et al. (2012).

This function computes $Q^{\pi}(s, a)$ based on the expected value of the successor states $Q^{\pi}(s', a')$. This technique called *bootstrapping* allows to learn the Q-function by solving the minimization problem

$$\begin{aligned} Q^{\pi}(s,a) = \min_{Q(s,a)} \| \int_{S} \delta(s' \mid s,a) \left(r(s,a,s') + \gamma \int_{A(s')} \pi(a' \mid s') Q^{\pi}(s',a') \, \mathrm{d}a' \right) \mathrm{d}s' \\ - Q^{\pi}(s,a) \|, \end{aligned} \tag{II.2.23}$$

which is known as the *Bellmann error* or expected *temporal difference* error because it describes the value difference between two time-steps. This error can be minimized using *dynamic programming* (Bellman 1957), although, this is not possible in practical settings because of the computational complexity and dependence on the transition function.

Temporal difference learning methods minimize the Bellmann error iteratively each time a new transition sample is observed. The *state-action-reward-state-action* (SARSA) algorithm (Rummery and Niranjan 1994; Sutton and Barto 1998) uses

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \alpha \left(r(s,a,s') + \gamma Q^{\pi}(s',a') - Q^{\pi}(s,a) \right), \tag{II.2.24}$$

to update the Q-function. α is a weighting factor for the error minimizing gradient $r(s, a, s') + \gamma Q^{\pi}(s', a') - Q^{\pi}(s, a)$. SARSA is an *on-policy* algorithm, as described in Section II.2.2.c.

An alternative is the so called *Q-learning* (Watkins 1989), that uses the update

$$Q^{\pi}(s,a) \leftarrow Q^{\pi}(s,a) + \alpha \left(r(s,a,s') + \gamma \max_{a' \in A(s')} Q^{\pi}(s',a') - Q^{\pi}(s,a) \right). \tag{II.2.25}$$

It is an *off-policy* algorithm, because samples can be obtained from any sampling policy. However, computing the maximizing action can be difficult in continuous or extreme large action spaces.

More efficient versions of these algorithms allow sample reuse by applying the same gradient update to past state-action pairs, weighted by $\lambda^{\Delta t}$. A technique known as *eligibility traces*. Δt is the time-step difference between the current and the past state-action pair. Hence, Q-value updates propagate along the trajectory. However, it is also possible use *experience replay* and save all observed transition samples and replay them later on to perform a Q-function update.

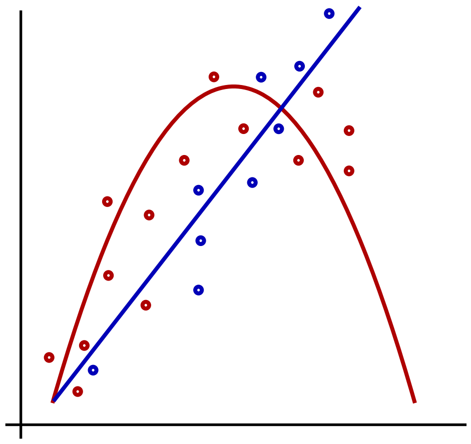


Figure II.3.2.: Example of linear and quadratic function approximation

II.3 Function Approximation

Function approximation (FA) is a about defining a possibly continuous function $f(x) \rightarrow \mathbb{R}$ from point samples $x' \in \mathcal{X}' \subseteq \mathcal{X}$. This can be beneficial because of multiple reasons:

- FA can *generalize* the sampled data to unobserved parts of the function space.
- Methods based on the samples can use the function $f(x)$ instead, that is possibly *faster to evaluate*.
- Depending on the structure of the FA, it can give access to *closed form* solutions.
- FA can be used to define a more compact representation of all samples, *reducing memory requirements*.

Function approximators can be categorized in two classes, *parametric* and *non-parametric*. Parametric function approximators define a function $f(x, \boldsymbol{\theta})$ that can be described by a finite set of parameters $\boldsymbol{\theta} \in \boldsymbol{\Theta}$. As an example, a polynomial is a parametric function as it is fully defined by its coefficients. Parametric approximators can substantially reduce the complexity of learning problems if the number of parameters is significantly lower than the number of samples. Although, it is required to select the correct function beforehand and the wrong approximator may induce large errors. Consider Figure II.3.2 with the point samples approximated by a linear (blue, $f(x) = ax + c$) and quadratic (red, $f(x) = ax^2 + bx + c$) function. The quadratic function is a better approximator for the red points but the linear function better captures the set of all points. Assuming both point sets (red and blue) are obtained by sampling from the same model, it can be seen that the best approximator depends on the already observed samples and is difficult to choose without prior knowledge. Non-parametric function approximators are capable of not only learning the function itself, but can also determine the best function class. However, they are subject to an infinite number of parameters and can therefore not be described by a compact representation. Non-parametric FA requires more data but often leads to better generalization.

Function Approximation in RL

For applying the policy search approach of Section II.2.3, it is required to define a policy $\pi(a \mid s)$. We can model such a policy as a function $f(s,a) = \pi(a \mid s)$. This function can be evaluated for each action if the number of actions $a \in A(s)$ is finite, allowing to choose a single action based on the given probabilities. In continuous, i.e. infinite, action spaces, it is only possible to draw samples from $\pi(a \mid s)$ without further assumptions. Some function approximators assume a specific distribution for $\pi(a \mid s)$, where it is possible to describe the complete probability distribution with a finite set of parameters. For example, assume a Gaussian $\mathcal{N}(\mu, \sigma)$ as the distribution for $\pi(a \mid s)$. This enables us to reduce the problem to finding the two parameters μ and σ. Especially the use of a Gaussian distribution is justified in this setting, as it is usually sufficient to obtain an approximated policy $\tilde{\pi}^*(a \mid s)$ with the same maximum as the optimal policy ($\arg\max_a \tilde{\pi}^*(a \mid s) = \arg\max_a \pi^*(a \mid s)$) for approximately maximizing the expected return (II.2.3).

Besides using FA for approximating policies, it can also be employed for defining the value function $Q(s,a)$ (II.2.4). This case is similar to approximating a policy $\pi(a \mid s)$ with the difference that the outcome is not defining a probability but an expectation. Therefore, the application to discrete action spaces is comparable, by defining $Q(s,a) = f(s,a)$. Assuming a given distribution for $Q(s,a)$ is not as common as in policy approximation for continuous action spaces as the exact values of suboptimal actions may matter, depending on the method used to derive a policy that maintains exploration, as explained in Section II.2.2.b.

II.3.1 Linear Functions[12]

Function approximation (FA) via linear functions assumes that there is a feature space $\boldsymbol{\phi}(x)$ representing the input and the functions output

$$f(x) = \boldsymbol{\theta}^T \boldsymbol{\phi}(x), \tag{II.3.26}$$

is linear in features, depending on the weights or parameters $\boldsymbol{\theta}$. Linear functions are a common choice as it is possible to learn the weights $\boldsymbol{\theta}$ efficiently. Depending on the optimization goal, e.g., minimizing a loss function, it is possible to use linear or convex optimization procedures that are fast and are guaranteed to converge to a single, global optimum.

[12] This section is based on Bishop 2006, Chapter 3.1.

$\boldsymbol{\phi}(x)$	x
$[1, 0, 0, 0]$	$[0, 0]$
$[0, 1, 0, 0]$	$[0, 1]$
$[0, 0, 1, 0]$	$[1, 0]$
$[0, 0, 0, 1]$	$[1, 1]$

Table II.1.: Tabular representation example for a two-dimensional object space

Due to the linearity, the function can not cover dependencies between elements of the feature vector $\boldsymbol{\phi}(x)$. Hence, all relevant dependencies have to be directly encapsulated in the feature vector itself. This can be achieved by handcrafting the features, which requires very detailed knowledge of the task at hand. The expert needs to know which features are dependent and to what degree, therefore this is not often used. Alternatively, it is possible to map the dimensions of the input object x into a higher-dimensional space that captures multiple dependencies where irrelevant dependencies should get a weight of 0 after the optimization process. This may greatly increase the number of features and therefore a higher number samples is usually required to achieve a sufficient approximation. In general, biased functions $f(x) = \boldsymbol{\theta}^T \boldsymbol{\phi}(x) + b$ can be obtained by adding a fixed *bias term* to the feature vector, resulting in $f(x) = [\boldsymbol{\theta}, b]^T [\boldsymbol{\phi}(x), 1]$.

In discrete spaces, it is possible to enumerate all possible combinations of all dimensions of the input object, as explained by Sutton and Barto (1998, Chapter 3.9). Such a tabular representation, as presented in Table II.1, is then a simple vector of Booleans. The tabular representation is not able to generalize, as only one feature is active for any given object, but each object is subject to its own weight in $\boldsymbol{\theta}$. This enables an exact representation of the function to approximate. This method is often used for defining a Q-function $Q(s, a)$ in simple domains. Besides the generalization issues, the dimensionality of the feature space can become extremely large. Additionally, it is not applicable to continuous feature spaces, but this issue can be solved by employing methods like *tile coding* (Sutton and Barto 1998, Chapter 8.3.).

An alternative is to use polynomials as feature space. Such a feature space can generalize and it is usually possible to employ efficient optimization methods. However, the degree of the polynomial has to be known in advance and higher-order polynomials can result in extremely high-dimensional feature spaces.

II.3.2 Kernel methods[14]

An alternative to tabular and polynomial function approximation is a similarity-based definition. As we only have access to a finite number of samples for computing the approximation, it is

[14] This section is based on Bishop (2006, Chapter 6).

sensible to define the function via the similarity to the given sample set. We assume a *kernel function*

$$\mathbb{k}(x, x') = \boldsymbol{\phi}(x)^T \boldsymbol{\phi}(x'), \tag{II.3.27}$$

with $\boldsymbol{\phi}(x)$ as a mapping of x into any suitable space via *basis functions*. A basis function defines a similarity measure, e.g., $\boldsymbol{\phi}(x) = x$ would translate to a linear similarity function. Alternatively, it is also possible to directly define a kernel function that corresponds to any scalar product in some feature space. In general, such a kernel-based feature space is infinite dimensional, because the kernel function can be evaluated between any two points. However, we only use a finite number of sample points $\mathscr{X}'$ where the function value is known. A commonly used kernel function is the squared exponential kernel

$$\mathbb{k}(x, x') = \exp\left(\frac{-(x - x')^2}{2l^2}\right), \tag{II.3.28}$$

with $l \in \mathbb{R}^+$ as the bandwidth parameter. We can again translate any mapping via a kernel function into a linear problem

$$f(x, \boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(x) = \boldsymbol{\theta}^T \mathbb{k}(x, \mathscr{X}'), \tag{II.3.29}$$

as the sample points $x' \in \mathscr{X}'$ are constant. $\mathbb{k}$ returns a vector by applying the kernel function to all points $x' \in \mathscr{X}'$. Calculating $\mathbb{k}(x, \mathscr{X}')$ explicitly is usually not sensible due to computational limitations. Kernel methods are still computational more expensive than other linear methods because of the amount of calculations and the requirement to store the sample set $\mathscr{X}'$ explicitly. The advantage of defining a kernel is that arguably less expert knowledge is required to define a similarity measure than to know the correct polynomial in advance. Defining polynomials is especially problematic in high-dimensional object spaces. Additionally, kernel methods can describe functions of any complexity, given a sufficient number of sample points.

II.3.3 Gaussian Processes[16]

Classic kernel methods and other function approximators suffer from the limitation that they are not able to capture the uncertainty of the approximation. A *Gaussian process* (GP) introduces a Gaussian prior $\Pr(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} \,|\, \mathbf{0}, \alpha^{-1}\mathbf{I})$ over the space of linear functions, defined by Equation II.3.26, with α as the precision hyper-parameter (Williams and Rasmussen 1996). $\boldsymbol{\Phi} = [\boldsymbol{\phi}(x'_1), \dots, \boldsymbol{\phi}(x'_n)]$ is the matrix of all samples $x' \in \mathscr{X}'$ where the outcomes

[16] This section is based on Bishop (2006, Chapter 6.4) and Rasmussen (2004).

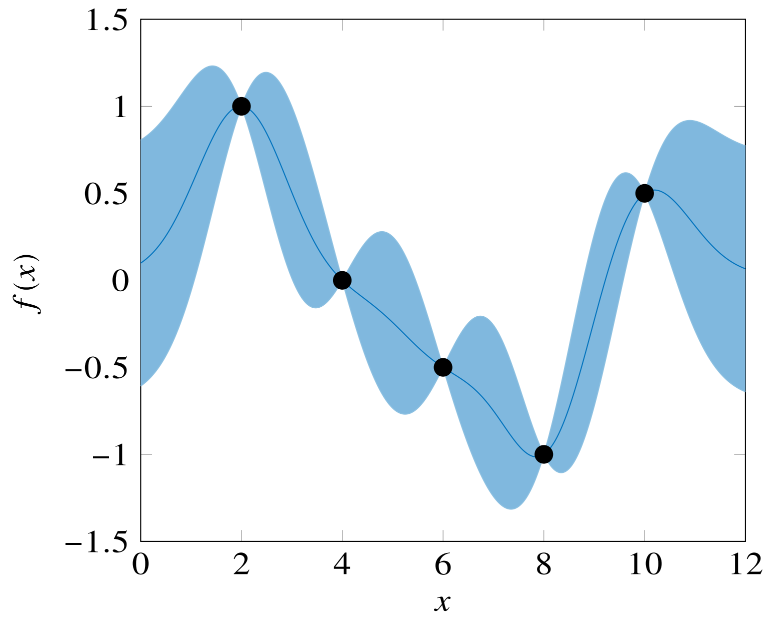


Figure II.3.3.: Posterior distribution of a GP with mean and standard deviation

$\mathbf{f} = [f(x'_1), \dots, f(x'_n)]$ are known and $\mathbf{f} = \boldsymbol{\theta}^T \boldsymbol{\Phi}$ is a linear function. We can rewrite the Gaussian prior as

$$\begin{aligned} \mathbb{E}[\mathbf{f}] &= \mathbb{E}[\boldsymbol{\theta}]^T \boldsymbol{\Phi} = \mathbf{0}, \\ \operatorname{cov}[\mathbf{f}] &= \mathbb{E}\left[\mathbf{f}\mathbf{f}^T\right] = \boldsymbol{\Phi}\mathbb{E}\left[\boldsymbol{\theta}\boldsymbol{\theta}^T\right]\boldsymbol{\Phi}^T = \frac{1}{\alpha}\boldsymbol{\Phi}\boldsymbol{\Phi}^T, \end{aligned} \tag{II.3.30}$$

with zero mean and the identity matrix as covariance matrix. The formulation $\operatorname{cov}(\mathbf{f}) = \frac{1}{\alpha}\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ of the covariance matrix is in fact a kernel matrix $\operatorname{cov}(\mathbf{f}) = \mathbb{k}(\mathscr{X}', \mathscr{X}')$, and it is possible to use any suitable replacement, as explained in Section II.3.2.

The posterior distribution of $f(x)$, given a new sample point x is then

$$\begin{aligned} \Pr(f(x) \mid x, \mathscr{X}', \mathbf{f}) &= \mathcal{N}(f(x) \mid A, B), \\ A &= \mathbb{k}(x, \mathscr{X}')\mathbb{k}(\mathscr{X}', \mathscr{X}')^{-1}\mathbf{f}, \\ B &= \mathbb{k}(x, x) - \mathbb{k}(x, \mathscr{X}')\mathbb{k}(\mathscr{X}', \mathscr{X}')^{-1}\mathbb{k}(x, \mathscr{X}')^T, \end{aligned} \tag{II.3.31}$$

where the kernel may be subject to hyper-parameters. For an more elaborate explanation of the equations, we refer the reader to Bishop (2006, Chapter 6.4) and Rasmussen (2004). Figure II.3.3 shows the mean and standard deviation of the posterior distribution for a GP with a squared exponential kernel (II.3.28) with the black points as training data. A GP is a Bayesian method

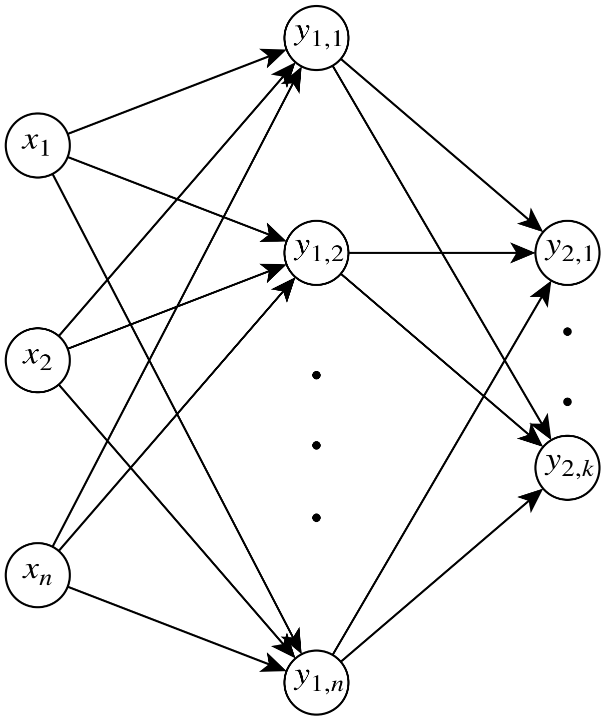


Figure II.3.4.: A neural network with a single hidden layer

that explicitly captures the uncertainty of the model, which can be especially useful for solving the exploration/exploitation problem (cf. Section II.2.2.b). The parameters $\boldsymbol{\theta}$ are integrated out and we only require $\mathscr{X}'$, $\mathbf{f}$ and α to evaluate the model at any point x.

II.3.4 Neural Networks[18]

Kernel-based methods can usually be optimized quickly due to the applicability of convex solvers, but evaluation of new points is expensive as it is required to calculate the kernel matrix. Perceptron-based *neural networks* (Rosenblatt 1958) can also approximation complex functions with a more compact model than kernel methods. The drawback is, their optimization problem is non-convex and is therefore computationally more expensive. Additionally, they are subject to a range of design choices that substantially influence the result and can not be determined in advance without expert knowledge.

Neural networks form a directed graph of so called neurons, defined by

$$y(x, \boldsymbol{\theta}) = h(\boldsymbol{\theta}^T x), \tag{II.3.32}$$

[18] This section is based on Bishop (2006, Chapter 5.1-5.3).

with h as a possibly nonlinear, activation function, that has to be defined in advance. This function should be differentiable as this allows the use of gradient methods for optimization. It is possible to add a bias term, like in linear function approximation (cf. Section II.3.1). The neural network graph is usually acyclic with three layers, as shown in Figure II.3.4, where each layer depends on the output of the last layer. The hidden layer depends on the input layer, defining the input object. The output layer returns the result of the function as aggregation of all hidden units. Each node can have a different activation function. The resulting function approximator for each of the outputs i in the network given in Figure II.3.4 is defined as

$$f_i(x, \boldsymbol{\theta}) = h_{2,i}(\boldsymbol{\theta}_{2,i}^T[h_{1,1}\boldsymbol{\theta}_{1,1}^T x, \dots, h_{1,n}\boldsymbol{\theta}_{1,n}^T x]), \tag{II.3.33}$$

with $\boldsymbol{\theta}$ as the tensor of weights for each neuron and $[h_{1,1}\boldsymbol{\theta}_{1,1}^T x, \dots, h_{1,n}\boldsymbol{\theta}_{1,n}^T x]$ as the vector of results from the hidden layer. The expressiveness of a neural network depends on the number of nodes in the hidden layer, but a neural network can theoretically approximate any function. In case a binary or classification function should be approximated, it is possible to use thresholding activation functions $h(x) = \mathbb{1}(x > t)$ for the output layer that indicates if the value exceeds t.

Training the weights of a neural networks is usually performed via back propagation algorithms (Werbos 1974), that measure the error between the expected and the observed output and propagates the error from the output layer back through all layers of the network. The weights of each node are moved in the direction of a minimizing solution, e.g., via (stochastic) gradient descent.

Deep Neural Networks

Deep neural networks (DNN) (Hinton and Salakhutdinov 2006) are a modern variant of neural networks that makes use of the fact that the expressiveness of a neural network increases when distributing a fixed number hidden nodes among multiple layers (Bianchini and Scarselli 2014). Therefore, a sequence of hidden layers is used. This allows a DNN to obtain higher expressiveness while keeping it tractable. A DNN can deal with very high-dimensional data and handle large amounts of training data. Furthermore, such a network can approximate very complex functions, making it well suited to deal with complex problems. However, the high number of parameters required to describe a high number of neurons also *requires* a high number of samples for training. This makes it difficult to use in cases where training samples are expensive to obtain. Furthermore, the optimization procedure and the number of training samples makes optimizing a DNN a time consuming process.

II.4 Recent Advances in Reinforcement Learning

Current research in reinforcement learning concerns three different subproblems. Most important are methods that solve the exploration/exploitation (Section II.2.2.b) dilemma efficiently. Reducing exploration as quickly as possible can substantially reduce the number of required samples and enables application to high-dimensional problem spaces. However, insufficient exploration prevents the algorithms from obtaining optimal or even good policies.

The second problem concerns the policy evaluation (cf. Section II.2.3). Monte Carlo estimates (Section II.2.3.a) are reliable but inefficient as it may be necessary to obtain a high number of samples to reduce the variance to an acceptable amount. Furthermore, Monte Carlo estimates can not be used to generalize to unseen policies or trajectories. Hence, more efficient estimators, such as value function approximation (Section II.2.3.b), are of interest. A function-based evaluation enables generalization but may be biased and can suffer from overfitting (Mannor et al. 2004). Therefore, techniques circumventing these problems are required.

Lastly, generalization and approximation techniques for policies and value functions are important as they decide if it is possible to derive reliable information for unevaluated policies or trajectories.

II.4.1 Advances in Exploration Methods

Several approaches for resolving the exploration/exploitation tradeoff have been proposed. The two general methods are *directed methods*, which guide the exploration into an explicit direction, and *undirected exploration*, which only define a stochastic policy for enabling exploration. Directed exploration can guide the exploration globally by defining an explicit search direction. This enables formal bounds on complexity and convergence, but usually limits the application to low-dimensional spaces. A stochastic policy allows deviations from the current estimate of the optimal action. Hence, they explore local regions close to the estimated optimum. Due to the locality, they can scale to high-dimensional spaces, but it is difficult to prove global convergence without global search.

Controlling Exploration with Stochastic Policies

When using undirected exploration, we only need to control the stochasticity of the policy, but do not define an explicit exploration direction. Hence, undirected exploration boils down to controlling a trade-off between maximizing the expectation (II.2.18) and the probability to sample in possibly suboptimal parts of the policy space.

Information loss-based exploration is a common, powerful method for updating stochastic policies that considers the fact that policy improvement may induce a loss of information. These methods increase the likelihood of sampling in the expected, optimal region while reducing the sampling probability in other parts of the policy space. Hence, they decrease the variance of the policy and induce an information loss. Loosing information too quickly can result in premature convergence. Hence, it is reasonable to constrain the information loss. This idea was first identified by Kakade (2001). He proposed to limit the information loss in terms of the Fisher information metric (Rao 1945; Shun-ichi 2012), which is the second derivate of the *Kullback-Leibler* (KL) divergence. The metric $F_s(\boldsymbol{\omega})$, based on the parameters of a parametric policy $\pi(a \mid s, \boldsymbol{\omega})$, is computed for each state independently. The criterion

$$F(\boldsymbol{\omega}) \equiv \mathbb{E}_{\Pr^{\pi}(s)}[F_s(\boldsymbol{\omega})], \tag{II.4.34}$$

is therefore not defined on a proper distribution, but on a collection of distributions for each state. Bagnell and Schneider (2003) overcame this problem by defining a criterion based on the distribution of trajectories $\Pr(\boldsymbol{\tau} \mid \boldsymbol{\omega})$, but the approach still depends on a parametric policy.

Peters et al. (2010) derived a more general solution by computing sample weights that can be used to fit any suitable policy. The basic principle is to bound the entropy loss of a policy improvement step in terms of KL divergence

$$D(\Pr^{\pi_{i+1}}(s,a) \mid\mid \Pr^{\pi_i}(s,a)) = \sum_{s,a} \Pr^{\pi_{i+1}}(s)\pi_{i+1}(a \mid s) \log\left(\frac{\Pr^{\pi_{i+1}}(s)\pi_{i+1}(a \mid s)}{\Pr^{\pi_i}(s,a)}\right) \leq \epsilon, \tag{II.4.35}$$

where $\Pr^{\pi_i}(s,a)$ is the data distribution observed in the last iteration and $\Pr^{\pi_{i+1}}(s,a) = \Pr^{\pi_{i+1}}(s)\pi_{i+1}(a \mid s)$ as induced by the new policy. A policy improvement step is now defined by maximizing the policy's expected return (II.2.3), while respecting the constraint (II.4.35), that limits the KL divergence to $\epsilon \in \mathbb{R}^+$. The complete *relative entropy policy search* (REPS) procedure can be defined using samples (s, a, s', r), possibly obtained off-policy. Hence, we can employ sample reuse, as described in Section II.2.2.c. Furthermore, a REPS iteration does not directly yield a new policy $\pi^{i+1}(a \mid s)$, but sample weights $\Pr^{\pi_{i+1}}(s,a)$ for all observed states s and actions a. From that, we can derive a new policy by weighted likelihood maximization

$$\pi_{i+1}(a \mid s) = \arg\max_{\pi} \sum_{(s,a)} \frac{\Pr^{\pi_{i+1}}(s,a)}{\Pr^{\pi_i}(s,a)} \log\left(\pi(a \mid s)\right), \tag{II.4.36}$$

summed over all obtained samples (s, a). This assumes that the state distribution is approximately stationary and therefore disregards $\Pr^{\pi}(s)$. The complete transition sample, including s' and r, are only required for maximizing the return.

Algorithm 1 R-max

```
Require: initial policy $\pi_0$, iteration limit $m$, update limit $n$, sample limit $k$, max. reward $r_{\max}$
 1: $\tilde{r}(s_r, a, s_r) \leftarrow r_{\max}$, $\tilde{\delta}(s_r \mid s_r, a) \leftarrow 1$    ▷ Initialize approximate system dynamics
 2: for $i = 0$ to $m$ do
 3:     $s \sim \mu(s)$    ▷ Draw initial state
 4:     for $j = 0$ to $n$ do
 5:         $a \sim \pi_i(a \mid s)$    ▷ Apply policy
 6:         $s' \sim \delta(s' \mid s, a)$    ▷ Apply transition function
 7:         $C(s, a, s') = C(s, a, s') + 1$
 8:         $C(s, a) = C(s, a) + 1$    ▷ Increment visit counts
 9:         $r_{\text{sum}}(s, a, s') \leftarrow r_{\text{sum}}(s, a, s') + r(s, a, s')$    ▷ Update observed reward sum
10:         if $C(s, a) \geq k$ then    ▷ Check sample limit
11:             for $s' \in C(s, a, \cdot)$ do    ▷ Update approximate model
12:                 $\tilde{r}(s, a, s') = r_{\text{sum}}(s, a, s') / C(s, a, s')$    ▷ Compute expected reward
13:                 $\tilde{\delta}(s' \mid s, a) \leftarrow C(s, a, s') / C(s, a)$
14:             end for
15:         else
16:             $\tilde{r}(s, a, s_r) \leftarrow r_{\max}$, $\tilde{\delta}(s_r \mid s, a) \leftarrow 1$
17:         end if
18:     end for
19:     $\pi_{i+1} = \text{computePolicy}(\tilde{r}(s, a, s'), \tilde{\delta}(s' \mid s, a))$    ▷ Compute (model-based) policy
20: end for
21: return improved policy $\pi_m$
```

We can use any suitable function approximator for $\pi_{i+1}(a \mid s)$ and are not restricted to parametric policies. As an example, we can use GPs, as introduced in Section II.3.3. Regret bounds can be proven in a setting with discrete, finite time horizon problems and layered, adversarial MDPs (Zimin and Neu 2013). However, empirical evaluations show good results in complex domains (Daniel et al. 2013; Kupcsik et al. 2014; Schulman et al. 2015).

Computing Explicit Search Directions

For reward-based exploration methods, known as *R-max*, upper bounds on the required samples for convergence have been proven (Brafman and Tennenholtz 2001). Their basic idea is to maintain an approximate model of the system dynamics $\tilde{\delta}$ and $\tilde{r}$ that differentiates between observed and unobserved states, as shown in Algorithm 1. The transition model is initialized with an absorbing state s_r, representing all unobserved states and a transition to itself with maximal reward for all available actions. In case a new state-action pair is observed, it is assumed that applying the transition function to the state with the given action leads to the absorbing state with maximal reward. The approximated transition model and the expected reward are set to the observed values after k samples. Hence, all state-actions are assumed to be able to obtain maximum return until k samples are drawn. Therefore, deriving an optimal policy from the approximate transition model guides the policy to state-actions that have not been sampled at least k times. To be able to compute a tabular representation for the approximate system dynamics, a finite state and action space is required. Jong and Stone (2007) enable applications to continuous domains by using instance-based similarity methods. For computing the expected return or next state for a state-action pair, similarity to observed samples is computed to determine a similarity weight. The expectation is then computed as the weighted average over all samples. This technique may introduce additional approximation errors, but the effects on the sample bound have not been analyzed.

II.4.2 Advances in Temporal Difference Learning

In practical settings, it is not possible to use a tabular representation for a Q-function. It is usually required to use a parametric function $\hat{Q}^{\pi}(s, a, \boldsymbol{\theta})$ that approximates the optimal function. A common way is to assume a linear function $\hat{Q}^{\pi}(s, a, \boldsymbol{\theta}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s, a)$, as we explained in Section II.3.1. However, updating a Q-function with temporal difference learning methods, as described in Section II.2.3.b, is now a problem of updating $\boldsymbol{\theta}$. Hence, updates will influence each other and minimizing temporal difference error for one sample may increase the error for another sample. Furthermore, it is usually not possible to obtain a linear approximation of the optimal Q-function and we have to deal with approximation errors.

Least-Squares Temporal Difference

The method of *least-squares temporal difference learning* (LSTD) can compute a linear Q-function by viewing the problem as a batch optimization problem (Boyan 1999; Hoffman et al. 2012). Therefore, it is possible to minimize the *Bellmann error* (II.2.23) directly by using all observed transition samples. Furthermore, the introduction of a projection operator Π such that $\Pi Q^{\pi}(s, a) = \mathbf{u}^{*T} \boldsymbol{\phi}(s, a)$ with $\mathbf{u}^* = \arg\min_{\mathbf{u} \in \mathbb{R}^k} ||\mathbf{u}^T \boldsymbol{\phi}(s, a) - Q^{\pi}(s, a)||_{\nu}^2$. $\mathbf{u}^*$ minimizes the squared l_2 loss (w.r.t. to distribution ν) of the difference between the desired function $Q^{\pi}(s, a)$ and the linear approximation $\hat{Q}^{\pi}(s, a, \boldsymbol{\theta})$. Hence, we can minimize the approximation errors induced by the linear approximation. Combining the projector Π with the Bellman operator (II.2.22) yields $\hat{Q}^{\pi} = \Pi T_{\pi} \hat{Q}^{\pi}$. The resulting optimization problem is nested and requires

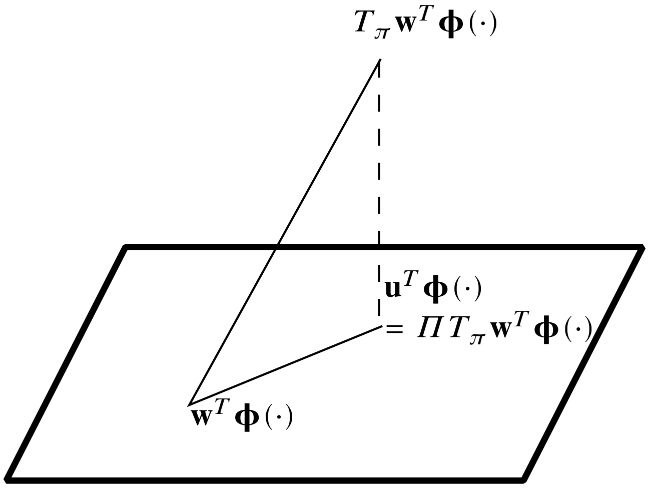

$T_\pi \mathbf{w}^T \boldsymbol{\phi}(\cdot)$
$\mathbf{u}^T \boldsymbol{\phi}(\cdot)$
$= \Pi T_\pi \mathbf{w}^T \boldsymbol{\phi}(\cdot)$
$\mathbf{w}^T \boldsymbol{\phi}(\cdot)$

Figure II.4.5.: Projecting the solution of the Bellman operator back into the function space

minimizing the *projection* error

$$\begin{aligned}\mathbf{u}^* = \arg\min_{\mathbf{u}\in\mathbb{R}^k} &\|\mathbf{u}^T \boldsymbol{\phi}(s,a) \\ &- \int_S \delta(s' \mid s,a)\left(r(s,a,s') + \gamma \int_{A(s')} \pi(a' \mid s')\mathbf{w}^T \boldsymbol{\phi}(s',a')\,\mathrm{d}a'\right)\mathrm{d}s'\|_v^2,\end{aligned} \tag{II.4.37}$$

with $\mathbf{u}^*$ as the minimizing parameter vector. The parameter vector $\mathbf{w}$ depends on the *fixed-point* error

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^k} \|\mathbf{w}^T \boldsymbol{\phi}(s,a) - \mathbf{u}^{*T} \boldsymbol{\phi}(s,a)\|_v^2, \tag{II.4.38}$$

that minimizes the differences between the desired and the linear Q-function. The equations can be defined in a model-free manner, if we have access to samples (s,a,s',a',r) from policy π. Based on numbered samples 1 to n, it is possible to create three sample matrices

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(s_1,a_1)^T \\ \vdots \\ \boldsymbol{\phi}(s_n,a_n)^T \end{bmatrix}, \boldsymbol{\Phi}' = \begin{bmatrix} \boldsymbol{\phi}(s'_1,a'_1)^T \\ \vdots \\ \boldsymbol{\phi}(s'_n,a'_n)^T \end{bmatrix}, \mathbf{R} = \begin{bmatrix} r(s_1,a_1,s'_1)^T \\ \vdots \\ r(s_1,a_1,s'_1)^T \end{bmatrix}.$$

allowing to define empirical versions of Equation II.4.37 and Equation II.4.38. As the samples already underly the unknown distributions $\delta(s' \mid s,a)$ and $\pi(a \mid s)$, we can disregard the factors and obtain

$$\mathbf{u}^* = \arg\min_{\mathbf{u}\in\mathbb{R}^k} \|\mathbf{u}^T \boldsymbol{\Phi} - (\mathbf{R} + \gamma \mathbf{w}^T \boldsymbol{\Phi}')\|_v^2, \tag{II.4.39}$$

as an empirical version of Equation II.4.37. As we now have access to a closed-form solution for $\mathbf{u}^*$ we can plug it in to Equation II.4.38, resulting in

$$\mathbf{w}^* = \left(\mathbf{\Phi}^T(\mathbf{\Phi} - \gamma\mathbf{\Phi}')\right)^{-1} \mathbf{\Phi}^T\mathbf{R} = \mathbf{A}^{-1}\mathbf{b}, \tag{II.4.40}$$

with $\mathbf{A} = \mathbf{\Phi}^T(\boldsymbol{\phi} - \gamma\mathbf{\Phi}')$ and $\mathbf{b} = \mathbf{\Phi}^T\mathbf{R}$. This linear equation system can be solved with any suitable solver. The resulting vector $\mathbf{w}^*$ is then the weighted vector $\boldsymbol{\theta}$ for our linear Q-function. Therefore, the empirical solution to LSTD gives us the means to calculate the parameter vector $\boldsymbol{\theta}$, defining the function $Q^\pi(s, a, \boldsymbol{\theta})$ only using samples (s, a, s', a', r) from policy π. Approximating $Q^\pi(s, a, \boldsymbol{\theta})$ with samples obtained from another policy π' can be achieved by weighting the samples via importance weighting, as introduced in Section II.2.2.c.

Regularized Least-Squares Temporal Difference

Temporal difference methods like LSTD often perform poorly if it is only possible to access a low number of samples in a high-dimensional approximation space. This can be attributed to overfitting. As the problem is the dimensionality of the space to approximate, it is reasonable to employ dimensionality reduction methods. For example, Ghavamzadeh et al. (2010) propose to use a random projection

$$\tilde{\boldsymbol{\phi}}(s, a) = \mathbf{A}\boldsymbol{\phi}(s, a), \tag{II.4.41}$$

with $\tilde{\boldsymbol{\phi}}(s, a)$ as the resulting low-dimensional feature space. $\mathbf{A} \in \mathbb{R}^{k \times D_{\boldsymbol{\phi}(s,a)}}, \mathbf{A} \sim \mathcal{N}(\mathbf{0}, \mathbf{1/k})$ is a projection matrix that reduces the dimensionality from $D_{\boldsymbol{\phi}(s,a)}$ to k. Hence, the projection now combines several features in a single dimension, enabling generalization along the dimensions and directly representing interdependencies between features. However, this is a form of lossy compression and important characteristics may vanish.

Alternatively, it is also possible to add an explicit regularizer to the LSTD algorithm, as shown by Hoffman et al. (2012). By applying a l_2 regularizer to the *projection* error (II.4.39), we obtain

$$\mathbf{u}^* = \arg\min_{\mathbf{u}\in\mathbb{R}^k} ||\mathbf{u}^T\mathbf{\Phi} - (\mathbf{R} + \gamma\boldsymbol{\theta}^T\mathbf{\Phi}')||_2^2 + \beta||\mathbf{u}||_2^2, \tag{II.4.42}$$

subject to the parameter β that defines the tradeoff between loss minimization and generalization. The regularized version of the *fixed-point* error (II.4.38)

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^k} ||\mathbf{w}^T\mathbf{\Phi} - \mathbf{u}^{*T}\mathbf{\Phi}||_2^2 + \beta'||\mathbf{w}||_2^2, \tag{II.4.43}$$

is also subject to a tradeoff parameter β'. In line with Equation II.4.40, we can also obtain a closed-form solution

$$\begin{aligned}
\boldsymbol{\theta} = \mathbf{w}^* &= \left(\mathbf{X}^T\mathbf{X} + \beta'\boldsymbol{I}\right)^{-1}\mathbf{X}^T\mathbf{y}, \\
\mathbf{X} &= \mathbf{C}\left(\mathbf{A} + \beta\boldsymbol{I}\right), \\
\mathbf{y} &= \mathbf{C}\mathbf{b}, \\
\mathbf{C} &= \boldsymbol{\Phi}\left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \beta\boldsymbol{I}\right)^{-1},
\end{aligned} \tag{II.4.44}$$

for the l_{22} *regularized LSTD* problem. $\mathbf{A} = \boldsymbol{\Phi}^T(\boldsymbol{\phi} - \gamma\boldsymbol{\Phi}')$ and $\mathbf{b} = \boldsymbol{\Phi}^T\mathbf{R}$ are as in the original LSTD definition. Other regularizers, such as l_1, can be used as well, as shown by Hoffman et al. (2012).

II.4.3 Advances in Function Approximation for Reinforcement Learning

Parametric policies $\pi(a\,|\,s, \boldsymbol{\theta})$ can work well in settings where explicit control functions are known. However, these functions can usually not be defined without consulting a domain expert, and selecting the wrong function class can result in significant problems. Fixing the policy or value function class beforehand restricts the possible solutions to models that can be represented with the given function class. If it is not possible to describe a (near-) optimal policy or value function with the given class, large errors can not be prevented. Hence, it is important to learn function approximators that do not depend on fixing the function class in advance. Furthermore, the exploitation/exploration tradeoff explained in Section II.2.2.b depends on the certainty of the policy and therefore the certainty of the value function. Hence, it is quite beneficial to be able to obtain certainty estimates explicitly, and not only expectations.

Gaussian Processes for Reinforcement Learning

A basic idea for learning functions without knowing the correct function class in advance are kernel methods. Especially interesting are GP kernels (Williams and Rasmussen 1996), as explained in Section II.3.3, because they are able to capture the certainty of the approximation. Rasmussen and Kuss (2004) use GPs for approximating the state value function (II.2.6) whereas Deisenroth and Rasmussen (2011) use a GP for learning the transition dynamics, rendering model-based reinforcement learning algorithms applicable, as explained in Section II.2.2.a.

A drawback of GP-based methods is that they have a computational complexity of $O(n^3)$ (Rasmussen 2004). They can not cope with very high sample counts and subset sampling strategies may be required. However, this can introduce approximation errors as we disregard already obtained data. Furthermore, GPs are subject to hyper-parameters that require tuning.

Deep Neural Networks for Reinforcement Learning

Mnih et al. (2015) propose to use a DNN (cf. Section II.3.4) to learn the Q-function (II.2.4), for discrete state-action spaces. The application of DNNs to continuous control problems was made possible by Lillicrap et al. (2015) and Gu et al. (2016), using a deterministic policy gradient algorithm (Silver et al. 2014). Schulman et al. (2015) also use a DNN as function approximator while incorporating the ideas of REPS into the policy update. Hence, it is possible to use complex policies with efficient updates rules that maintain exploration.

DNN-based algorithms achieve superior results, but at the cost of high sample counts and exceptional high computational cost. Hence, they are only of limited applicability outside of simulation-based settings. As we want to advance methods capable of learning from human feedback, we want to limit the number of required samples. Hence, we disregarded DNN-based approaches and leave the topic open for further work, as we describe in Section VII.4.2.

II.5 Preference Learning

Preference learning (PL) is about learning preference relations between objects. In contrast to techniques with numeric feedback, relational information is only binary. The expert only needs to determine if the relation holds for a given object pair or not. This is arguably easier than defining numeric feedback, as it is sufficient to determine if the preference relation holds or not, instead of defining a value, selected from a possibly infinite range. Preference relations are less exact and the contained amount of information is reduced. In case an exact evaluation is available, this is a drawback, but reliable information is often hard or impossible to obtain. Especially when obtaining feedback from human, non-professional users reliable, numeric reward can not be assumed, as explained in Section I.1. PL is especially suited for learning with humans in the loop, because humans are usually able to compare the quality of two options, as stated by Thurstone (1927).

II.5.1 Preferences[20]

A preference $x_i \succ x_j$ denotes that x_i is preferred over x_j for two choices x_i and x_j in a set of possible choices $\mathscr{X}$. Several short-hand notations can be used (Kreps 1988):

- $x_i \succ x_j$: The first choice is *strictly preferred*.
- $x_i \prec x_j$: The second choice is *strictly preferred*, i.e., $x_j \succ x_i$.
- $x_i \asymp x_j$: The choices are *indifferent*, meaning neither $x_i \succ x_j$ nor $x_j \succ x_i$ holds.

[20] This section is based on Kreps (1988, Chapter 1&2).

– $x_i \succeq x_j$: The first choice is *weakly preferred*, i.e., $x_j \succ x_i$ does not hold.

– $x_i \preceq x_j$: The second choice is *weakly preferred*, i.e., $x_i \succ x_j$ does not hold.

According to Kreps (1988), a relation must satisfy several properties to be considered a preference relation. In the following, we will present two of the stated axioms that allow us to simplify the problem:

Axiom 1: $x_i \succ x_j$ and $x_j \succ x_k$ imply $x_i \succ x_k$. This statement is equivalent to defining $\succ$ (and $\prec$) as transitive. Therefore, we can determine that *strict preference* induces an order or ranking over objects.

Axiom 2: For all x_i and x_j, exactly one relation $x_i \succ x_j$, $x_i \prec x_j$ or $x_i \asymp x_j$ holds. Stated differently, at most one strict preference relation holds for each object pair. The result of this property, together with the transitivity axiom, is a *weak order*. There are no incomparable pairs of objects.

Hence, *strict preference* and its negation is sufficient to describe any weak order. As a result, we use *preference* as a synonym for *strict preference*. Furthermore, indifference is usually disregarded as it is difficult to determine equivalent quality of objects, unless they are identical. Therefore, $\prec$ is equivalent to $\not\succ$ and it is sufficient to consider the $\succ$ relation.

II.5.2 Preference-based Ranking

Preference learning (PL) is a well-known technique outside the area of RL, where it is used to rank objects or labels. In difference to RL, the underlying objects are usually not generated by an MDP and it is mostly assumed to be a batch problem, where it is not possible to request new preferences. Different settings for PL have been defined, depending on the available information and the task. We follow the unifying terminology by Fürnkranz and Hüllermeier (2010b), allowing us to relate the PL tasks to RL problems in Chapter III.

Object Ranking

Within the area of object ranking (Kamishima et al. 2010), the task is to rank the elements of an object set $\mathscr{X}$. The elements $x \in \mathscr{X}$ in this set are usually described by a feature vector $\boldsymbol{\phi}(x)$. Feedback is obtained as a set of pairwise preferences $x_i \succ x_j \in \mathscr{X} \times \mathscr{X}$. The task is now to find a model $f(\tilde{\mathscr{X}})$ that gets a subset $\tilde{\mathscr{X}} \subseteq \mathscr{X}$ as input and returns an ordering. The performance of the model is determined by comparing the order of the returned set with the order induced by the preferences.

Instance Ranking

In the instance ranking setting (Fürnkranz et al. 2009; Fürnkranz and Hüllermeier 2010b), the objects are subject to a single label $y \in \mathcal{Y}$ each where $\mathcal{Y}$ is a finite set with an order, e.g $y_1 \succ y_2 \succ \cdots \succ y_n$. The feedback is given in terms of the label class an objects belongs to. The task is to find a model $f(\tilde{\mathcal{X}})$ that orders a given set of objects. This task can be reduced to the object ranking setting, but knowing the number and ranking distance between labels (and therefore objects) is usually beneficial for learning a good model. Instance ranking is closely related to ordinal classification, but the label order is assumed to be an approximation of an unknown, underlying ranking over all objects.

Label Ranking

The third setting differs from the others, as the task is not to rank the objects, but labels associated with them (Hüllermeier et al. 2008; Vembu and Gärtner 2010). An object is associated with multiple labels $y \in \mathcal{Y}$ and preferences take the form of $y_1 \succ_{x_i} y_2$. Each preference is only valid for the given object x_i. The objective is to find a ranking model $f(x) \Rightarrow \mathcal{Y}_x$, taking instances $x \in \mathcal{X}$ as input, but the output is an ordering of $\mathcal{Y}$. The models performance can also be determined by comparing the returned label order with the ranking induced by the preferences.

II.5.3 Approaches to Preference Learning

Different approaches to preference-based ranking have been studied and they can be mainly categorized into two types. The first approach is to learn a numeric evaluation function where the resulting evaluation correlates with the objects rank, as we explain in Section II.5.3.a. The alternative, that we describe in Section II.5.3.b, is to learn a (binary) model for each preference relation that determines if the according relation is expected to be satisfied by a given object or label pair. Building a single model can use the given training data more efficiently in case multiple preference symbols are available as it is not required to partition the data depending on the preference relation. Furthermore, a numeric evaluation can be beneficial as it allows to derive more fine-grained information that allows to determine degrees of preference, e.g., in terms of value difference. This can be beneficial in interactive settings like RL where it is required to determine for which object pair to request a new preference. However, only binary ground truth is observable and a surrogate loss that maps the numeric values to binary feedback is required. Binary preference relation models directly minimize the binary loss, but degrees of preference are not directly available.

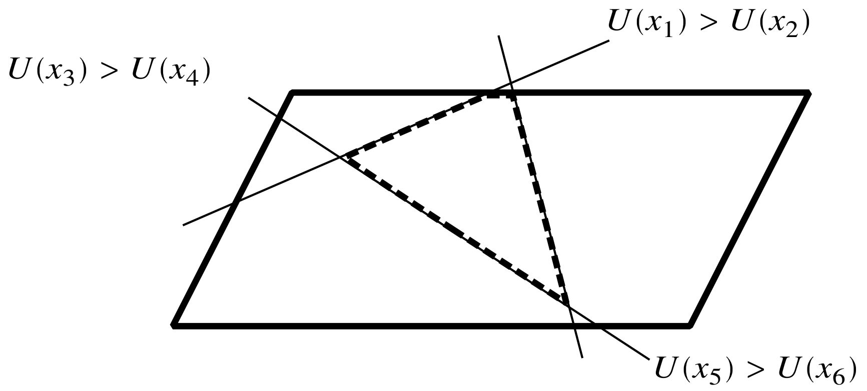


Figure II.5.6.: A utility space, constraint by three preferences

II.5.3.a Learning Utility Functions[22]

Formally, a utility function U is a function that assigns numeric scores to each object $U : \mathscr{X} \to \mathbb{R}$ or object/label pair $U : \mathscr{X} \times \mathscr{Y} \to \mathbb{R}$. Preferences can than be considered constraints for the space of utility functions, e.g., $x_i \succ x_j \Leftrightarrow U(x_i) > U(x_j)$ with the task to find an utility function not violating the given constraints. Alternatively, it is possible to define loss functions, e.g., $L(\{x_i \succ x_i\}, U) \to \mathbb{R}$, and minimize the loss over all given preferences. As it is not always possible to satisfy all preferences, due to noise or restrictions of the function class (cf. Section II.3), defining loss functions is usually more practical. Loss-based utility learning can also solve the problem of choosing a single utility function out of the space of valid models. Consider Figure II.5.6 where the utility space (bold rectangular) is constraint by three preferences (thin lines), but the space of valid utility functions (dashed area) is still covering an area, possibly containing multiple solutions. A continuous loss function may impose loss differences in the optimal area and result in a single, optimal solution. It is possible to rephrase the constraint learning problem as a loss-based learning problem by using indicator functions for defining the loss, e.g., $U(x_i) > U(x_j) \Leftrightarrow L(x_i \succ x_j) = \mathbb{1}\big(U(x_i) - U(x_j) > 0\big)$, but non-continuous loss functions are usually difficult to optimize. Furthermore, this definition does not induce differences in the optimal area of Figure II.5.6. Therefore, continuous functions like the preference-based hinge loss (Joachims 2002)

$$\begin{aligned} &L(x_i \succ x_j) \geq 1 - \big(U(x_i) - U(x_j),\big) \\ &\text{s.t. } L(x_i \succ x_j) > 0, \end{aligned} \tag{II.5.45}$$

are commonly used, but they introduce a tradeoff between satisfying multiple preferences and maximizing the utility difference of single preferences. In case there exists a solution with

[22] This section is based on Aiolli and Sperduti (2010) and Waegeman and De Baets (2010).

$L(x_1 \succ x_2) + L(x_3 \succ x_4) > L(x_5 \succ x_6)$, it may be beneficial to maximize the utility difference $U(x_5) - U(x_6)$ while violating the stated preferences $x_1 \succ x_2$ and $x_3 \succ x_4$. Hence, defining well suited loss functions for preference tasks is a problem itself.

II.5.3.b Learning Preference Relations[24]

When learning preference relations, the task is to train a binary model that predicts if the preference relation holds or not, e.g., $M(x_i, x_j) = \{\succ, \not\succ\}$. In general, the problem is equivalent to classification tasks (Fürnkranz and Hüllermeier 2010c), with the goal to minimize the number of incorrectly predicted preference relations, concerning a given test set. When applying preference relation models to label ranking tasks (cf. Section II.5.2), it can be beneficial to learn a separate model for each label pair $M_{y_i \succ y_j}(x)$ instead of learning a unified model $M(y_i, y_j, x)$ (Fürnkranz et al. 2009). Application of this method depends on the size and the finiteness of the label set.

II.5.3.c Function Approximation in Preference Learning

Utility functions, as introduced in Section. II.5.3.a, as well as the preference relation models of Section II.5.3.b can also be defined using a *function approximation* (FA), as introduced in Section II.3. In the case of utility functions, this is straight forward by setting $U : f$. Such a continuous function can also be turned into a binary, preference relation model, e.g., if the returned value exceeds a certain threshold, a positive prediction is returned. Alternatively, non-continuous predictors can be used instead. This relates learning preference relations to classification by defining classes for each preference predicate (Fürnkranz and Hüllermeier 2010c). Many models for classification have been proposed, but as they are not relevant for this thesis, we do not cover those techniques in detail and forward the interested reader to Flach (2012). An exception are neural networks that are able to perform the thresholding implicitly while still learning a continuous function, as explained in Section II.3.4.

[24] This section is based on Hüllermeier et al. (2008).

Chapter III

Preference-based Reinforcement Learning[26]

[26] This chapter is based on Wirth and Fürnkranz (2013d) and Wirth, Akrour, Neumann, and Fürnkranz (2017)

The assumption of a numeric reward for *Markov decision processs* (MDPs), as introduced in Section II.2.1, does not hold in many domains, as explained in Section I.1. The application of *reinforcement learning* (RL) is currently limited to specific domains and requires experts for implementing a solution. Alternatives have already been proposed, like *inverse reinforcement learning* (IRL) (Zhifei and Meng Joo 2012) or ordinal reward functions (Weng et al. 2013), but they still require expert knowledge, are restricted to specific domains or assume a specific structure of the reward function. Preference-based feedback can be seen as a generalization of various feedback structures, as will be explained in Section III.2.2. Furthermore, preference-based feedback is arguably the least demanding type of feedback, from a human point of view.

Several approaches to *preference-based reinforcement learning* (PBRL) have been proposed, but they are subject to a wide range of assumptions and design choices. It is currently unclear which problems have to be solved and how the solution techniques are related to each other. A general, modular overview of PBRL is missing. Therefore, we present a unified description of PBRL and list the subproblems explicitly, as well as the according design principles that can be encountered in the literature. This allows us to point out differences and similarities between the methods explained throughout this thesis and related methods. We make use of the unified description to elaborate on the formal background of each method. We also discuss the related assumptions and requirements that have to be considered for our approaches and other PBRL algorithms.

III.1 Markov Decision Process with Preferences

We will call the formal, mathematical framework for preference-based reinforcement learning an *Markov decision process with preferences* (MDPP). It is based upon a MDP, as introduced in Section II.2.1, but replaces the reward function r with pairwise preference function ρ. An MDPP is defined by a sextuple $(S, A, \mu, \delta, \gamma, \rho)$.

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 $\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j$. 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

$$\zeta = \{\zeta_i\} = \{\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2}\}_{i=1\ldots N}, \tag{III.1.1}$$

which has been sampled using ρ. Υ 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 possible 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.

III.2 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 reformulate the objective for a single preference as maximization problem concerning the difference of the probabilities for the trajectories $\boldsymbol{\tau}_1$ and $\boldsymbol{\tau}_2$, e.g.,

$$\{\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2\} \Leftrightarrow \pi^* = \arg\max_{\pi} \left(\Pr^{\pi}(\boldsymbol{\tau}_1) - \Pr^{\pi}(\boldsymbol{\tau}_2)\right), \tag{III.2.2}$$

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., $L(\pi, \{\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2\}) = -\left(\Pr^{\pi}(\boldsymbol{\tau}_1) - \Pr^{\pi}(\boldsymbol{\tau}_2)\right)$ The domain of L differs, depending on the approach. Yet, in the multi-preference case, satisfying one preference might interfere with satisfying 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 single objective or loss function, resulting in a loss *vector* $\mathbf{L}(\pi, \zeta)$. 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* $\mathscr{L}$, as introduced by Duchi et al. (2010),

$$\mathscr{L}(\pi, \zeta) = \sum_{i=1}^{|\zeta|} \alpha_i L(\pi, \zeta_i), \tag{III.2.3}$$

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 usually want to maximize the realization probability of undominated trajectories, as we are less interested in suboptimal trajectories. Therefore, preferences concerning the least dominated trajectories should have the highest weight. However, in other settings, it might be more relevant to realize preferences involving high risk trajectories for preventing major accidents. Hence, different weights α_i are required for different settings. However, all available methods 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. The definition and minimization of the single preference loss $L(\pi, \zeta_i)$ is algorithm-specific and is explained in the Section III.3.

III.2.1 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 Figure III.2.1. The learning usually starts with a set of trajectories, either predefined or derived from a given sampling 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 and the possibilities are explained in Section III.3.1. For computing a policy π based on the observed preferences, 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 estimates a relational model for the expert's preferences, and

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

The dashed path in Figure III.2.1 relates to policy learning. Both other approaches, learning a preference model or a utility function, are *surrogate-based* in that they learn a surrogate from

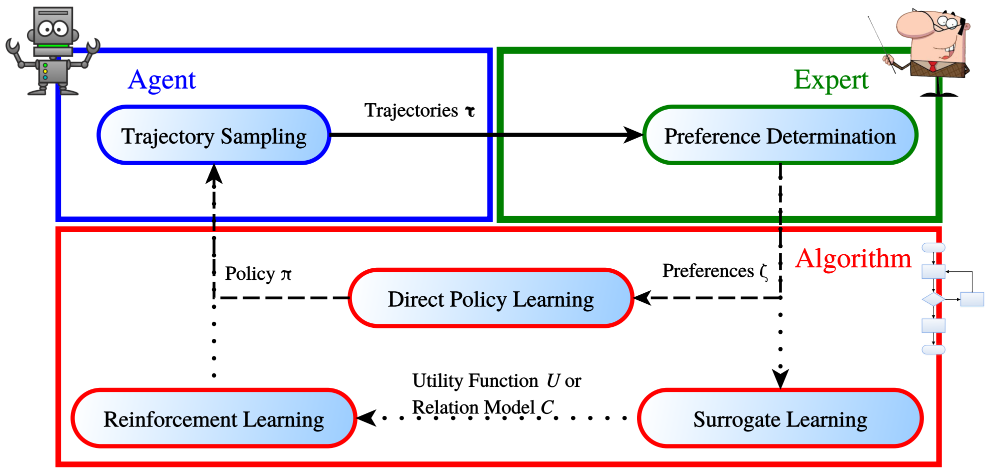


Figure III.2.1.: PBRL: Learning policies from preferences via direct (dashed path) and surrogate-based (dotted path) approaches.

the obtained preferences, which is in turn used for deriving a maximizing policy (cf. the dotted path in Figure III.2.1). For this optimization, often conventional reward-based policy optimization algorithms can be used (cf. Section II.2.3). All three cases are explained in more detail in Section III.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 we describe in Section III.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.

III.2.2 Related Problem Settings

Preference-based reinforcement learning is closely related to several other learning settings, which we will briefly discuss in this section. We also show that preference-based feedback can be assumed to be the least demanding type of feedback, with respect to the expert's cognitive load. Furthermore, preferences can be seen as a generalization of several other forms of feedback, although they may not contain the same amount of information.

III.2.2.a Learning with Advice

Learning with advice (Knox and Stone 2010, 2012; Griffith et al. 2013; Torrey and Taylor 2013) differs 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. Advice usually concerns possible action or state choices. Hence, the evaluator must be able to determine an expected outcome for a state or action choice, which usually requires knowledge over an approximate, optimal policy. This is comparable to defining state or action preferences with long-term expectation, as described in Section III.3.1.

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 numeric reward signal. Such advice is similar to state preferences, but the states are demonstrated by the expert and not evaluated in a pairwise manner. By using rules, demonstrated states or actions, the advised choice is assumed to be preferable over all possible alternatives. Therefore, the expert must compare the choice with a high number of alternatives. In general, learning from advice requires domain knowledge.

III.2.2.b Ordinal Feedback

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. Ordinal feedback requires knowledge over a coarse grading, hence this type of feedback is more demanding than only defining preferences. Daniel et al. (2014) assume a numeric rating of trajectories, which is even more demanding than ordinal grading without explicit values. 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) introduce a method that can use preferences as well as coarse, numeric feedback. The presented experiments indicate that mixing preference feedback and numeric feedback leads 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.

III.2.2.c 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 is maximizing an internal reward function and the goal is to identify this reward function. In this setting, the trajectories are supplied by the expert, not by the algorithm. Therefore, the expert must be familiar with reasonable policies to create trajectories that are realizable. There are also no explicit pairwise preferences, yet they can be derived implicitly as the demonstrated sequences can be assumed to be preferred over all other trajectories. Many IRL approaches are based on implicitly defined preferences and can be applied to PBRL by adapting them to explicit preferences and maximize the obtained reward signal (see Section III.3.2.c). Implicit preferences are also used by Knox and Stone (2009), however, the method 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 (Ziebart et al. 2008; Rothkopf and Dimitrakakis 2011) relax this assumption and allow sub-optimal demonstrations, based on a parametric optimality prior like a softmax function. Alternatively, an optimality score can be computed (Dimitrakakis and Rothkopf 2011). However, in general the demonstrations are assumed to be optimal or near optimal. Hence, the expert must be familiar with the expected, optimal solution for a given task, requiring domain knowledge. 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.

III.2.2.d Feedback-less 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 always degrades over time, which is reasonable in some domains. They apply their method to a robot following task, where a robot needs to follow a second robot based on visual sensor 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.

III.2.2.e 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 such a dueling bandit problem, when viewing the arms as policies. Busa-Fekete et al. (2013, 2014) realized that idea, which we discuss in Section III.3.2.a. Approaches that learn a preference model from action preferences also share similarities with this setting (cf. Section III.3.2.b). 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 such as machine translation (Sokolov et al. 2016).

III.3 Design Principles for PBRL

Preference-based reinforcement learning algorithms are subject to different design choices, which we will review in this section. Our first design choice is what type of preferences are considered, i.e., *trajectory*, *action* or *state* preferences. These different types introduce different levels of complexity for the expert as well as the algorithm (Section III.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 (Section III.3.2). The learned representation is directly connected to the problem of assigning preferences to states and actions, given a possibly delayed feedback signal. We explain this problem and possible approaches in Section III.3.3. The next design principle is how to collect new feedback, i.e., how new trajectories are generated and selected to obtain new preference feedback from the expert, which we discuss in Section III.3.4. Having obtained a representation, we need to derive an optimized policy. In Section III.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 Section III.3.6. Section III.3.7 discusses a tabular overview of all algorithms, according to the design principles.

III.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. They impose different challenges for the expert and the

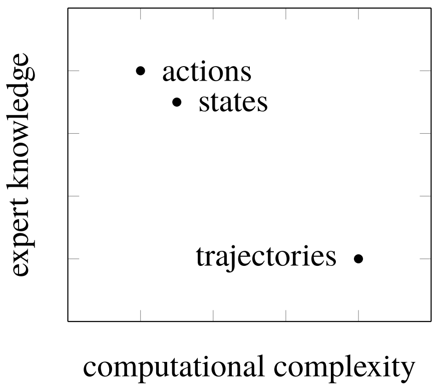


Figure III.3.2.: Complexity trade-off for different preference types

.

algorithm, as illustrated in Figure III.3.2. Action and state preferences depend on model or policy knowledge and can be difficult to aggregate. Trajectory preferences define feedback over multiple decisions made along the sequence and it is computationally complex to determine the relevant decision points. We discuss the specific problems for the expert and the computational complexity in more detail in the following subsections.

III.3.1.a Action Preferences

An *action preference* compares two actions for the same state, e.g, that in state s, action a_i should be preferred to a_j.

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 is 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, he 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 assumes 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.

III.3.1.b State Preferences

A *state preference* $s_i \succ s_j$ determines that state s_i is preferred to state s_j. A state preference is equivalent to saying that there is an action in state s_i that is preferred to all actions available in state s_j.

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 we analyze in Chapter V (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) who try to solve 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 we explain in Section III.3.2.c.

III.3.1.c Trajectory Preferences

A *trajectory preference* $\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j$ specifies that the trajectory $\boldsymbol{\tau}_i$ should be preferred over the dominated trajectory $\boldsymbol{\tau}_j$. 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*, which we discuss in Section III.3.3. This assignment problem is particularly difficult if not all trajectories are starting in the same state. Trajectories with different start states are solutions to different initial situations. Hence, the preference could also be attributed to the initial situation, not only to the applied policy. In practice, no algorithm known to the author is capable to deal 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 that states a_i is preferred to a_j in state s is equivalent to the trajectory preference $\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j, \boldsymbol{\tau}_i = \{s, a_i\}, \boldsymbol{\tau}_j = \{s, a_j\}$. The case of a state preference $s_i \succ s_j$ can be reformulated as indicated above, yielding the set of trajectory preferences $\{\forall a_j \in A(s_j) : (s_i, \pi^*(s_i)) \succ (s_j, a_j)\}$.

III.3.2 Defining the Learning Problem

As discussed in Section III.2.1, 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 Section III.1. 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 Section III.2

III.3.2.a 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 $\pi(a \mid s, \boldsymbol{\omega})$ (II.2.19) is used for defining the policy distribution $\Pr(\pi \mid \zeta)$. Algorithm 2 shows how to collect the preferences. The basic strategy is two sample two policies, described by a parameter vector, from the posterior distribution $\Pr(\pi|\zeta)$ induced by the preferences. Each policy pair is used to create one or more trajectories, defining potential preference queries $\tilde{\zeta}$ by pairing the resulting trajectories. One of multiple selection criteria, that we discuss in Section III.3.4.c, 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.

Algorithm 2 Policy Likelihood

```
Require: prior $\Pr(\pi)$, step limit $k$, sample limit $m$, iteration limit $n$
 1: $\zeta = \emptyset$                                          ▷ Start with empty preference set
 2: for $i = 0$ to $n$ do
 3:     $\tilde{\zeta} = \emptyset$                              ▷ Clear potential preference queries
 4:     for $j = 0$ to $m$ do
 5:         $s \sim \mu(s)$                                      ▷ Draw initial state $s$
 6:         $\pi_1, \pi_2 \sim \Pr(\pi|\zeta)$                   ▷ Draw two policies from posterior
 7:         for $j = 0$ to $n$ do
 8:             $\boldsymbol{\tau}^{\pi_1} = \textsc{rollout}(s, \pi_1, k)$     ▷ Create k-step trajectory using policy $\pi_1$
 9:             $\boldsymbol{\tau}^{\pi_2} = \textsc{rollout}(s, \pi_2, k)$     ▷ Create k-step trajectory using policy $\pi_2$
10:             $\tilde{\zeta} \leftarrow \tilde{\zeta} \cup (\boldsymbol{\tau}^{\pi_1}, \boldsymbol{\tau}^{\pi_2})$   ▷ Add trajectory pair to potential preference queries
11:         end for
12:     end for
13:     $(\boldsymbol{\tau}_1, \boldsymbol{\tau}_2) = \textsc{selectQuery}(\tilde{\zeta})$        ▷ Select a preference query
14:     $\zeta \leftarrow \zeta \cup \textsc{obtainTrajectoryPreferences}(\boldsymbol{\tau}_1, \boldsymbol{\tau}_2)$   ▷ Query expert
15: end for
16: return $\arg\max_\pi \Pr(\pi|\zeta)$                          ▷ Return maximum-a-posterior
```

The likelihood $\Pr(\pi|\zeta)$ 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.,

$$\Pr(\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2} \mid \pi) = \Phi\left(\frac{\mathbb{E}[d(\boldsymbol{\tau}_{i1}, \boldsymbol{\tau}^\pi)] - \mathbb{E}[d(\boldsymbol{\tau}_{i2}, \boldsymbol{\tau}^\pi)]}{\sqrt{2}\sigma_p}\right), \tag{III.3.4}$$

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,

$$\Pr(\pi \mid \zeta) \propto \Pr(\pi) \prod_{i=1}^{|\zeta|} \Pr(\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2} \mid \pi), \tag{III.3.5}$$

and the posterior is approximated using *Markov chain Monte Carlo* (MCMC) simulation (Andrieu et al. 2003). This technique 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, but Euclidean distances are hard to use in many domains, such as for higher-dimensional continuous state spaces. Furthermore, if the state

Algorithm 3 Policy Ranking

```
Require: candidate policies $\Pi_0$, step limit $k$, sample limit $m$, iteration limit $n$
 1: for $i = 0$ to $n$ do
 2:     $\zeta = \emptyset$
 3:     for 0 to $m$ do
 4:         $\pi_1, \pi_2 = \textsc{selectPolicies}(\Pi_i, \zeta)$    ▷ Select policies to compare
 5:         $s \sim \mu(s)$    ▷ Draw initial state $s$
 6:         $\boldsymbol{\tau}^{\pi_1} = \textsc{rollout}(s, \pi_1, k)$    ▷ Create k-step trajectory using policy $\pi_a$
 7:         $\boldsymbol{\tau}^{\pi_2} = \textsc{rollout}(s, \pi_2, k)$    ▷ Create k-step trajectory using policy $\pi_2$
 8:         $\zeta \leftarrow \zeta \cup \textsc{obtainTrajectoryPreferences}(\boldsymbol{\tau}^{\pi_1}, \boldsymbol{\tau}^{\pi_2})$    ▷ Query expert
 9:     end for
10:     $\Pi_{i+1} = \text{EDPS}(\Pi_i, \zeta)$    ▷ Compute candidate policies, based on preferences
11: end for
12: return $\arg\max_\pi \sum_{\pi,\pi' \in \Pi_n} \Pr(\pi \succ \pi')$    ▷ Return highest ranked policy
```

space is not continuous but only a set of symbols, such a distance measure is not applicable at all. Defining a parametric policy also requires domain knowledge, although, in many domains such as robotics, good parametric policy representations are often known (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 3. A confidence bound method that we discuss in Section III.3.4.a 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.,

$$\Pr(\pi_1 \succ \pi_2) \Leftrightarrow \mathbb{E}_{\boldsymbol{\tau}^{\pi_1}, \boldsymbol{\tau}^{\pi_2}}[\rho(\boldsymbol{\tau}^{\pi_1} \succ \boldsymbol{\tau}^{\pi_2})]. \tag{III.3.6}$$

For an observed set of preferences ζ, the expectation in (III.3.6) can be approximated as

$$\Pr(\pi_1 \succ \pi_2) \approx \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(\boldsymbol{\tau}_{i1}^{\pi_1} \succ \boldsymbol{\tau}_{i2}^{\pi_2}). \tag{III.3.7}$$

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

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

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).

III.3.2.b 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 4, *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 $C_{ij}(s)$ for each action pair a_i, a_j 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

Algorithm 4 Preference-based Approximate Policy Iteration

```
Require: initial policy $\pi_0$, iteration limit $m$, state sample limit $k$, rollout limit $n$
 1: for $i = 0$ to $m$ do
 2:     for 0 to $k$ do
 3:         $s \sim \mu(s)$                                   ▷ Sample $k$ states per iteration
 4:         $\Upsilon = \emptyset, \zeta = \emptyset$
 5:         for $\forall a \in A(s)$ do
 6:             for 0 to $n$ do
 7:                 $\Upsilon \leftarrow \Upsilon \cup$ ROLLOUT$(s, a, \pi_i)$    ▷ Create trajectory, starting with $(s, a)$
 8:             end for
 9:         end for
10:         $\zeta \leftarrow$ OBTAINACTIONPREFERENCES$(\Upsilon)$             ▷ Query expert
11:     end for
12:     $C^{\pi_i}$ = COMPUTEPREFERNCEMODEL$(\zeta)$          ▷ Compute the preference function C
13:     $\pi_{i+1}$ = COMPUTEPOLICY$(C^{\pi_i})$                     ▷ Compute greedy policy
14: end for
15: return improved policy $\pi_m$
```

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

$$k(s,a) = \sum_{\forall a_i, a_j \in A(s), a_j \neq a} C(a_i \succ a_j \mid s) = \sum_{\forall a_i, a_j \in A(s), a_j \neq a} C_{ij}(s) \tag{III.3.8}$$

correlates with $\rho\left(a_i \succ a_j \mid s\right)$ 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 (III.2.2).

In Chapter IV (Wirth and Fürnkranz 2013a, 2013b), we also use action preferences in a similar way. However, C is treated 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.

III.3.2.c 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. Furthermore, 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. Consequently, we call this function a *utility function* to distinguish it from a fixed reward function[27].

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

$$\pi^* = \max_{\pi} \mathbb{E}_{\mathrm{Pr}^{\pi}(\boldsymbol{\tau})}[U(\boldsymbol{\tau})], \tag{III.3.9}$$

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 5 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 more trajectories from a policy and derive queries by exhaustive or selective comparison (cf. Section III.3.4.a & III.3.4.b). 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.

[27] 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 5 Utility-based PbRL

```
Require: initial policy $\pi_0$, iteration limit $m$, state sample limit $k$, rollout limit $n$
 1: $\zeta = \emptyset$
 2: for $i = 0$ to $m$ do
 3:     $\Upsilon = \emptyset$
 4:     for 0 to $k$ do
 5:         $s \sim \mu(s)$                                   ▷ Sample $k$ states per iteration
 6:         for 0 to $n$ do
 7:             $\Upsilon \leftarrow \Upsilon \cup$ ROLLOUT$(s, \pi_i)$    ▷ Create trajectory, starting with state $s$
 8:         end for
 9:     end for
10:     $(\boldsymbol{\tau}_1, \boldsymbol{\tau}_2) =$ CREATEQUERY$(\Upsilon)$    ▷ Create a preference query
11:     $\zeta \leftarrow \zeta \cup$ OBTAINTRAJECTORYPREFERENCES$(\boldsymbol{\tau}_1, \boldsymbol{\tau}_2)$    ▷ Query expert
12:     $U^{\pi_i} =$ COMPUTEUTILITYFUNCTION$(\zeta)$    ▷ Compute the utility function
13:     $\pi_{i+1} =$ COMPUTEPOLICY$(U^{\pi_i})$    ▷ Compute a policy
14: end for
15: return improved policy $\pi_m$
```

Linear utility functions

The most common approach is to use utility functions that are linear in a feature vector, as introduced in Section II.3.1. We may use state action features $\boldsymbol{\phi}(s, a)$ resulting in a utility $U(s, a) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s, a)$, or trajectory features $\boldsymbol{\phi}(\boldsymbol{\tau})$ yielding $U(\boldsymbol{\tau}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{\tau})$. In order to find such a linear utility function, we can define a loss function $\mathscr{L}$ which is given by the (weighted) sum of the pairwise disagreement loss L, i.e.,

$$\mathscr{L}(\boldsymbol{\theta}, \zeta) = \sum_{i=1}^{|\zeta|} \alpha_i L(\boldsymbol{\theta}, \zeta_i), \tag{III.3.10}$$

as described in Section III.2. Using the linearity of the utility function, we can define the utility difference

$$d(\boldsymbol{\theta}, \zeta_i) = \boldsymbol{\theta}^T(\boldsymbol{\phi}(\boldsymbol{\tau}_{i1}) - \boldsymbol{\phi}(\boldsymbol{\tau}_{i2})), \tag{III.3.11}$$

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

$$L^{\succ}(\boldsymbol{\theta}, \zeta_i) = \mathbb{1}(d(\boldsymbol{\theta}, \zeta_i) \leq \epsilon), \tag{III.3.12}$$

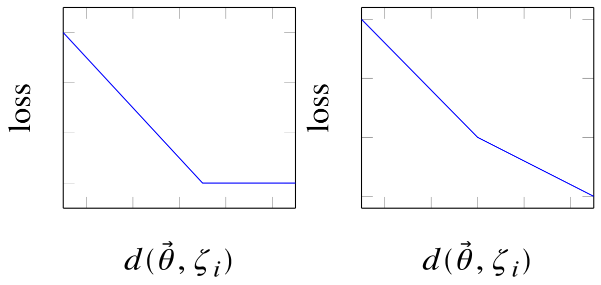
Name	Loss Equation	Reference
hinge/ranking SVM	$L^{\succ}(\boldsymbol{\theta}, \zeta_i) = \max(0, 1 - d(\boldsymbol{\theta}, \zeta_i))$	Chapter V, Zucker et al. (2010), Akrour et al. (2011, 2012), and Wirth and Fürnkranz (2012, 2015)
IRL	$L^{\succ}(\boldsymbol{\theta}, \zeta_i) = c(d(\boldsymbol{\theta}, \zeta_i))$ $c(x) = \begin{cases} -2x & \textbf{if } x \leq 0 \\ -x & \textbf{otherwise} \end{cases}$	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	$p_{\text{sig}\,\boldsymbol{\theta}}(\zeta_i) = \frac{1}{1+\exp(-m \cdot d(\boldsymbol{\theta},\zeta_i))}$	Chapter V, VI, Christiano et al. (2017)
0/1	$p_{01\,\boldsymbol{\theta}}(\zeta_i) = \mathbb{1}(d(\boldsymbol{\theta}, \zeta_i) < 0)$	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) and Akrour et al. (2014)

Table III.1.: Loss functions for linear utility functions

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 satisfaction) for trading off multiple preferences. Therefore, continuous loss functions are more often used. A number of continuous loss functions have been presented in the literature.

Table III.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 (Chapter V, Zucker et al. 2010; Akrour et al. 2011, 2012; Runarsson and Lucas 2012; Wirth and Fürnkranz 2012; Runarsson and Lucas 2014; Wirth and Fürnkranz 2015) define the loss function $L^{\succ}(\boldsymbol{\theta}, \zeta_i)$ as hinge loss (Fig. III.3.3a) that can be optimized using SVM ranking algorithms (Chapter V, Herbrich et al. 1999; Joachims 2002), others (Wirth et al. 2016) as a piecewise linear loss function (Fig. III.3.3b) inspired by IRL 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



(a) hinge (b) IRL

Figure III.3.3.: Shape of loss functions

modeled as likelihood functions $p_{\boldsymbol{\theta}}(\zeta_i)$ for the preferences. In this case, we have to optimize the log likelihood, i.e.,

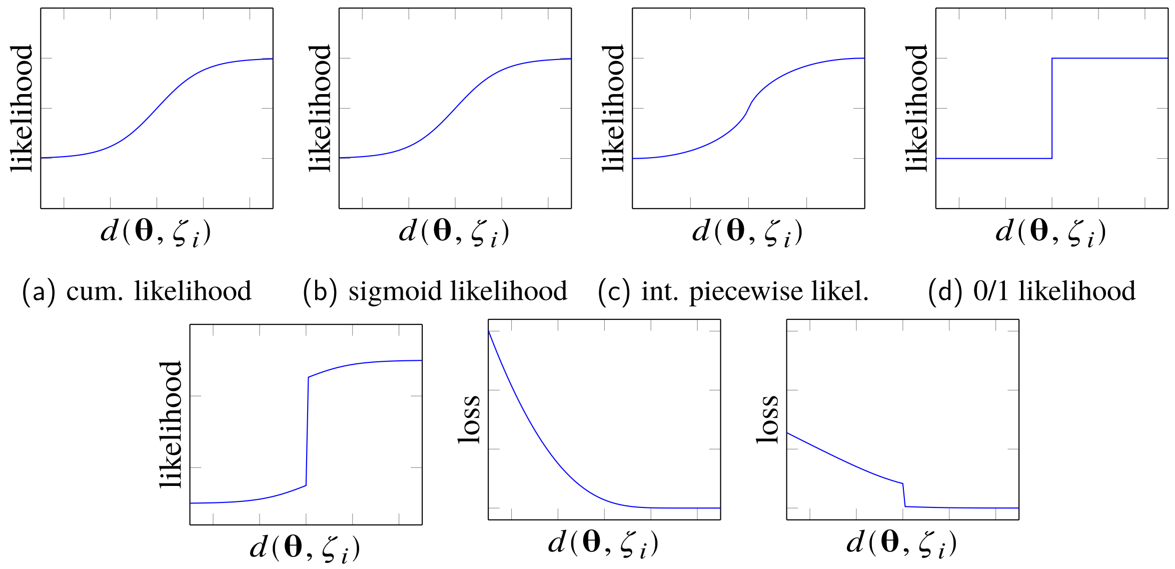
$$\mathscr{L}(\boldsymbol{\theta}, \zeta) = -\log \prod_{i=1}^{|\zeta|} p_{\boldsymbol{\theta}}(\zeta_i) = -\sum_{i=1}^{|\zeta|} \log\left(p_{\boldsymbol{\theta}}(\zeta_i)\right) = \sum_{i=1}^{|\zeta|} \mathscr{L}(\boldsymbol{\theta}, \zeta_i),$$

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

$$\Pr(\boldsymbol{\theta} \mid \zeta) \propto \Pr(\boldsymbol{\theta}) \prod_{i=1}^{|\zeta|} p_{\boldsymbol{\theta}}(\zeta_i). \tag{III.3.13}$$

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 in Section VI.2 (Wirth et al. 2016) of this dissertation, where we use *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 method that does not use an explicit cost function for obtaining the utility is by Jain et al. (2013) and Jain et al. (2015). It uses the preferences to compute a gradient for the parameter vector of the utility function, i.e., $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha(\boldsymbol{\phi}(\boldsymbol{\tau}_{i1}) - \boldsymbol{\phi}(\boldsymbol{\tau}_{i2}))$, 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 $\mathscr{L}(\boldsymbol{\theta}, \zeta)$ is related to the policy loss $\mathscr{L}(\pi, \zeta)$ (see Section III.2), as the policy is subject to the system



(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 III.3.4.: Shape of likelihood functions and the according negative log likelihood

dynamics whereas the utility is not. Nevertheless, it is assumed that maximizing the expected utility (III.3.9) 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.,

$$L^{\succ}(U, \zeta_i) = \|d(U, \zeta_i) - k_i\|_2, \qquad \text{(III.3.14)}$$

This loss function can be minimized by any regression algorithm. For example, Gritsenko and Berenson (2014) use M5P (Wang and Witten 1997).

The rank values are subject to a somewhat arbitrary scale (e.g., the difference between mis-predictions 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 $-\sum_{i=1}^{|\zeta|} \log\left(p_{\boldsymbol{\theta}}(\zeta_i)\right)$ 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 backpropgation techniques for minimizing the loss.

The approach in this dissertation (Chapter VI; 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 Section III.3.3.c.

III.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.

III.3.3.a 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 (cf. Section II.2.3). We use state-preferences $s_{i1} \succ s_{i2}$ in Chapter V (Wirth and Fürnkranz 2012, 2015), as proposed by Runarsson and Lucas (2012) and Runarsson and Lucas (2014). The preferences 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 Table III.1. Using the utility function, we can define the policy

$$\pi^*(a \mid s) = \mathbb{1}\left(a = \arg\max_{a'} \mathbb{E}_{\delta}[\delta(s' \mid s, a) U(s')]\right), \tag{III.3.15}$$

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.

III.3.3.b Return-based Utility

Many techniques 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; Gritsenko and Berenson 2014; Jain et al. 2015; Kupcsik et al. 2015) as defined by a return-utility function $U(\boldsymbol{\tau})$. The return utility function $U(\boldsymbol{\tau}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{\tau})$ is typically defined by a low-dimensional trajectory feature space $\boldsymbol{\phi}(\boldsymbol{\tau})$.

Zucker et al. (2010), Jain et al. (2013), Gritsenko and Berenson (2014), and Jain et al. (2015) assume a domain-specific trajectory feature space, whereas Akrour et al. (2011, 2012) cluster the state space using ϵ-means clustering (Duda and Hart 1973) for defining trajectory features. The trajectory features are given by the relative number 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 $\boldsymbol{\tau} = [\mathbf{s}, \boldsymbol{\omega}]$ with a context vector $\mathbf{s}$ 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 $U(\boldsymbol{\tau})$ (cf. Section III.3.2.c).

III.3.3.c 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) and Akrour et al. (2014) and Wirth et al. (2016) (cf. Section VI.2.1) use the *feature averages* (Ng and Russell 2000) along a trajectory

$$\boldsymbol{\phi}(\boldsymbol{\tau}) = \sum_{t=0}^{|\boldsymbol{\tau}|} \gamma^t \boldsymbol{\phi}(s_t, a_t), \tag{III.3.16}$$

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

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

with $U(s_t, a_t) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s_t, a_t)$, 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, we use a kernel function in Chapter VI 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

$$\begin{aligned} U(s,a) &= \sum_{s' \in S} \left(\delta(s' \mid s, a) \left(U(s,s') + \max_{a' \in A} \sum_{s'' \in S} \delta(s'', s', a') U(s' \mid s'') \right) \right), \\ U(s,s') &= \boldsymbol{\theta}^T \boldsymbol{\phi}(s,s'), \end{aligned} \tag{III.3.17}$$

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 frameworks 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 Section III.3.2.c, 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 (Chapter VI; 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.

III.3.3.d Intermediate Action Preferences

We extend the action preference algorithm of Fürnkranz et al. (2012) in Chapter IV (Wirth and Fürnkranz 2013a) 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 a (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, as we explain in Section IV.3.

This approach may converge faster than the one of Fürnkranz et al. (2012) using a lower number of preferences. However, the speedup comes at the cost of possible approximation errors due to possibly incorrect, heuristic estimates of the preferences. In Section IV.4 (Wirth and Fürnkranz 2013b), we 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 number of preferences is sufficient. However, many domains do not satisfy this uniform reward assumption.

III.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* (Jong and Stone 1976; Howard 1988).

Several algorithms disregard this problem and only work with an initial set of preferences (Chapter V; Zucker et al. 2010; Runarsson and Lucas 2012; Sugiyama et al. 2012; Wirth and Fürnkranz 2012; Gritsenko and Berenson 2014; Runarsson and Lucas 2014; Wirth and Fürnkranz 2015). 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 number 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.

III.3.4.a 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.

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 methods (Chapter IV, VI; Busa-Fekete et al. 2013; Jain et al. 2013; Wirth and Fürnkranz 2013a, 2013b; Busa-Fekete et al. 2014; Jain et al. 2015; 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 number 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) and Jain et al. (2015) use *rapidly-exploring random trees* (LaValle and Kuffner 1999) whereas we employ approximate policy iteration (Dimitrakakis and Lagoudakis 2008) with an EXP3-like exploration method (Auer et al. 1995) in Chapter IV (Wirth and Fürnkranz 2013a, 2013b).

We also use *relative entropy policy search* (REPS) (cf. Section II.4.1; Peters et al. 2010) for computing a stochastic policy based on the policy's return, estimated with *least-squares temporal difference learning* (LSTD) (cf. Section II.4.2; Boyan 1999), in Section VI.3 (Wirth et al. 2016). Kupcsik et al. (2015) employ *contextual REPS* (Kupcsik et al. 2014), but directly in the parameter space instead of the 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.

In contrast to undirected techniques, Wilson et al. (2012) propose two directed exploration criteria that are derived from the preference learning method explained in Section III.3.4.c. 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) and Akrour et al. (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* (LSPI) (Lagoudakis and Parr 2003a), that does not require additional samples for updating the policy in an off-policy manner. Yet, this technique requires a sufficient number 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. Jain et al. (2013) and Jain et al. (2015) extend 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.

III.3.4.b 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 (Chapter VI; Akrour et al. 2011, 2012; Akrour et al. 2014; Wirth et al. 2016). 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 (Chapter IV; Wilson et al. 2012; Busa-Fekete et al. 2013; Jain et al. 2013; Wirth and Fürnkranz 2013a, 2013b; Busa-Fekete et al. 2014; Jain et al. 2015; Kupcsik et al. 2015), or the user selects the samples for a preference query (Zucker et al. 2010).

Exhaustive Preference Query Generation

Some methods require that each trajectory is evaluated by the expert by defining preferences to each other trajectory, either for comparing and selecting policies (Busa-Fekete et al. 2013, 2014) or for evaluating and updating a single policy (Chapter IV; Jain et al. 2013; Wirth and Fürnkranz 2013a, 2013b; Kupcsik et al. 2015). These methods ignore that generating preferences can be more costly than generating trajectories and, hence, typically require a large number of expert queries.

Greedy Preference Query Generation

Greedy methods (Akrour et al. 2014) fully optimize a given objective, as we describe in Section III.3.4.c, which can be either an undirected or directed exploration objective. After the optimization is finished, a trajectory generated from the optimized policy is used to query the expert's evaluation function. While this method 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 Generation

The above-mentioned problem was recognized by several authors (Akrour et al. 2011, 2012; Wilson et al. 2012; Wirth et al. 2016; Christiano et al. 2017) and in Chapter VI. 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 Section III.3.4.c). This approach is followed by Akrour et al. (2011, 2012) where the search in parameter space is stopped early, after a given number of iterations. We request new preferences after a single iteration of the used policy optimization algorithm, as we explain in Chapter VI (Wirth et al. 2016). In each iteration, several trajectories are generated by the current policy but only a subset (typically one) is used for an expert query. We introduce different selectors in Section VI.4, that are based on principles encountered in the literature. Most commonly used are selection criteria based on expected utility improvement or expected utility change. Christiano et al. (2017) followed a similar approach, but 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 our approach, 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 preference queries. Our approach in Chapter VI (Wirth et al. 2016) and the algorithm by Akrour et al. (2011, 2012) 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.

III.3.4.c 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. We offer an additional comparison in Section VI.5.1.

Wilson et al. (2012) 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

$$\int_{\boldsymbol{\tau}_i,\boldsymbol{\tau}_j} \Pr(\boldsymbol{\tau}_i \mid \pi_i) \Pr(\boldsymbol{\tau}_j \mid \pi_j) d(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j), \tag{III.3.18}$$

where $d(\boldsymbol{\tau}_i, \boldsymbol{\tau}_j)$ is a trajectory difference function (III.3.4). 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 V over the policy space that results from adding a preference

$$V(\Pr(\pi \mid \zeta) \,||\, \Pr(\pi \mid \zeta \cup \{\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2}\}), \tag{III.3.19}$$

using potential queries by sampling policies from the current distribution. Different measures could be used as V. Wilson et al. (2012) use an approximation of the variational distance. In Section VI.4.4, we describe a variant that uses the *Kullback-Leibler divergence* of possible updates to the utility distribution $\Pr(U \mid \zeta)$. The query maximizing the criterion, based on a finite trajectory sample set, is then posed to the expert.

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

$$\begin{aligned}
\tilde{U}(\pi) &= U(\pi) + \alpha E(\pi), \\
U(\pi) &= \mathbb{E}_{\Pr^{\pi}(\boldsymbol{\tau})}[U(\boldsymbol{\tau})], \\
E(\pi) &= \min \Delta(\pi, \pi'), \pi' \in \Pi_t, \\
\Delta(\pi, \pi') &= \frac{|S_1| + |S_2| - |S_3|}{\sqrt{|S_1| + |S_3|}\sqrt{|S_2| + |S_3|}},
\end{aligned} \tag{III.3.20}$$

used by Akrour et al. (2011), adds a diversity term to the expected utility of a policy, comparing the number of states in S_1, only visited by π, the number of states in S_2, only visited by π' and the number of 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. III.3.3.b). Akrour et al. (2012) and Akrour et al. (2014) propose a different selection criterion that maximizes the expected utility of selection

$$\tilde{U}(\boldsymbol{\tau}) = \Pr(\boldsymbol{\tau} \succ \boldsymbol{\tau}^* \mid U)\mathbb{E}_{\zeta \cup \{\boldsymbol{\tau} \succ \boldsymbol{\tau}^*\}}\left[U(\boldsymbol{\tau})\right] + \Pr(\boldsymbol{\tau} \prec \boldsymbol{\tau}^* \mid U)\mathbb{E}_{\zeta \cup \{\boldsymbol{\tau} \prec \boldsymbol{\tau}^*\}}\left[U(\boldsymbol{\tau}^*)\right], \tag{III.3.21}$$

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; Akrour et al. 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

$$\boldsymbol{\tau} = \arg\max_{\boldsymbol{\tau} \in \boldsymbol{\tau}^{[i]}} \mathbb{E}\left[U_{\zeta}(\boldsymbol{\tau})\right], \tag{III.3.22}$$

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.

III.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.

III.3.5.a Direct Policy Search

Direct policy search methods 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 techniques. They maximize the objective defined by Equation III.2.2 and Equation III.2.3, however, this is difficult to compute directly due to the dependence on the transition dynamics. Hence, methods like policy likelihood or ranking (cf. Section III.3.2.a) are used. Alternatively, common return-based optimization can be employed by defining an utility-based return, as introduced in Section III.3.3.b. 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 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) and Akrour et al. (2014) also use an evolutionary strategy ($1 + \lambda$ ES; Auger 2005) for optimization in a parametric policy space, which requires a high number 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).

III.3.5.b Value-based Reinforcement Learning

Value-based methods 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 (Chapter VI; Akrour et al. 2014; Wirth et al. 2016; Christiano et al. 2017) is used to compute a value function for the policy (cf. Section III.3.2.c). This value function can, for example, be estimated by temporal difference methods such as LSTD (Boyan 1999), as explained in Section II.4.2. Given the value function, a new policy can be obtained in the policy improvement step. For example, we use a variant of the REPS algorithm (cf. Section II.4.1; Peters et al. 2010) to update the policy in Chapter VI (Wirth et al. 2016). 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 number of transition samples to be obtained beforehand which again stabilizes the policy iteration process.

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

III.3.5.c Planning

Many other approaches (Zucker et al. 2010; Jain et al. 2013; Gritsenko and Berenson 2014; Jain et al. 2015) 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 by Jain et al. (2013), Gritsenko and Berenson (2014), and Jain et al. (2015). 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.

III.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 methods are applicable. *Model-based approaches* assume that the system dynamics are known in advance, which allows the use of planning algorithms (Zucker et al. 2010; Jain et al. 2013; Gritsenko and Berenson 2014; Jain et al. 2015) or to directly derive an optimal policy, as in Chapter V (Wirth and Fürnkranz 2012, 2015) or by Runarsson and Lucas (2012), Sugiyama et al. (2012), and Runarsson and Lucas (2014) . In addition, some techniques 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 number 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 (Chapter IV, VI; Fürnkranz et al. 2012; Busa-Fekete et al. 2013; Wirth and Fürnkranz 2013a, 2013b; Akrour et al. 2014; Busa-Fekete et al. 2014; Wirth et al. 2016). 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 are not able to reuse samples, with the exception of the algorithm we introduce in Chapter VI (Wirth et al. 2016). Without sample reuse, a large number of samples is required 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.

III.3.7 Summary of PBRL Algorithms

Table III.2 shows a tabular summary of the discussed algorithms, grouped by the different learning problems, introduced in Section III.3.2. We list the introduced key concepts and the different *requirements*, stated throughout this chapter. The *preferences* column is related to the different types of preferences mentioned in Section III.3.1. The *trajectory generation* and *preference (query) generation* columns are based on the subsections of Section III.3.4, where we mention how trajectories and preference queries are generated to solve 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 Section III.3.2 and Section III.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
Chapter IV, Wirth et al. (2013a, 2013b)	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 + λ-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) and Jain et al. (2015)	Trajectory	Homogen+Guided	Exhaustive	Gradient	RRT	M
Kupcsik et al. (2015)	Trajectory	Homogen	Exhaustive	convex solver	C-REPS	PP
Runarsson et al. (2012, 2014)	State	-	-	SVM	Direct Max.	M
Sugiyama et al. (2012)	Trajectory	-	-	convex solver	Direct Max.	M, DS, DA
Chapter V, Wirth et al. (2012, 2015)	State	-	-	SVM, LP	Direct Max.	M
Chapter VI, Wirth et al. (2016)	Trajectory	Homogen	Interleaved	LP, ELS	LSTD, REPS	
Zucker et al. (2010)	State	Homogen	Guided	SVM	ARA*	M

Table III.2.: Overview of PbRL approaches by design principle

Algorithms: 1 + λ-*ES*: 1 + λ 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 (Herbrich et al. 1999; Joachims 2002), *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

Chapter IV

Reinforcement Learning with Preference Surrogates

This chapter focuses on the effects of the assumption of trajectory-based preference feedback (cf. Section III.3.1.c). In contrast to numeric state-action rewards, we have to deal with three sources of information loss:

- Aggregation: Preferences are only valid for complete trajectories and information about the quality of state-actions is not accessible.
- Binarization: Preferences do not contain information about the value of utility difference of two trajectories. We can only determine if the expert's, internal utility is higher or lower.
- Relativization: Numeric rewards determine a quality on an absolute scale, whereas preferences are only relative. The quality of a trajectory can only be determined by requesting preference feedback in relation to another trajectory, whereas numeric rewards do not require explicit comparisons.

We focus on solving the aggregation problem, by computing an approximation for an explicit solution for the temporal credit assignment problem, as explained in Section III.3.3. The ability to attribute trajectory preferences to specific time steps allows us to generalize the obtained preference feedback to states and actions. Therefore, we can derive additional preferences from a single trajectory preference, increasing the amount of training data without requiring additional expert queries. We also analyze the effects of the binarization by considering an alternative approach to the *temporal credit assignment problem* that enables us to use graded preferences. Graded preferences are still aggregated and relative feedback, but contain a notion of difference.

To be able to focus on the stated problems, we decided to base our analysis on the *preference-based reinforcement learning* (PBRL) algorithm by Fürnkranz et al. (2012). It does not apply any generalization methods besides using a neural-network function approximator (cf. Section II.3.4) for modeling the policy. The structure of the algorithm enabled us to easily replace the policy with a tabular version. Hence, we removed all generalization methods that could influence the results and only observe the effects of the temporal credit assignment method. Besides considering the assignment problem, we also aim at improving the sample efficiency in terms of required preferences.

Therefore, we analyze the following research questions in this chapter:

1.A Can we improve efficiency by solving the temporal credit assignment problem explicitly?

1.B Is the temporal credit assignment problem a substantial limitation for PBRL in terms of efficiency that should be investigated further?

1.C Is binary feedback a substantial limitation for efficiently learning in a *Markov decision process with preferences* (MDPP)?

1.D How can we improve the preference sample efficiency to reduce the cognitive load for the expert?

IV.1 Preference-based Approximate Policy Iteration

Fürnkranz et al. (2012) introduced an algorithm for PBRL that is based on the multi-class variant of *approximate policy iteration with rollouts* (API; Lagoudakis and Parr 2003b). Actions are considered to be classes in a multi-class problem which enables the modeling of a policy as a classifier that predicts the most likely action, given a state. The training information $\mathscr{E}$ for the classifier is generated by performing rollouts, given the current policy, for computing the expected return of an action $Q^{\pi}(s,a)$ via Monte Carlo sampling, as shown in Equation II.2.20. The current best action $\arg\max_{a\in A(s)} Q^{\pi}(s,a)$ is then used as the target class for defining a new training instance

$$\mathscr{E} = \mathscr{E} \leftarrow \{s, \arg\max_{a\in A(s)} Q^{\pi}(s,a)\}. \tag{IV.1.1}$$

Hence, a classifier trained on $\mathscr{E}$ can be used as an approximation for the optimal policy as it predicts the action maximizing the Q-function. To be able to apply this idea to preference-based feedback, the Q-function is modeled as a label ranker (cf. Section II.5.2) with the actions as labels. The ranker learns a preference relation surrogate, as discussed in Section II.5.3.b, that predicts the preference relation between two actions, given a state. Therefore, it defines a *relative Q-function* $\tilde{Q}^{\pi}_{\text{rel}}(s,a,a') = \mathbb{1}(\Pr(a \succ a' \mid s) > 0.5)$ that reconstructs the binary preferences. Training information $\mathscr{E}$ is obtained by comparing trajectory pairs $\boldsymbol{\tau}_i$, $\boldsymbol{\tau}_j$ with the same, initial state s but two different initial actions $\boldsymbol{\tau}_i(0) = (s,a_i)$, $\boldsymbol{\tau}_j(0) = (s,a_j)$, $a_i \neq a_j$. A training instance is then derived by using the encountered preference as a sample for the expected relation of the initial actions

$$\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j \Rightarrow \mathscr{E} \leftarrow \mathscr{E} \cup \{s, a_i, a_j\}. \tag{IV.1.2}$$

In contrast to API, the training information $\mathscr{E}$ is used to compute $\tilde{Q}^{\pi}_{\text{rel}}(s,a,a')$, and not $\pi(s)$. The resulting policy is defined by deterministically determining the highest ranked action

$$\begin{aligned} \pi_{i+1}(s) &= \arg\max_a \tilde{Q}^{\pi_i}(s,a), \\ \tilde{Q}^{\pi_i}(s,a) &= \sum_{a'\in A(s), a'\neq a} \tilde{Q}^{\pi_i}_{\text{rel}}(s,a,a'). \end{aligned} \tag{IV.1.3}$$

and selecting it greedily. As shown by Algorithm 6, this approximation is computed for k state samples each iteration, based on n rollouts for all actions $a \in |A|$. The function ROLLOUT samples new trajectories using the current policy π_i. The trajectory preferences obtained by OBTAIN-

Algorithm 6 Preference-based Approximate Policy Iteration

```
Require: initial policy $\pi_0$, iteration limit $m$, state sample limit $k$, rollout limit $n$
 1: for $i = 0$ to $m$ do
 2:     $\mathscr{E} = \emptyset$                                          ▷ Set of training data
 3:     for 0 to $k$ do
 4:         $s \sim \mu(s)$                                   ▷ Sample $k$ states per iteration
 5:         $\Upsilon = \emptyset, \zeta = \emptyset$
 6:         for $\forall a \in A(s)$ do
 7:             for 0 to $n$ do
 8:                 $\Upsilon \leftarrow \Upsilon \cup$ ROLLOUT$(s, a, \pi_i)$   ▷ Create trajectory, starting with $(s, a)$
 9:             end for
10:         end for
11:         $\zeta \leftarrow$ OBTAINTRAJECTORYPREFERENCES$(\Upsilon)$                ▷ Query expert
12:         $\mathscr{E} \leftarrow \mathscr{E} \cup$ DETERMINEACTIONPREFERENCE$(\zeta)$   ▷ Preferences for initial actions
13:     end for
14:     $\tilde{Q}^{\pi_i}_{\text{rel}}$ = COMPUTEQFUNCTION$(\mathscr{E})$           ▷ Approximate relative Q Function
15:     $\pi_{i+1}$ = COMPUTEPOLICY$(\tilde{Q}^{\pi_i}_{\text{rel}})$                ▷ Compute greedy policy
16: end for
17: return improved policy $\pi_m$
```

TrajectoryPreferences are converted to action preferences (determineActionPreference), as explained. The method computeQFunction updates the label ranker and computePolicy applies Equation IV.1.3 to derive a new policy. The *preference-based approximate policy iteration* (PBPI) algorithm of Fürnkranz et al. (2012) is fairly inefficient because trajectory feedback is reduced to preferences over initial state-action pairs, disregarding all intermediate states observed along the trajectory. To be able to approximate the Q-function for the entire state space, it is therefore also required to be able to initiate the rollouts from any state S, equivalent to $\forall (s \in S)\, \mu(s) > 0$. The sampling is also performed for all actions, without considering the expected outcome. An exploration/exploitation tradeoff, as described in Section II.2.2.b, is not used. Additionally, old training information is discarded after each policy update and not reused.

IV.2 Every-Visit Preference Monte Carlo

The PBPI algorithm can be seen as a bandit algorithm that approximates the expected outcome of each arm with a *first-visit* Monte Carlo strategy. Hence, we can use the ideas of *every-visit* Monte Carlo strategies for using the rollout information more efficiently, as explained in Section II.2.3.a. Rollouts are also used to obtain feedback for intermediate states in the trajectory, not only the initial one. The bandit view also enables us to consider arm selection strategies (Kuleshov and

Algorithm 7 Every-Visit Preference Monte Carlo

```
Require: initial policy $\pi_0$, iteration limit $m$, sample limit $n$
 1: for $i = 0$ to $m$ do
 2:     $s \sim \mu(s)$
 3:     $\Upsilon = \emptyset, \zeta = \emptyset$
 4:     for $j = 0$ to $n$ do
 5:         $a$ = BANDITSTRATEGY($s$)                    ▷ Apply bandit strategy for selection $a$
 6:         $\Upsilon \leftarrow \Upsilon \cup$ PREFERENCEROLLOUT($s, a, \pi_i$)   ▷ Create trajectory, starting with $(s, a)$
 7:     end for
 8:     $\zeta \leftarrow$ OBTAINTRAJECTORYPREFERENCES($\Upsilon$)          ▷ Query expert
 9:     $\mathscr{E} \leftarrow$ DETERMINEACTIONPREFERENCES($\zeta$)    ▷ Preferences for multiple state-actions
10:     $\tilde{Q}^{\pi_i}_{\text{rel}}$ = COMPUTEQFUNCTION($\mathscr{E}$)          ▷ Approximate relative Q Function
11:     $\pi_{i+1}$ = COMPUTEPOLICY($\pi_i, \tilde{Q}^{\pi_i}_{\text{rel}}$)              ▷ Compute greedy policy
12: end for
13: return improved policy $\pi_m$
```

Precup 2014) for introducing an exploration/exploitation strategy that eliminates the requirement of sampling every action in every iteration.

The basic idea of every-visit Monte Carlo is to also compute $Q^{\pi}(s, a)$ for states within the trajectory, not only the initial one. When obtaining preference feedback for learning $\tilde{Q}_{\text{rel}}(s, a, a')$, we need a way to approximate a solution to the temporal credit assignment problem (cf. Section III.3.3), for enabling every-visit Monte Carlo procedures. In the following, we present two approximation strategies for solving this problem. Both algorithms work in the framework of *every-visit preference Monte Carlo* (EPMC; Algorithm 7), which adapts the every-visit Monte Carlo strategy to preference-based feedback. In contrast to PBPI, we select only a single start state in each iteration because we use a gradient-based update for the policy and can obtain training information for other parts of the state space by solving the temporal credit assignment problem. We explain the gradient-based update in Section IV.5.3, that is implemented in the method COMPUTEPOLICY. Solving the temporal credit assignment problem requires us to change the rollout strategy, implemented as PREFERENCEROLLOUT, as we explain in Section IV.5.2. We introduce a mechanism that allows us to derive additional preferences for states that occur in multiple trajectories. The techniques for solving the temporal credit assignment are implemented in the DETERMINEACTIONPREFERENCE function, explained in Section IV.3 and Section IV.4. In contrast to PBPI, we may obtain multiple action preferences from a single trajectory preference and also approximate degrees of preference. Furthermore, we replace the exhaustive action sampling (Algorithm 6, line 6) with a BANDITSTRATEGY action selection method that allows us to reduce the amount of required rollouts.

IV.3 Probabilistic Temporal Credit Assignment[29]

The basic idea is to estimate the probability that an action choice (s, a) belongs to a set of preferred actions. As we are restricted to feedback in form of pairwise preferences, we only estimate this probability for a pairwise comparison to another action. It is not easily possible to determine by "how much" an action should be preferred over another one. Hence, we assume that an action is maximally preferred if it is part of the set of preferred actions. The probability of membership to this set is then used to update $\tilde{Q}_{\text{rel}}(s, a, a')$. Hence, the training information is not binary anymore, but a probability sample $\Pr(a \succ a' \mid s)$. This converts the classifier $\tilde{Q}_{\text{rel}}(s, a, a')$ into a regressor that predicts the probability $\Pr(a \succ a' \mid s)$.

IV.3.1 Computing intermediate Action Preferences

For estimating $\Pr(a \succ a' \mid s)$, we obtain preferences for all pairings of n trajectories (see Section IV.2). In the simplest case $n = 2$, we only compare a pair of trajectories. In general, we will obtain $n(n-1)/2$ pairwise preferences for these n trajectories.

Given $\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2$, we want to estimate the probability $\Pr(a \succ a' \mid s, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2)$ that an action a, $(s, a) \in \boldsymbol{\tau}_1$ is preferred over an action a', $(s, a') \in \boldsymbol{\tau}_2, a \neq a'$ in a particular state $s \in \boldsymbol{\tau}_1 \cap \boldsymbol{\tau}_2$. For doing so, we make the following assumptions:

1. The reason for $\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2$ can be found in the overlap of states $\mathscr{K} = \boldsymbol{\tau}_1 \cap \boldsymbol{\tau}_2$ if we are following an optimal policy in all other states.

2. Let $\mathscr{M}$ be the set of states for which the action a taken in $\boldsymbol{\tau}_1$ is preferable to the action a' taken in $\boldsymbol{\tau}_2$, i.e.,
$$\Pr(a \succ a' \mid s, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2, s \in \mathscr{M}) = 1.$$
We call $\mathscr{M}$ the set of *decisive states*.

3. The probability $\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2$ is proportional to the ratio of *decisive states* $\mathscr{M}$ to all overlapping states $\mathscr{K}$. Therefore, the more often we pick a preferred action, the more likely is the resulting trajectory preferred:
$$\Pr(\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2 \mid p = |\mathscr{M}|) = \frac{p}{|\mathscr{K}|}. \tag{IV.3.4}$$

[29] This section is based on Wirth and Fürnkranz (2013a).

$|\mathscr{M}|$ denotes the number of *decisive states* and $|\mathscr{H}|$ is the number of overlapping states. p is the number of decisive states assumed to exist. Note that a state may occur multiple times in each trajectory and therefore also in $\mathscr{M}$ and $\mathscr{H}$.

For estimating $\Pr(a \succ a' \mid s, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2)$, we require $\Pr(s \in \mathscr{M} \mid \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2)$ to be able to solve

$$\Pr(a \succ a' \mid s, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) = \Pr(a \succ a' \mid s, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2, s \in \mathscr{M}) \Pr(s \in \mathscr{M} \mid \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2). \quad \text{(IV.3.5)}$$

This is reduced to $\Pr(a \succ a' \mid s, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) = \Pr(s \in \mathscr{M} \mid \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2)$, because of Assumption 2, where we assume that it is not possible to determine the exact value of $\Pr(a \succ a' \mid s, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2, s \in \mathscr{M})$ easily. $\Pr(s \in \mathscr{M} \mid \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2)$ can be estimated directly for specific p by applying

$$\Pr(s \in \mathscr{M} \mid p = |\mathscr{M}|, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) = \frac{p}{|\mathscr{H}|}, \quad \text{(IV.3.6)}$$

using the priori probability[30] $\Pr(s \in \mathscr{M}) = \frac{1}{2}$. Assuming a binomial distribution induced by $\Pr(s \in \mathscr{M}) = \frac{1}{2}$, we obtain $\Pr(p = |\mathscr{M}|) = \binom{|\mathscr{H}|}{p} \cdot \frac{1}{2}^{|\mathscr{H}|}$. Applying Bayes theorem to Equation IV.3.4 and the prior probability $\Pr(\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) = \frac{1}{2}$, yields

$$\Pr(|\mathscr{M}| = p \mid \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) = \frac{\Pr(\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2 \mid p = |\mathscr{M}|) \Pr(p = |\mathscr{M}|)}{\Pr(\boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2)} = \frac{p}{|\mathscr{H}|} \binom{|\mathscr{H}|}{p} \frac{1}{2}^{|\mathscr{H}|-1}.$$

We can now estimate $\Pr(s \in \mathscr{M} \mid \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2)$ by summing over all possible cases for p:

$$\Pr(s \in \mathscr{M} \wedge |\mathscr{M}| = p \mid \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) = \Pr(s \in \mathscr{M} \mid p = |\mathscr{M}|, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) \Pr(|\mathscr{M}| = p \mid \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2),$$

$$\Pr(s \in \mathscr{M} \mid \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) = \sum_{p=0}^{|\mathscr{H}|} \Pr(s \in \mathscr{M} \wedge |\mathscr{M}| = p \mid \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) = \frac{|\mathscr{H}| + 1}{2|\mathscr{H}|}. \quad \text{(IV.3.7)}$$

Equation IV.3.7 is now used to determine the value of Equation IV.3.5, resulting in an additional training instances

$$\mathscr{E} \leftarrow \mathscr{E} \cup \Pr(a \succ a' \mid s, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2). \quad \text{(IV.3.8)}$$

30 The observed prior probabilities for $\Pr(s \in \mathscr{M})$ can not be used because of computational reasons, namely time and numerical stability problems.

The case $\boldsymbol{\tau}_2 \succ \boldsymbol{\tau}_1$ can be mapped to $\Pr(a \succ a' \,|\, s, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2)$ by using $\Pr(a \succ a' \,|\, s, \boldsymbol{\tau}_1 \succ \boldsymbol{\tau}_2) = 1 - \Pr(a' \succ a \,|\, s, \boldsymbol{\tau}_2 \succ \boldsymbol{\tau}_1)$ due to the preference symmetries mentioned in Section II.5.1.

We have to be able to assume that the decisive action(s) did really occur in an overlapping state $s \in \boldsymbol{\tau}_1 \cap \boldsymbol{\tau}_2$. This can be achieved by always applying a deterministic, optimal policy π^* in all states $s \in \boldsymbol{\tau}_1, s \notin \boldsymbol{\tau}_2$ and vice versa. This will yield the same, optimal outcome for all non-overlapping states. Of course, π^* is unknown, but we can use the current best approximation (IV.1.3). This is realized by the EPMC rollout procedure, Section IV.5.2.

IV.3.2 Reconstructing the Q-Function

For obtaining a policy using the principle from Equation IV.1.3, we require $\tilde{Q}(s, a)$. Reconstructing $\tilde{Q}(s, a)$ is comparable to reconstructing $\Pr(a \,|\, s)$ from $\Pr(a \succ a' \,|\, s)$ in the probabilistic setting (cf. Section IV.3). Computing element-wise probabilities from pairwise samples is known as *pairwise coupling* (Hastie and Tibshirani 1997). Several methods have been proposed in the literature (Wu et al. 2004); we rely on the suggestion of Price et al. (1994), who estimate $\Pr(a \,|\, s)$ as

$$\Pr(a \,|\, s) = \frac{1}{\sum\limits_{a' \in A(s), a' \neq a} \frac{1}{\Pr(a \succ a' \,|\, s)} - (|A(s)| - 2)}. \tag{IV.3.9}$$

As a result, we can use $\tilde{Q}(s, a) \sim \Pr(a \,|\, s)$ and obtain probabilities for suboptimal actions.

IV.4 Numeric Temporal Credit Assignment[32]

As an alternative to the probabilistic approximation, we can use a numeric approach to the temporal credit assignment problem that allows us to incorporate prior knowledge. We relate a preference to a factor using the unknown return $R(\boldsymbol{\tau})$ (II.2.2) of a trajectory. Hence,

$$\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j \Leftrightarrow R(\boldsymbol{\tau}_i) > R(\boldsymbol{\tau}_j) \Leftrightarrow R(\boldsymbol{\tau}_i) = R(\boldsymbol{\tau}_j) \cdot x_{i,j}, \tag{IV.4.10}$$

where $x_{i,j}$ is the relative improvement of $\boldsymbol{\tau}_i$ over $\boldsymbol{\tau}_j$. $x_{i,j}$ is symmetric, e.g., $x_{i,j} = \frac{1}{x_{j,i}}$. Assuming the (unknown) rewards to be all positive ($r \in \mathbb{R}^+$) results in $x_{i,j} \in \mathbb{R}^+$. This way, it is also possible to assume that a higher $x_{i,j}$ value is always preferable, because it results in a higher return.

The predicate $\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j$ is equivalent to $x_{i,j} > 1$ for $r \in \mathbb{R}^+$, hence, a preference defines the relations where $x > 1$ is valid. When required, a negative reward space $r \in\rightarrow \mathbb{R}^-$ can also be

[32] This section is based on Wirth and Fürnkranz (2013b).

used and according changes are described within the relevant sections. Here, the difference is $\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j \Rightarrow x_{i,j} < 1$.

Furthermore, we assume some prior knowledge over the distribution of rewards. In our case, we use a low-variance assumption for the reward distribution. This enables the approximation of a trajectories return (II.2.2) by

$$R(\boldsymbol{\tau}) \approx \sum_{t=0}^{|\boldsymbol{\tau}|} \gamma^t \cdot \bar{r}_{\boldsymbol{\tau}}, \tag{IV.4.11}$$

where $\bar{r}_{\boldsymbol{\tau}} = \frac{1}{|\boldsymbol{\tau}|} R(\boldsymbol{\tau})$ is the (unknown) mean of all values. All tasks that can be reduced to finding the shortest or longest trajectory through the state space satisfy this assumption because they can be described with a fixed step penalty or reward. Additionally, terminal-only reward problems are also subject to low variance. In fact, a large number of real world tasks do fit into this framework:

- Game Playing - Each move has a reward of 0 with a terminal reward depending on the outcome of the game.
- Planning - Every step of the plan is penalized with all states that achieve the goal as terminal states. This results in the shortest, successful plan having the highest reward.
- Navigation - A uniform time discretization for the state space enables the use of a fixed step penalty.

The assumed feedback in the form of $x_{i,j} > 1$ s.t. $R(\boldsymbol{\tau}_i) = R(\boldsymbol{\tau}_j) \cdot x_{i,j}$ results in several problems. At first, our feedback is limited to information about $x_{i,j}$, which can not be used to calculate $R(\boldsymbol{\tau}_i)$ or $R(\boldsymbol{\tau}_j)$ without knowledge about the value of either of these. Secondly, the exact value of $x_{i,j}$ is also unknown. Additionally, our (relative) return sample is only valid for estimating the outcome of the initial state-action pair (s_0, a_0) and we can not derive any information about other time-steps. However, we also need information about $(s_t, a_t), t \neq 0$ for applying every-visit methods. Hence, we need to be able to approximate feedback for partial trajectories $\boldsymbol{\tau} = \boldsymbol{\tau}^{\vdash}.\boldsymbol{\tau}^{\dashv}$ with $\boldsymbol{\tau}^{\vdash}$ as the first part of the trajectory and $\boldsymbol{\tau}^{\dashv}$ as the second part. More exactly, an approximation for $R(\boldsymbol{\tau}_i^{\dashv}) = R(\boldsymbol{\tau}_j^{\dashv}) \cdot y_{i,j}$ s.t. $\boldsymbol{\tau}_i = \boldsymbol{\tau}_i^{\vdash}.\boldsymbol{\tau}_i^{\dashv}, |\boldsymbol{\tau}_i^{\vdash}| \neq 0, \boldsymbol{\tau}_j = \boldsymbol{\tau}_j^{\vdash}.\boldsymbol{\tau}_j^{\dashv}, |\boldsymbol{\tau}_j^{\vdash}| \neq 0$ is required. $x_{i,j}$ and $y_{i,j}$ are comparable, but $x_{i,j}$ always concerns complete trajectories $\boldsymbol{\tau}_i$ and $\boldsymbol{\tau}_j$ (IV.4.10). $y_{i,j}$ denotes the same factor, but for partial trajectories, e.g., $R(\boldsymbol{\tau}_i^{\dashv}) = R(\boldsymbol{\tau}_j^{\dashv}) \cdot y_{i,j}$.

In short, our problems are as described in the beginning of this chapter:

1. Relativization: $R(\boldsymbol{\tau}_i)$ and $R(\boldsymbol{\tau}_j)$ are unknown, because we only obtain relative feedback.
2. Binarization: The exact value of $x_{i,j}$ is unknown, because we only obtain binary feedback.

3. Aggregation: The value of $y_{i,j}$ is unknown, because we only obtain feedback for complete trajectories.

This section explains the problems in detail as well as the applied solutions.

IV.4.1 The Relative Q-Function for Numeric Approximations

Without a specific reward sum $R(\boldsymbol{\tau})$ as feedback, it is not possible to estimate $Q(s, a)$ directly. According to the Monte Carlo framework, we can use $R(\boldsymbol{\tau})$ as sample for the expected return $Q(s_0, a)$ of the initial state of the trajectory $\boldsymbol{\tau}(0) = (s_0, a)$. Hence, we can also consider Q-values relative to each other as $Q(s_0, a) = Q(s_0, a') \cdot x_{i,j}$, if we can obtain two trajectories that start in the same state s_0 but have different, initial actions, e.g., $\boldsymbol{\tau}_i(0) = (s_0, a)$ and $\boldsymbol{\tau}_j(0) = (s_0, a')$. As a result, we can define a Q-function in terms of relative feedback to solve Problem 1.

We only store a value $Q_{\text{rel}}(s_0, a, a')$ as relative Q value for a comparison of $Q(s_0, a)$ and $Q(s_0, a')$, as defined by

$$\begin{aligned}
Q(s_0, a) &= Q(s_0, a') \cdot x_{i,j}, \\
\Leftrightarrow\ Q(s_0, a) &= Q(s_0, a') + Q(s_0, a') \cdot (x_{i,j} - 1), \\
\Leftrightarrow\ Q(s_0, a') + Q_{\text{rel}}(s, a, a') \cdot Q(s_0, a') &= Q(s_0, a') + Q(s_0, a') \cdot (x_{i,j} - 1), \\
\Leftrightarrow\ Q_{\text{rel}}(s_0, a, a') &= x_{i,j} - 1.
\end{aligned} \tag{IV.4.12}$$

$Q_{\text{rel}}(s_0, a, a')$ and its symmetric counterpart

$$\begin{aligned}
Q(s_0, a') &= Q(s_0, a) + Q_{\text{rel}}(s_0, a', a) \cdot Q(s_0, a), \\
\Leftrightarrow\ -Q_{\text{rel}}(s_0, a', a) \cdot Q(s_0, a) + Q(s_0, a') &= Q(s_0, a), \\
\Leftrightarrow\ -Q_{\text{rel}}(s_0, a', a) \cdot Q(s_0, a) + Q(s_0, a') &= Q(s_0, a') + Q_{\text{rel}}(s_0, a, a') \cdot Q(s_0, a'), \\
\Leftrightarrow\ Q_{\text{rel}}(s_0, a', a) &= Q_{\text{rel}}(s_0, a, a') \cdot \left(-\frac{Q(s_0, a')}{Q(s_0, a)}\right), \\
\Leftrightarrow\ Q_{\text{rel}}(s_0, a', a) &= Q_{\text{rel}}(s_0, a, a') \cdot \left(-\frac{1}{x_{i,j}}\right),
\end{aligned} \tag{IV.4.13}$$

are stored separately, because the exact value of $x_{i,j}$ is not known. We can only compute estimates that may differ depending on the ordering of the trajectories, as we explain in the next section.

IV.4.2 Handling Binary Preferences

According to Equation IV.4.12, we require the value $x_{i,j}$ as learning information for our relative Q-function, but we only obtain feedback in the form of $x_{i,j} > 1$. We purposely underestimate $x_{i,j}$ (as $\tilde{x}_{i,j}$), such that $x_{i,j} = \tilde{x}_{i,j} + k_{i,j}$ for $r \in \mathbb{R}^+$. We can now set $\tilde{x}_{i,j} = 1$ for describing "at least slightly better", resulting in

$$\tilde{Q}_{\text{rel}}(s_0, a, a') = Q_{\text{rel}}(s_0, a, a') - k_{i,j}, \tag{IV.4.14}$$

to solve Problem 2. The symmetric counterpart (IV.4.13) would be

$$\begin{aligned} Q_{\text{rel}}(s_0, a', a) &= Q_{\text{rel}}(s_0, a, a') \cdot \left(-\frac{1}{x_{i,j}}\right), \\ \Leftrightarrow Q_{\text{rel}}(s_0, a', a) &= (\tilde{Q}_{\text{rel}}(s_0, a, a') + k_{i,j}) \cdot \left(-\frac{1}{\tilde{x}_{i,j} + k_{i,j}}\right), \\ \Leftrightarrow Q_{\text{rel}}(s_0, a', a) &= (\tilde{x}_{i,j} - 1 + k_{i,j}) \cdot \left(-\frac{1}{\tilde{x}_{i,j} + k_{i,j}}\right), \end{aligned} \tag{IV.4.15}$$

but we can only use $\tilde{Q}_{\text{rel}}(s_0, a', a) = \tilde{Q}_{\text{rel}}(s_0, a, a') \cdot -\frac{1}{\tilde{x}_{i,j}} = -\frac{\tilde{x}_{i,j}-1}{\tilde{x}_{i,j}}$ because $k_{i,j}$ is unknown. We can see that this estimate is an overestimate by proving the relation to the correct estimate (IV.4.15). The equation

$$\begin{aligned} -\frac{\tilde{x}_{i,j} - 1}{\tilde{x}_{i,j}} &\geq -\frac{\tilde{x}_{i,j} - 1 + k_{i,j}}{\tilde{x}_{i,j} + k_{i,j}}, \\ \Leftrightarrow -(\tilde{x}_{i,j} - 1)(\tilde{x}_{i,j} + k_{i,j}) &\geq -(\tilde{x}_{i,j} - 1 + k_{i,j}) \cdot \tilde{x}_{i,j}, \\ \Leftrightarrow -\tilde{x}_{i,j}^2 - \tilde{x}_{i,j}k_{i,j} + \tilde{x}_{i,j} + k_{i,j} &\geq -\tilde{x}_{i,j}^2 + \tilde{x}_{i,j} - \tilde{x}_{i,j}k_{i,j}, \\ \Leftrightarrow k_{i,j} &\geq 0, \end{aligned} \tag{IV.4.16}$$

is always true because of $k_{i,j} \in \mathbb{R}^+$. For determining the best approximation $\tilde{Q}_{\text{rel}}(s, a, a')$, we can now average the directly determined underestimates and the overestimates derived from $\tilde{Q}_{\text{rel}}(s, a', a)$.

Using $r \in \mathbb{R}^-$, this becomes an average over directly determined overestimates and derived underestimates, because of $x_{i,j} = \tilde{x}_{i,j} - k_{i,j}$.

IV.4.3 Estimating Preferences for Partial Trajectories

In an every-visit Monte Carlo method, we do not only want to update $Q(s_0, a)$, but also intermediate state-action pairs. We want to estimate $y_{i,j}$ s.t. $R(\boldsymbol{\tau}_i^{\dashv}) = R(\boldsymbol{\tau}_j^{\dashv}) \cdot y_{i,j}$ with $y_{i,j}$ as the relative value of the latter part of each trajectory from a predefined split point. An approximation for $y_{i,j}$ would allow us to overcome Problem 3 because this value defines the relative difference for partial trajectories and therefore the relative Q-value difference for intermediate state-action pairs.

In the following, $\bar{r}_{\boldsymbol{\tau}}$ denotes the mean of rewards in the trajectory with r_t^{Δ} as the deviation of the reward $r(s_t, a_t) = \bar{r}_{\boldsymbol{\tau}} + r_t^{\Delta}$. Considering the low-variance assumption for the trajectory return (IV.4.11) yields

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{|\boldsymbol{\tau}|-1} \gamma^t r(s_t, a_t) \;\; = \sum_{t=0}^{|\boldsymbol{\tau}|-1} \gamma^t (\bar{r}_{\boldsymbol{\tau}} + r_t^{\Delta}) \approx \sum_{t=0}^{|\boldsymbol{\tau}|} \gamma^t \cdot \bar{r}_{\boldsymbol{\tau}}. \tag{IV.4.17}$$

This is a geometric series and therefore equivalent to $R(\boldsymbol{\tau}) \approx \bar{r}_{\boldsymbol{\tau}} \frac{1-\gamma^{|\boldsymbol{\tau}|+1}}{1-\gamma}$. Additionally, we define the relative trajectory length difference $d = \frac{|\boldsymbol{\tau}^{\dashv}|}{|\boldsymbol{\tau}^{\vdash}.\boldsymbol{\tau}^{\dashv}|}$ and obtain

$$\begin{aligned} R(\boldsymbol{\tau}_i) &= R(\boldsymbol{\tau}_j) \cdot x_{i,j}, \\ \Rightarrow \bar{r}_{\boldsymbol{\tau}_i} \frac{1-\gamma^{|\boldsymbol{\tau}_i|+1}}{1-\gamma} &\approx \bar{r}_{\boldsymbol{\tau}_j} \frac{1-\gamma^{|\boldsymbol{\tau}_j|+1}}{1-\gamma} \cdot x_{i,j}, \end{aligned} \tag{IV.4.18}$$

$$\begin{aligned} R(\boldsymbol{\tau}_i^{\dashv}) &= R(\boldsymbol{\tau}_j^{\dashv}) \cdot y_{i,j}, \\ \Rightarrow \bar{r}_{\boldsymbol{\tau}_i} \frac{1-\gamma^{d_i|\boldsymbol{\tau}_i|+1}}{1-\gamma} &\approx \bar{r}_{\boldsymbol{\tau}_j} \frac{1-\gamma^{d_j|\boldsymbol{\tau}_j|+1}}{1-\gamma} \cdot y_{i,j}. \end{aligned} \tag{IV.4.19}$$

By expanding Equation IV.4.19 and inserting Equation IV.4.18 we can determine the value of $y_{i,j}$ relative to the value $x_{i,j}$ as

$$\begin{aligned}
\bar{r}_{\boldsymbol{\tau}_i}\frac{1-\gamma^{d_i|\boldsymbol{\tau}_i|+1}}{1-\gamma} &\approx \bar{r}_{\boldsymbol{\tau}_j}\frac{1-\gamma^{d_j|\boldsymbol{\tau}_j|+1}}{1-\gamma}\cdot y_{i,j},\\
\Leftrightarrow \bar{r}_{\boldsymbol{\tau}_i}\frac{1-\gamma^{d_i|\boldsymbol{\tau}_i|+1}}{1-\gamma}\frac{1-\gamma^{|\boldsymbol{\tau}_i|+1}}{1-\gamma}&,\\
&\approx \bar{r}_{\boldsymbol{\tau}_j}\frac{1-\gamma^{d_j|\boldsymbol{\tau}_j|+1}}{1-\gamma}\frac{1-\gamma^{|\boldsymbol{\tau}_i|+1}}{1-\gamma}\cdot y_{i,j},\\
\Leftrightarrow \bar{r}_{\boldsymbol{\tau}_j}\frac{1-\gamma^{d_i|\boldsymbol{\tau}_i|+1}}{1-\gamma}\frac{1-\gamma^{|\boldsymbol{\tau}_j|+1}}{1-\gamma}\cdot x_{i,j}&,\\
&\approx \bar{r}_{\boldsymbol{\tau}_j}\frac{1-\gamma^{d_j|\boldsymbol{\tau}_j|+1}}{1-\gamma}\frac{1-\gamma^{|\boldsymbol{\tau}_i|+1}}{1-\gamma}\cdot y_{i,j},\\
\Leftrightarrow \frac{(1-\gamma^{d_i|\boldsymbol{\tau}_i|+1})(1-\gamma^{|\boldsymbol{\tau}_j|+1})}{(1-\gamma^{d_j|\boldsymbol{\tau}_j|+1})(1-\gamma^{|\boldsymbol{\tau}_i|+1})}\cdot x_{i,j} &\approx y_{i,j}.
\end{aligned} \tag{IV.4.20}$$

Equation IV.4.20 defines the relative difference for intermediate states. This estimate is only calculated for states occurring in both trajectories. Different states cannot be compared without some kind of distance or similarity function, introducing new requirements for the state space definition.

Our solution to Problem 2, as introduced in Section IV.4.2, requires us to assume $\tilde{x}_{i,j} + k_{i,j}$ to be the true reward factor. Hence, $y_{i,j} \approx z_{i,j} \cdot (\tilde{x}_{i,j} + k_{i,j})$ with $z_{i,j} = \frac{(1-\gamma^{d_i|\boldsymbol{\tau}_i|+1})(1-\gamma^{|\boldsymbol{\tau}_j|+1})}{(1-\gamma^{d_j|\boldsymbol{\tau}_j|+1})(1-\gamma^{|\boldsymbol{\tau}_i|+1})}$. But again, due to $k_{i,j} \in \mathbb{R}^+$, we know $y_{i,j} \approx z_{i,j} \cdot (\tilde{x}_{i,j} + k_{i,j}) \geq z_{i,j} \cdot \tilde{x}_{i,j}$ which means $z_{i,j} \cdot \tilde{x}_{i,j}$ is an underestimation of the true value. We omit $y_{i,j} < 1$ for $r \in \mathbb{R}^+$ ($y_{i,j} > 1$ for $r \in \mathbb{R}^-$), as this would be a training information contradicting the current preference, but it is probably an estimation error.

IV.4.4 Reconstructing the Q-Function

The estimates $y_{i,j}$ from Equation IV.4.20 are used as training data

$$\mathscr{E} \leftarrow \mathscr{E} \cup \{s, a_i, a_j, y_{i,j} - 1\}. \tag{IV.4.21}$$

The resulting function $\tilde{Q}_{\text{rel}}(s, a, a')$ is not defining a probability anymore. Hence, it is not reasonable to use the pairwise coupling scheme introduce in Section IV.3.2 to reconstruct $\tilde{Q}(s, a)$. Therefore, we use the average, relative difference

$$\tilde{Q}(s, a) = \frac{1}{|A(s)| - 1} \sum_{a, a' \in A(s), a' \neq a} \tilde{Q}_{\text{rel}}(s, a, a'). \tag{IV.4.22}$$

IV.5 Improving Sample Efficiency

In this section, we show how to improve efficiency in terms of required trajectory and preference samples. An efficient exploration/exploitation method allows to overcome the assumption of exhaustive action sampling, as mentioned in Section IV.1. Furthermore, we discuss how to reuse preference information from prior iterations for reducing the amount of required feedback.

IV.5.1 EXP3 for Preference Feedback

For solving the exploration/exploitation dilemma (Section II.2.2.b) we chose a method that is inspired by the EXP3 policy (Auer et al. 1995, 2003), because it is a stochastic action selector that can be applied to adversarial bandit problems. We chose adversarial bandits for reducing the amount of requirements that have to be considered. Furthermore, policies for the adversarial case always maintain an explorative part to be able to react to adversarial changes. In the preference case, this allows us to ensure that our policy will not converge before we have explored the expert's preference function sufficiently (cf. Section III.3.4). The exploration policy

$$\begin{aligned}
\pi^{\text{EXP3}}_{i+1}(a \mid s) &= \frac{\eta}{|A|} + (1 - \eta) \frac{\exp\left(\frac{\eta}{|A|} \tilde{G}_{s,a,i}\right)}{\sum\limits_{b \in A(s)} \exp\left(\frac{\eta}{|A|} \tilde{G}_{s,b,i}\right)}, \\
\tilde{G}_{s,a,i} &= \sum_{t=1}^{i} \tilde{g}_{s,a,t}, \\
\tilde{g}_{s,a,i} &= \frac{\tilde{Q}^{\pi_i}(s, a)}{\pi^{\text{EXP3}}_{i}(a \mid s)},
\end{aligned} \tag{IV.5.23}$$

is equivalent to the one of Audibert and Bubeck (2009), but we need to replace $g_{i,t}$ with a preference-based signal. $g_{i,t}$ is the expected outcome, i.e, Q^{π}, hence, we can replace this term with our preference estimated Q-function $\tilde{Q}^{\pi}(s, a)$, computed by both introduced approximations. We replaced η with $\frac{\eta}{|A|}$ to be able to define $\eta \in (0, 1]$ instead of $\eta \in (0, \frac{1}{|A|}]$. Figure IV.5.1 shows an example of the resulting distribution. The parameter η controls the slope

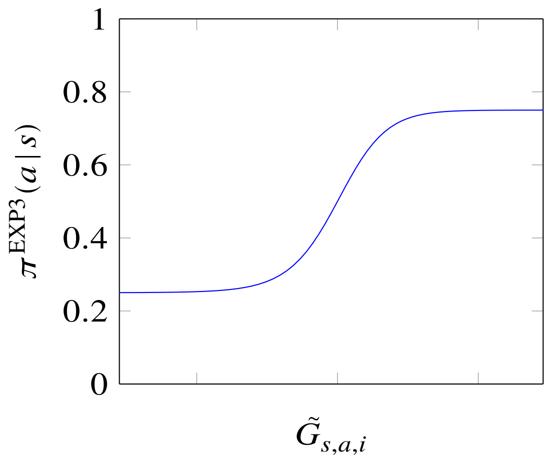


Figure IV.5.1.: EXP3 Distribution

as well as the level of the minimal and maximal plateaus. A low value results in a shallow slope, but extreme plateau levels. A high value decreases the gap between the plateaus, although with a steep slope. Therefore, exploration is maintained by either raising the minimum action probability or by a shallow slope.

IV.5.2 Rollouts for EPMC

If we want to compute additional preferences along trajectories, it is required to access states where we have feedback for different actions, as we require training information for $\tilde{Q}(s, a, a')$. Hence, we apply the current policy π until we observe a state that was already encountered in this iteration. In such a state, we then select an action that differs from the already encountered one, enabling us to reuse preference feedback for such overlapping states. These overlapping states can be seen as additional bandits where it is sensible to also apply an according exploration/exploitation strategy. Hence, we also use the EXP3 algorithm in these states. The resulting rollout strategy is shown as Algorithm 8. For the first rollout in each iteration, we always use the stochastic EXP3 strategy to ensure every action can be sampled.

IV.5.3 Sample-Efficient Policy Updates

The original algorithm assumed that it is possible to sample a sufficient number of states in each iteration for completely evaluating the policy, as shown in Algorithm 6. Hence, sufficient training samples for computing a policy are obtain in each iteration and all old training data is discarded. To overcome this problem, we apply a *state-action-reward-state-action* (SARSA) like update (II.2.24) and only move the policy into the direction of our new estimate, by apply-

Algorithm 8 Rollout Strategy for Every Visit Preference Monte Carlo

```
Require: initial state $s$, policy $\pi_i$, current trajectories $\boldsymbol{\tau}^{[i]}$, hyper-parameter $\eta$, horizon $h$
 1: while $A(s) \neq \emptyset$ and $|\boldsymbol{\tau}| < h$ do
 2:     if $s \in \boldsymbol{\tau}^{[i]}$ or $\boldsymbol{\tau}^{[i]} = \emptyset$ then
 3:         $a \sim \pi_i^{\text{EXP3}}(a \mid s)$                ▷ Apply EXP3 policy
 4:     else
 5:         $a \sim \pi_i(a \mid s)$                ▷ Apply current, best policy
 6:     end if
 7:     $\boldsymbol{\tau} \leftarrow \boldsymbol{\tau} \cup \{(s, a)\}$
 8:     $s \sim \delta(s' \mid s, a)$                ▷ Apply transition function
 9: end while
10: return trajectory $\boldsymbol{\tau}$
```

ing a learning rate. Therefore, we do not completely disregard already learned information but overwrite the estimate gradually. To update the tabular representation (cf. Section II.3.1), we use the gradient-based update

$$\tilde{Q}^{\pi_{i+1}}(s, a, a') = (1 - \alpha)\tilde{Q}^{\pi_i}(s, a, a') + \alpha, \tilde{Q}^{\mathscr{E}_i}(s, a, a') \qquad \text{(IV.5.24)}$$

with $\tilde{Q}^{\mathscr{E}_i}(s, a, a')$ as the approximation obtained from the action preferences obtained in iteration i (see Algorithm 7). As a result, training data from past iterations is reused, but its influence diminishes when new samples are obtained.

IV.6 Experiments

The algorithms are evaluated by applying them to the cart pole, mountain car and acrobot domain, as described in Appendix A. Preferences are obtained by comparing the discounted return (II.2.2) of both trajectories with a discount factor $\gamma = 0.98$. In the mountain car and acrobot domain, the return is defined as a penalty for the number of steps required to reach a goal state. The cart pole domain uses a positive reward for each step the system is stable.

All hyper-parameters (λ, α, η) are tuned, based on a grid search with $10^3 = 1000$ different combinations. For the numeric reward comparison, we use SARSA (cf. Section II.2.3.b; Rummery and Niranjan 1994; Sutton and Barto 1998), with an ϵ_n-greedy decay scheme (Auer et al. 2002). Its hyper-parameters (λ, α, ϵ_c, ϵ_d) are selected based on $5^4 = 625$ grid trials. The presented results are based on a greedy policy, derived from the current exploration policy. This is required as we are subject to an always-explore policy, overcoming premature convergence problems, as explained in Section IV.5.1. All results are presented via the median return and quartiles obtained over 100 trials.

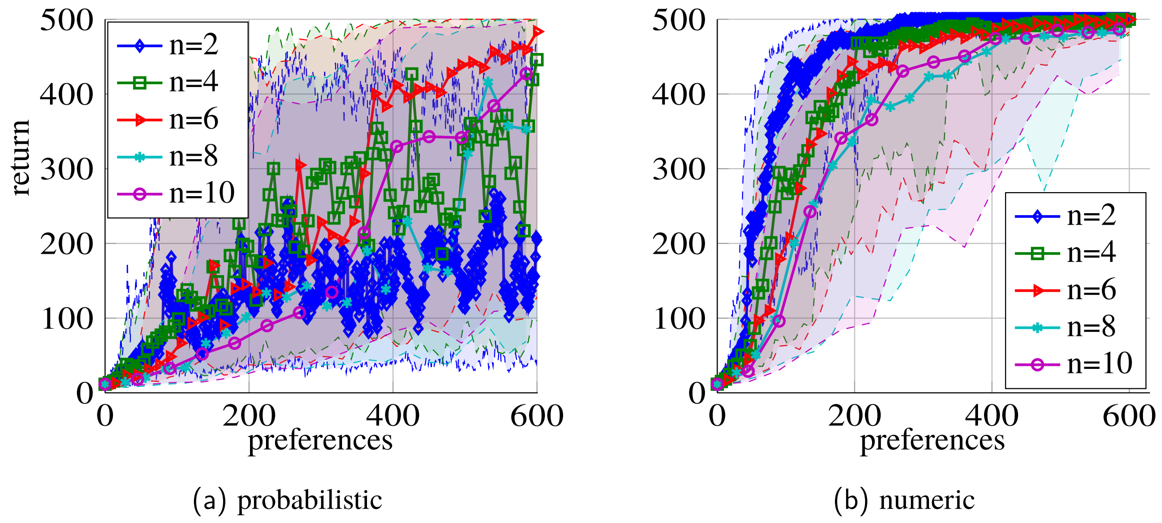


Figure IV.6.2.: Cart pole results for EPMC

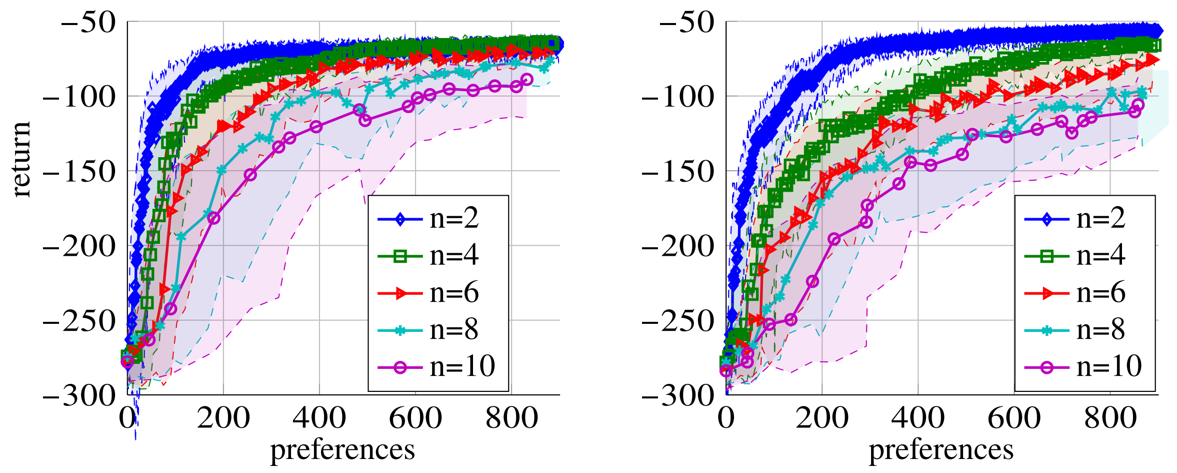
IV.6.1 Preference Efficiency

This section shows the results obtained with different numbers of trajectory samples per iteration ($n = 2$, 4, 6, 8 and 10, cf. Algorithm 7). All configurations generate the same number of preferences in total. The algorithm can access $\frac{n(n-1)}{2}$ preferences in each iteration, because we obtain a preference for every possible trajectory pair. Therefore, a lower number of trajectories per iteration (parameter n) results in a higher number of trajectories and policy updates required to obtain the same number of preferences. We always compare the probabilistic (Section IV.3) approximation with the numeric (Section IV.4) counterpart.

In the cart pole domain with numeric approximation, one preference per iteration ($n = 2$) is sufficient to achieve stable convergence, as shown by Figure IV.6.2b. We can also see that more preferences per iteration do not yield as much information as obtaining new preferences after a policy improvement step. The configurations that obtain fewer preferences per iteration (low n) and therefore update the policy more often perform better.

When using the probabilistic approximation, 15 preferences per iteration ($n = 6$) are required to achieve stable convergence, as shown by Figure IV.6.2a. However, more preferences per iteration do not improve the results, as observed in the numeric setting. Hence, it seems obtaining preferences from a new approximation of the optimal policy is better than obtaining more preferences in a single iteration. We can also determine that the additional information in form of the uniform reward prior improves the approximation, because the numeric variant substantially outperforms the probabilistic approach.

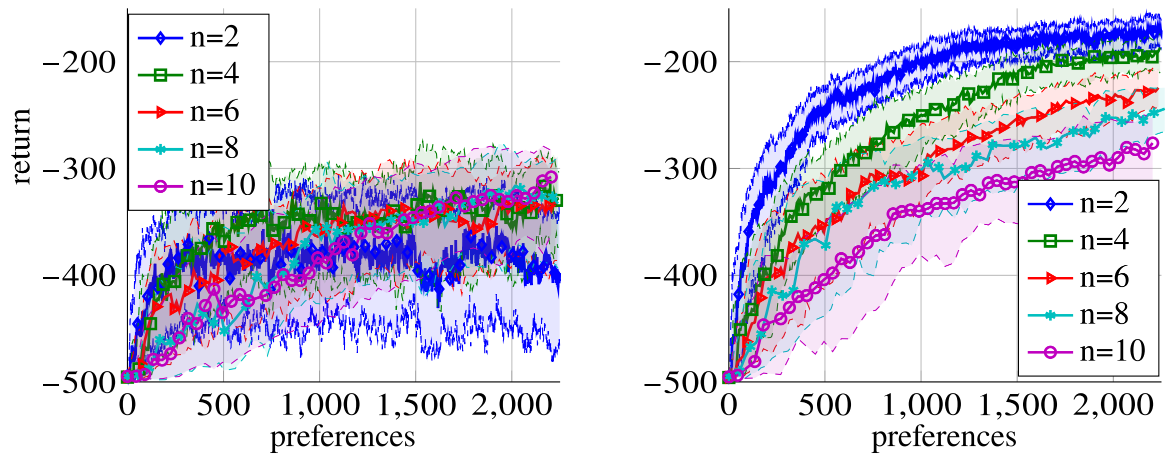
Figure IV.6.3 show the results from the mountain car domain where both approximations perform comparably. The probabilistic version converges slightly faster, but the numeric variant



(a) probabilistic

(b) numeric

Figure IV.6.3.: Mountain car results for EPMC



(a) probabilistic

(b) numeric

Figure IV.6.4.: Acrobot results for EPMC

results in a slightly better policy. As in the cart pole domain, we can observe that it is beneficial to update the sampling policy before requesting additional preferences (low n). In the mountain car domain, this also holds for the probabilistic approximation when only obtaining one preference per iteration.

For the acrobot domain (Figure IV.6.4), we can see that the numeric approximation outperforms the probabilistic variant substantially. This further supports our claim that it is useful to incorporate prior knowledge. The numeric variant does again not benefit from obtaining more than one preference per iteration ($n = 2$), whereas the probabilistic approximation requires 45 ($n = 10$) preferences per iteration to converge.

In general, we can conclude that it is more important to evaluate policy estimates quickly, by obtaining new preferences after updating the policy, than to improve the policy evaluation. It is necessary to obtain a minimal number of preferences per iteration before this effect is visible, but the value depends on the approximation technique. The numeric variant can converge with only one preference per iteration whereas the probabilistic method may require multiple preferences. However, both methods converge in all evaluated domains, given a sufficient amount preferences per iteration.

The original PBPI algorithm requests $k\frac{|A|(|A|-1)}{2}n$ preferences in each iteration, as it compares all actions, based on n rollouts each for k state samples. The EPMC algorithm requests $\frac{n(n-1)}{2}$ preferences each iteration, as it obtains a ranking for n rollouts. Hence, EPMC is more preference efficient as long as $n - 1 \ll k|A|(|A| - 1)$. The setup presented by Fürnkranz et al. (2012) uses values $k, n \geq 10$, resulting in at least $10\frac{3(3-1)}{2}10 = 300$ preferences per iteration for the three experiment domains, as they discretize the action space into $|A| = 3$ actions. However, $n \leq 10$ is sufficient for convergence of the EPMC algorithm. Therefore, we do not show graphical comparisons between PBPI and EPMC as the PBPI algorithm only manages to complete one iteration with the number of preferences required by EPMC for convergence in the cart pole and mountain car domain.

As we use a purely incremental algorithm that scales with $O(n)$ concerning the obtained preferences, it is very fast. Approximating the solution to the temporal credit assignment problem and updating the policy is possible in a fraction of a second in all analyzed domains.

IV.6.2 Exploration

The presented EPMC algorithm is subject to several hyper-parameters. The effect of η is of special interest, as it controls the the exploration/exploitation tradeoff (cf. Section II.2.2.b). In contrast to classic *reinforcement learning* (RL), it is not sufficient to explore the transition dynamics, but we also need to explore the expert's preference function, as explained in Section III.3.4. This section shows the effect of changing the η parameter, using contour plots. Each plot (Figures IV.6.5- IV.6.7) is based on the quartile results obtained with $n = 2$, also used in Section IV.6.1. The x-axis relates to the number of obtained preferences, whereas the y-axis shows different values for η. The color is the trajectory return value of the quartile, ranging from blue (low return) to red (high return).

Figure IV.6.5 shows the cart pole domain, where we can see that the probabilistic approximation struggles with convergence. The upper quartile shows convergence around $\eta = 0.7$, but the lower and median quartile do not converge within 600 preferences. Using the numeric approximation, the algorithm converges with nearly all values for η, but tuning the value improves the

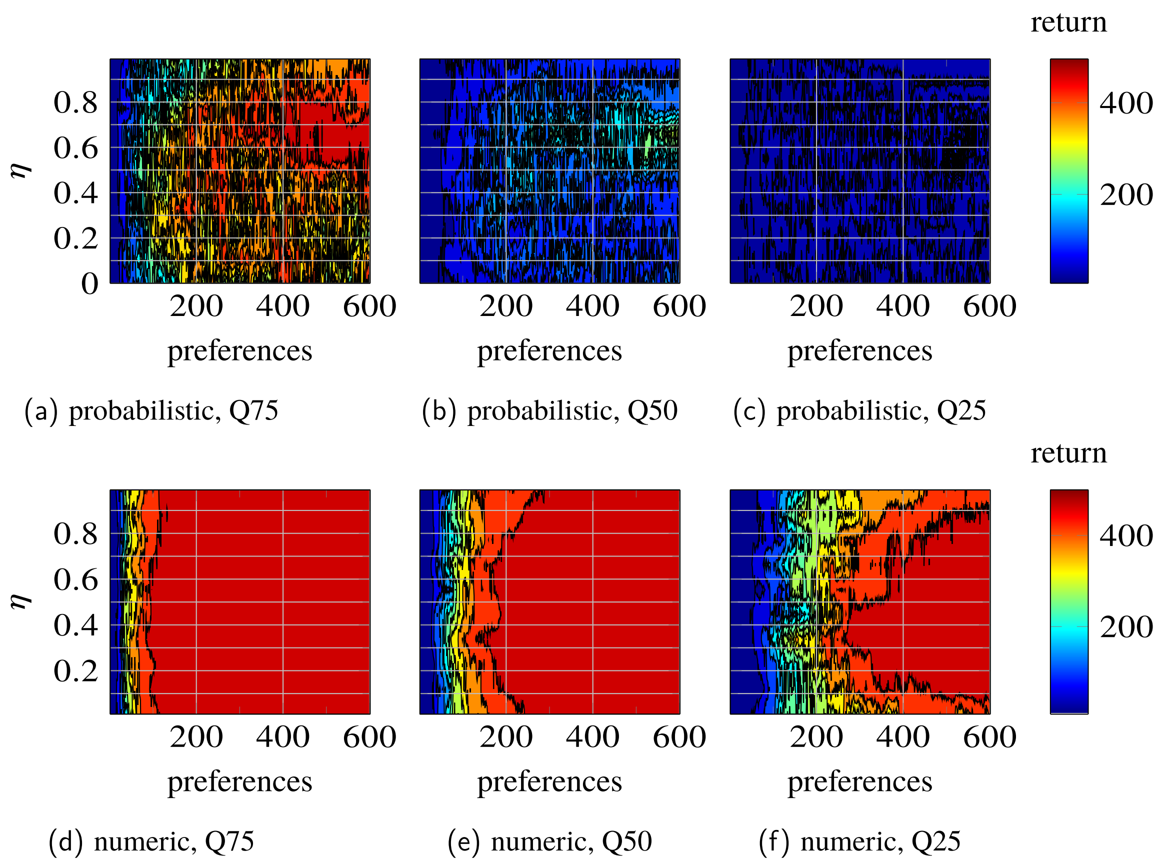


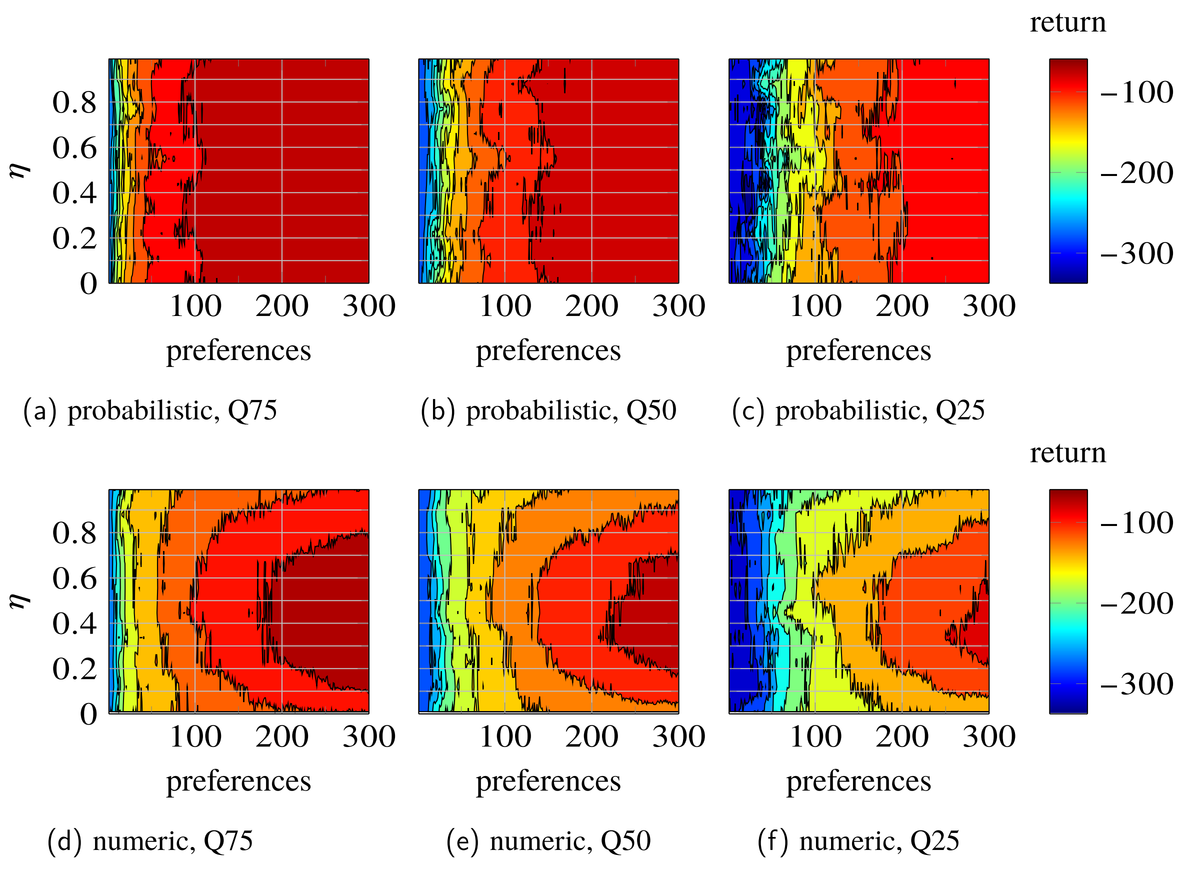
Figure IV.6.5.: Influence of η in the cart pole domain

results. Especially the lower quartile benefits from selecting the best ($\eta = 0.35$) value. Extreme values for η decrease performance substantially in the 25% quartile.

In the mountain car domain, shown in Figure IV.6.6, we can again observe that both approaches converge smoothly. The probabilistic approach is virtually independent from tuning η, but the results in the numeric setting depend greatly on tuning. As in the cart pole domain, we can observe that extreme values are problematic and that the optimum is around $\eta = 0.35$.

The probabilistic approximation hardly converges with $n = 2$ in the acrobot domain, as it was already observable in Figure IV.6.4a. Figure IV.6.7 also shows that extreme η values are not beneficial for speeding up convergence when using the numeric heuristic. We can again observe a single optimum, this time around $\eta = 0.6$.

In general, we can conclude that a well-defined exploration/exploitation tradeoff is substantial for PBRL when using an efficient approximator for the temporal credit assignment problem.



(a) probabilistic, Q75 (b) probabilistic, Q50 (c) probabilistic, Q25

(d) numeric, Q75 (e) numeric, Q50 (f) numeric, Q25

Figure IV.6.6.: Influence of η in the mountain car domain

IV.6.3 Trajectory Efficiency

In Figure IV.6.8, we show a comparison in terms of trajectory efficiency. The results are obtained with $n = 10$, as this setting derives the highest number of preferences from a fixed number of trajectories. We compare with the SARSA algorithm to determine the information loss of preferences compared to numeric feedback. Furthermore, we also considered a setting where we assume that the preference difference values $x_{i,j}$ (see Section IV.4) are known, derived from the return difference. We can see that the reward-based SARSA significantly outperforms the preference algorithms in the cart pole domain. A comparable result can be observed in the acrobot domain, but to a lesser extend. SARSA also outperforms the preference-based configurations in the mountain car domain, but only slightly. Interestingly, for obtaining a SARSA configuration that can outperform the preference setting in the mountain car domain, we had to sample 625 additional hyper-parameter configurations, resulting in 1250 configurations total. This shows that it can be more difficult to tune SARSA than the preference-based algorithms.

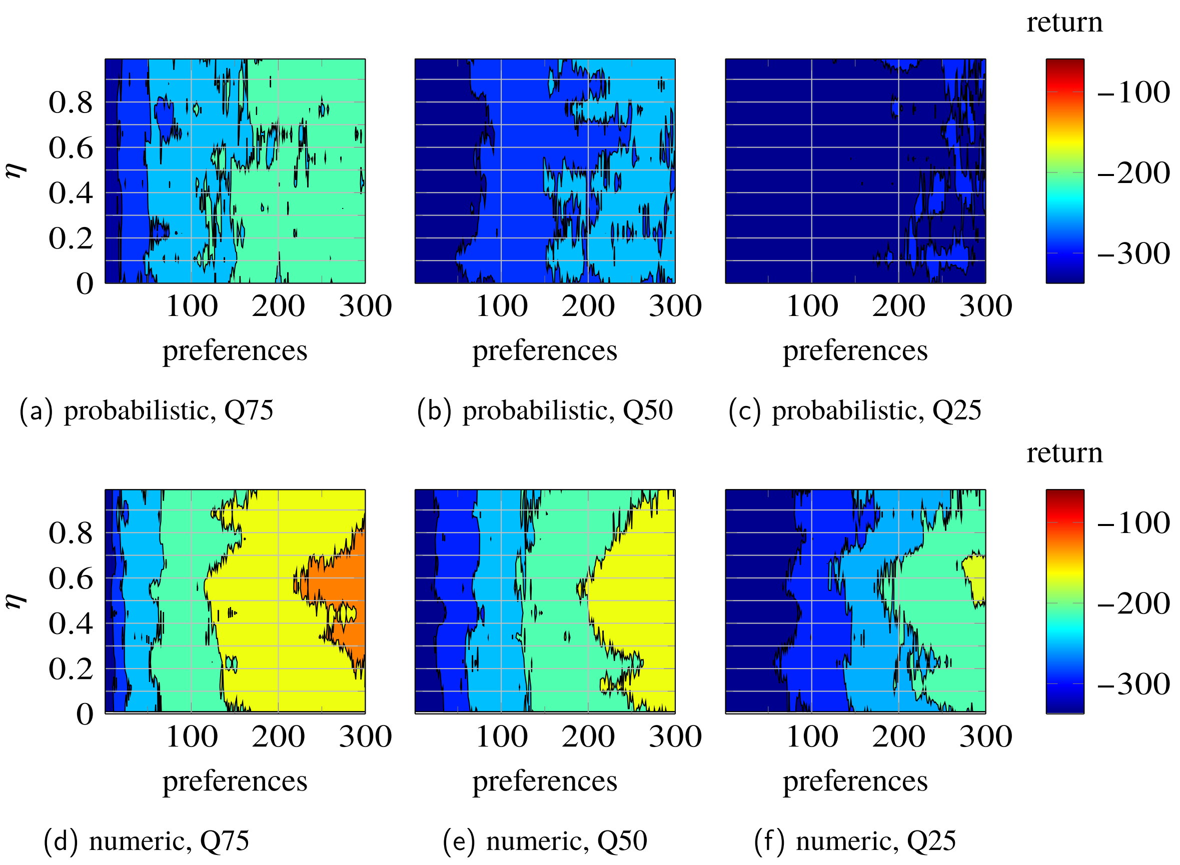
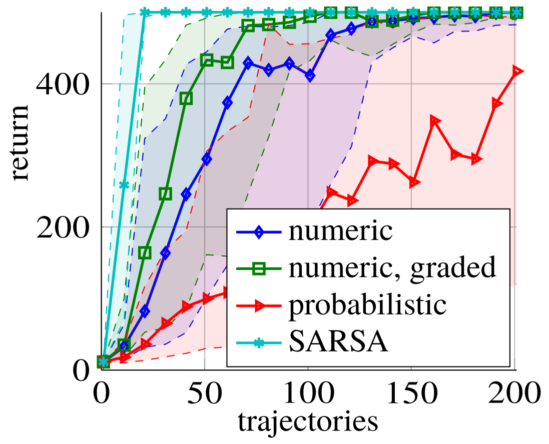


Figure IV.6.7.: Influence of η in the acrobot domain

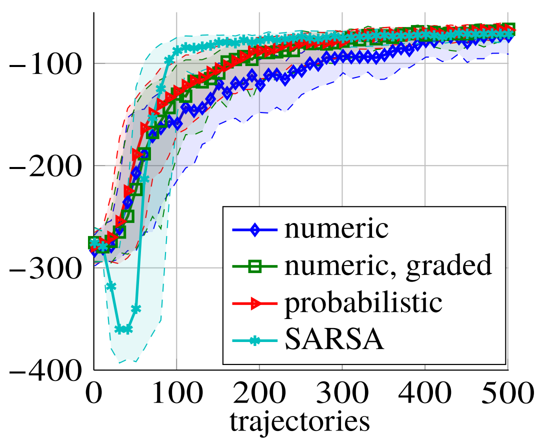
The configuration with access to the exact value $x_{i,j}$ is only slightly better in all domains. Hence, the performance advantage of numeric rewards is probably more related to the fact that we can obtain feedback for every state-action pair and not only overlapping states.

IV.6.4 Violating the Low-Variance Assumption

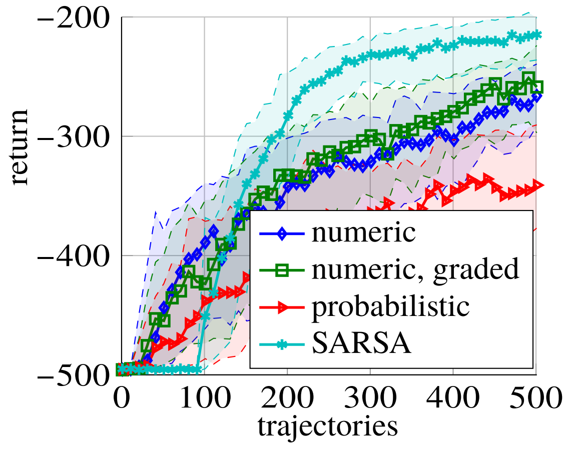
Finally, we determine the effects of violating the low-variance assumption for the numeric temporal credit assignment method (cf. Section IV.4). We compare the numeric approximation with SARSA in a simple 5×5 grid world with the task to move from the upper left corner to the lower right. The step penalties are sampled from a Gaussian distribution $\mathcal{N}(r \mid \mu, \sigma)$. Table IV.1 shows the absolute and relative difference between the optimal SARSA policy and the preference-based result. It can be seen that the policy degrades gracefully with small violations of the low-variance assumption. Only extreme violations result in substantial quality loss. Furthermore, the SARSA method results in only slightly better policies than the EPMC approach when using a low variance.



(a) Cart pole



(b) Mountain car



(c) Acrobot

Figure IV.6.8.: Comparison of EPMC algorithms with $n = 10$

μ	σ	binary feedback	graded feedback
-10	1	-3.2 (4.3 %)	-2.7 (3.6 %)
-10	2	-7.3 (10.2 %)	-6.7 (9.5 %)
-10	3.33	-11.2 (16.4 %)	-7.6 (11.7 %)
-50	1	-1.1 (0.3 %)	-0.7 (0.2 %)
-50	2	-4.2 (1.1 %)	-3.5 (0.9 %)
-50	5	-16.3 (4.3 %)	-13.7 (3.5 %)
-50	10	-33.9 (9.5 %)	-26.2 (6.3 %)
-50	16.66	-53.9 (17.1 %)	-38.6 (11.5 %)

Table IV.1.: Effects of violating the low-variance assumption

IV.7 Conclusion

We showed that efficient approximations for solutions to the temporal credit assignment problem result in a substantial reduction in terms of required preferences. Furthermore, both proposed methods converge in all evaluated domains, given a sufficient preference amount per iteration. Hence, both methods show the benefit of solving the temporal credit assignment problem explicitly, answering Research Question 1.A.

Considering the substantial improvements over PBPI, it is clear that EPMC is able to use trajectory preferences more efficiently. Therefore, we can determine that the introduced approaches to solve the temporal credit assignment problem are substantial for a speed up in PBRL, answering Research Question 1.B. We also showed that the binarization problem (cf. Research Question 1.C) is less important because information about the degree of preferences does not result in substantial improvements. Hence, explicit solutions to the temporal credit assignment problem should be investigated further.

Besides the temporal credit assignment, it is relevant to update policies early and re-evaluate the approximation, giving insight into Research Question 1.D. Interestingly, obtaining more preferences per iteration than required for convergence is usually not beneficial. However, the amount of required preferences differs and can not be easily determined. We also observed that a well-defined exploration/exploitation tradeoff is important for a learning speed up. This can be seen by the influence of different EXP3 configurations. However, the proposed adaption of EXP3 is probably not suited for every domain and algorithm, as some experiments do not benefit from optimizing the tradeoff. We can also determined that prior assumptions over the reward are beneficial for learning and should be considered in future approaches. Hence, we can answer Research Question 1.D by stating that temporal credit assignment, fast policy re-evaluations, well defined exploration/exploitation trade-offs and suitable priors are important for improving preference sample efficiency.

Chapter V

Learning Utility Functions from Human Preferences[34]

[34] This chapter is based on Wirth and Fürnkranz (2015).

In the previous chapter, we have shown that *preference-based reinforcement learning* (PBRL) is substantially limited by the problem that we can not obtain feedback for every state-action along the sampled trajectories. Furthermore, we can not obtain feedback without asking the expert explicitly. Hence, we want to investigate methods that allow better approximations of state-action outcomes and generalize to trajectories that have not been explicitly evaluated. Utility functions (cf. Section II.5.3.a) are a possible solution as they describe a continuous model that can be learned from preferences. Additionally, PBRL algorithms usually run in cycles (cf. Section III.2.1), but learning a policy is a problem that can be solved independently if we can access numeric utilities. The iterative process is only required to solve the exploration/exploitation dilemma, introduced in Section II.2.2.b. In case sufficient samples to calculate an approximate policy are available, further sample collection is not required. Hence, when using a data basis that is large enough to be able to assume a sufficient sample count, we can view the policy optimization process as a batch-learning problem. This enables us to analyze the utility learning problem in more detail, without considering the influence of the exploration process. For obtaining an even more focused setup, we use state (and action) preferences with long-term expectation in this chapter (Section III.3.1), allowing us to circumvent the temporal assignment problem, mentioned in Section III.3.3.

The remaining question is how to learn a utility function based on human preferences. It can not be assumed that it is possible to learn a utility function that is capable of reconstructing the given preferences perfectly, because of several problems. Human judgment is not flawless and can be inconsistent in the sense that the given preferences may contain cycles or violate the total order assumption introduced in Section III.1. Furthermore, humans may define contradicting preferences due to the stochastic feedback function ρ or in case multiple humans are involved. Additionally, the set of obtained preferences ζ may not be representative for the feedback function ρ and induce a bias. Hence, we have to determine the problems that arise by using human preferences and if it is required to deal with them explicitly.

Therefore, we analyze the following research questions in this chapter:

2.A Can we generalize human preference-based feedback signals using utility functions?

2.B What problems occur when learning from human preferences?

2.C Can we achieve substantial performance improvements by explicitly dealing with human errors or bias?

In Section V.1, we explain why we use chess as a problem to introduce our approach. Section V.2 discusses the information that can typically be found in annotated chess games. In particular, we focus on *portable game notation* (PGN) files with *numeric annotation glyph* (NAG) (Edwards 1994), a widely available data format. Section V.3 shows how to extract preference information from such data and how to use this information for learning an evaluation function via an object ranking algorithm. In our experimental setup (Section V.4), we train a support-vector machine with the preference data, based on the state feature values given by a strong chess engine. This enables the creation of a new heuristic evaluation function by using the learned (linear) SVM model. The quality of the resulting function is evaluated in a chess engine tournament. We discuss the results of several different experiments that follow this general pattern in Section V.5. In Section V.6.1, we introduce an improved noise handling method, based on a sigmoid loss, and analyze the effect.

V.1 Domain Overview

In the game of chess, qualitative human feedback is amply available in the form of game annotations. Game annotations have been provided by strong chess players for centuries, and are amply available in chess books and in recent years also in increasing amounts in online chess databases. The annotations often do not only consist of textual annotations, which are harder to use, but use a language of qualitative annotation symbols, which we will describe later in this chapter.

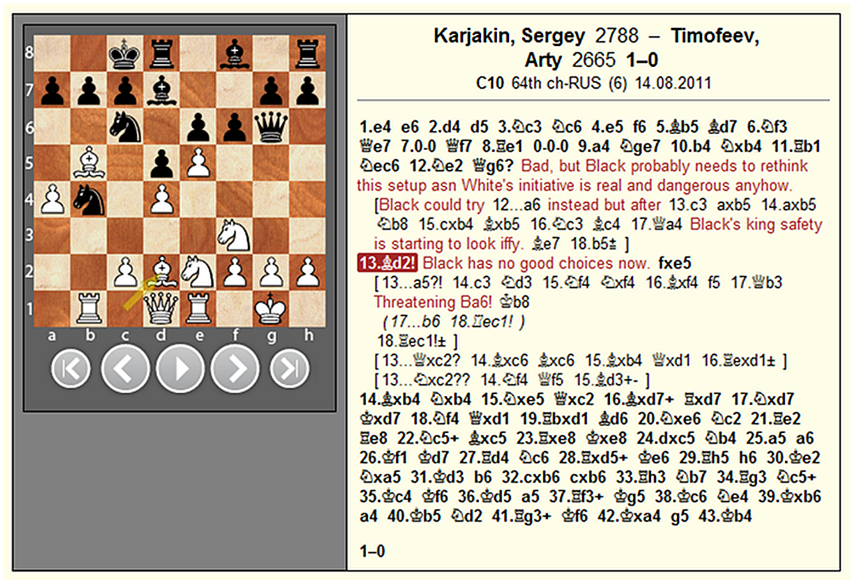
However, this rich source of information about the game is rarely used in the literature on machine learning in chess (Skiena 1986; Fürnkranz 1996) or in games in general (Fürnkranz and Kubat 2001; Ghory 2004; Bowling et al. 2006; Fürnkranz 2010), despite evidence that it is possible to utilize this information (Gomboc et al. 2004). Much of the work in this area has concentrated on the application of reinforcement learning algorithms to learn meaningful evaluation functions (Baxter et al. 2000; Beal and Smith 2001; Droste and Fürnkranz 2008). These approaches have all been modeled after the success of TD-Gammon (Tesauro 2002), a learning system that uses temporal-difference learning (Sutton 1988) for training a game evaluation function (Tesauro 1992). However, all these algorithms were trained exclusively on self-play, entirely ignoring human feedback that is readily available in annotated game databases.

Chess is also especially interesting, as many high-quality AI agents are already available.[35] They usually use hand-crafted evaluation functions, directly defining a policy. Hence, we can try to learn the parameters of such a parametric policy from preferences, circumventing the problem of defining a suitable function approximator for the policy (cf. Section II.3).

V.2 Game Annotations in Chess

Chess is a game of great interest, which has generated a large amount of literature that analyzes the game. Particularly popular are game annotations, which are regularly published after

[35] `https://stockfishchess.org/`, `https://komodochess.com/`, `http://hem.bredband.net/petero2b/javachess/index.html`



Karjakin, Sergey 2788 – **Timofeev, Arty** 2665 **1–0**
C10 64th ch-RUS (6) 14.08.2011

1.e4 e6 2.d4 d5 3.♘c3 ♘c6 4.e5 f6 5.♗b5 ♗d7 6.♘f3 ♕e7 7.0-0 ♕f7 8.♖e1 0-0-0 9.a4 ♘ge7 10.b4 ♘xb4 11.♖b1 ♘ec6 12.♘e2 ♕g6? Bad, but Black probably needs to rethink this setup asn White's initiative is real and dangerous anyhow.
[Black could try 12...a6 instead but after 13.c3 axb5 14.axb5 ♘b8 15.cxb4 ♗xb5 16.♘c3 ♗c4 17.♕a4 Black's king safety is starting to look iffy. ♗e7 18.b5±]
13.♗d2! Black has no good choices now. **fxe5**
[13...a5?! 14.c3 ♘d3 15.♘f4 ♘xf4 16.♗xf4 f5 17.♕b3 Threatening Ba6! ♔b8
(17...b6 18.♖ec1!)
18.♖ec1!±]
[13...♕xc2? 14.♗xc6 ♗xc6 15.♗xb4 ♕xd1 16.♖exd1±]
[13...♘xc2?? 14.♘f4 ♕f5 15.♗d3+-]
14.♗xb4 ♘xb4 15.♘xe5 ♕xc2 16.♗xd7+ ♖xd7 17.♘xd7 ♔xd7 18.♘f4 ♕xd1 19.♖bxd1 ♗d6 20.♘xe6 ♘c2 21.♖e2 ♖e8 22.♘c5+ ♗xc5 23.♖xe8 ♔xe8 24.dxc5 ♘b4 25.a5 a6 26.♔f1 ♔d7 27.♖d4 ♘c6 28.♖xd5+ ♔e6 29.♖h5 h6 30.♔e2 ♘xa5 31.♔d3 b6 32.cxb6 cxb6 33.♖h3 ♘b7 34.♖g3 ♘c5+ 35.♔c4 ♔f6 36.♔d5 a5 37.♖f3+ ♔g5 38.♔c6 ♘e4 39.♔xb6 a4 40.♔b5 ♘d2 41.♖g3+ ♔f6 42.♔xa4 g5 43.♔b4

1–0

Figure V.2.1.: An annotated chess game taken from `http://chessbase.com/`.

important or interesting games have been played in tournaments. These annotations reflect the analysis of a particular game by a (typically) strong professional chess player. They have been produced without time constraints, and the annotators can resort to any means they deem necessary for improving their judgment, including the consultation of colleagues, books, or computers. Thus, these annotations are usually of a high quality, but are still noisy as different players have different play styles. In fact, contradicting annotations may occur.

Annotated chess games are amply available, not only in chess books or magazines. Chess databases store millions of games, many of them annotated. For example, the largest database distributed by the company *Chessbase*,[36] contains over five million games, more than 66,000 of which are annotated. Chess players of all strengths use them regularly to study the game or to prepare against their next opponent. Chess players annotate games with alternative lines of play and/or textual descriptions of the evaluation of certain lines or positions. An open format for storing chess games and annotations is the so-called *portable game notation* (PGN) (Edwards 1994). PGN games are represented as a sequence of moves. The moves themselves are recorded in standard algebraic notation, which specifies the move number, the moving piece, its originating square (optionally), and its destination square. For example, in the chess position shown in Figure V.2.1, the empty square in the lower left corner has the co-ordinates a1, and the position of the black rook in the upper right corner would be denoted as ♖h8 (or Rh8 in the source file). Most importantly, however, typical events in a chess game can be encoded with a standardized set of symbols, which has been popularized by the Chess Informant book series.[37]

[36] `http://www.chessbase.com/`
[37] `http://www.chessinformant.rs/`

There is a great variety of these NAGs, referring to properties of the positions (e.g., attacking chances, pawn structures, etc.), the moves (e.g., forced moves, better alternatives, etc.), or to external properties of the game (such as time constraints). Many of them are only rarely used. In this work, we will focus on the most commonly used symbols, which annotate the quality of moves and positions:

- *move evaluation:* Each move can be annotated with a symbol indicating its quality. Six symbols are commonly used:
  - very poor move (??),
  - poor move (?),
  - speculative move (?!),
  - interesting move (!?),
  - good move (!),
  - very good move (!!).
- *position evaluation:* Each move can be annotated with a symbol indicating the quality of the position it is leading to:
  - white has a decisive advantage ($+-$),
  - white has a moderate advantage ($\pm$),
  - white has a slight advantage ($\stackrel{+}{=}$),
  - equal chances for both sides ($=$),
  - black has a slight advantage ($\stackrel{=}{+}$),
  - black has a moderate advantage ($\mp$),
  - black has a decisive advantage ($-+$),
  - the evaluation is unclear (∞).

We will denote the set annotation symbols with $\mathcal{A} = \mathcal{A}_S \cup \mathcal{A}_A$ where $\mathcal{A}_S$ are position, or state, annotations and $\mathcal{A}_A$ are move, or action, annotations. $\mathcal{A}(a, s)$ are the annotations associated with a given move a in a state s. For single annotations, we use the symbol $\mathring{a} \in \mathcal{A}$. Move and position evaluations can be organized into a partial order which we denote with the symbol $\sqsupset$. The move evaluations can be ordered as

$$+- \ \sqsupset \ \pm \ \sqsupset \ \stackrel{+}{=} \ \sqsupset = \sqsupset \ \stackrel{=}{+} \ \sqsupset \ \mp \ \sqsupset \ -+$$

and the position evaluations as

$$!! \sqsupset \; ! \sqsupset \; !? \sqsupset \; ?! \sqsupset \; ? \sqsupset \; ??.$$

Note that, even though there is a certain correlation between position and move annotations (good moves tend to lead to better positions and bad moves tend to lead to worse positions), they are not interchangeable. A very good move may be the only move that saves the player from imminent doom, but must not necessarily lead to a very good position. Conversely, a bad move may be a move that misses a chance to mate the opponent right away, but the resulting position may still be good for the player. For this reason, $\sqsupset$ is partial in the sense that it is only defined on $\mathcal{A}_A \times \mathcal{A}_A$ and $\mathcal{A}_S \times \mathcal{A}_S$, but not on $\mathcal{A}_A \times \mathcal{A}_S$.

In addition to annotating games with NAG symbols, annotators can also add textual comments and variations. These complement the moves that were actually played in the game with alternative lines of play that could have happened or that illustrate the assessment of the annotator. Typically, such variations are short move sequences that lead to more promising states than the moves played in the actual game. Variations can also have NAG symbols, and may contain sub-variations.

Figure V.2.1 shows an example of an annotated chess-game. The left-hand side shows the game position after the 13th move of white. Here, black is in a difficult position because his last move 12...♕g6? was a mistake. After the strong answer by white (13.♗d2!), black has several alternatives to the move 13...f×e5 played in the game. 13...a5?! is best, but even here white has the upper hand at the end of the variation (18.♖ec1!±). 13...♕×c2? is a mistake, and a third alternative, 13...♘×c2?? is even worse, eventually leading to a position that is clearly lost for black (+−).

It is important to note that this feedback is of qualitative nature, i.e., it is not clear what the expected reward is in terms of, e.g., percentage of won games from a position with evaluation ±. However, according to the above-mentioned relation $\sqsupset$, it is clear that positions with evaluation ± are preferable to positions with evaluation ⩲ or worse (=, ⩱, ∓, −+). Similarly, among the options available for black at his 13th move, we can establish the ordering 13...f×e5 $\succ$ 13...a5?! $\succ$ 13...♕×c2? $\succ$ 13...♘×c2?? , where the symbol $\succ$ is used to annotate a preference (cf. Section V.3).

As we will see in the following, we will collect preference statements over positions. For this, we have to uniquely identify chess positions. Chess positions can be efficiently represented in the *Forsyth-Edwards Notation* (FEN), which is a serialized, textual representation of the game board, capturing all data that is required to uniquely identify a chess state.

V.3 Learning an Evaluation Function from Annotated Games

Our approach of learning from qualitative game annotations is based on the idea of transforming the notation symbols into preference statements between pairs of positions, which we describe in more detail in Sections V.3.1 and V.3.2. As annotations are an evaluation of the expected outcome of a game, the resulting state preferences are a measure of long-term expectations. Each such preference may then be viewed as a constraint on a utility function for chess positions, which can be learned with state-of-the-art machine learning algorithms such as support-vector machines (Section V.3.4). The utility function is purely based on the position information, reducing the problem to ranking all positions with an object ranking approach, as described in Section II.5.2.

V.3.1 Generating Preferences from Position Annotations

The training data that are needed for an object ranking algorithm can be generated from game annotations of the type discussed in Section V.2. First, we parse each game in the database $\mathscr{G}$, replaying all moves so that we are able to generate all occurring positions. For each move a that has a set of associated position annotation symbols $\mathscr{A}_A(a, s)$ we associate the annotation symbols with the position s' that results after playing each move.

Algorithm 9 shows the algorithm for generating state preferences. The first loop collects for each game a list of all positions that have a position annotation. In a second loop, all encountered annotated positions are compared. For all pairs of positions (s', s'') where s' received a higher evaluation than s'', a corresponding position preference is generated. Position s' receives a higher evaluation than s'' if the associated position evaluation $å'$ prefers white to a stronger degree than the annotation $å''$ associated with s'' (position evaluations are always viewed from the white player).

As can be seen, we only compare position preferences within the same game and not across games. This has several reasons. One is, of course, that a comparison of all annotated position pairs in the entire database would be computationally infeasible. However, it is also not clear whether this would be a sensible thing to do. For example, it is possible that different annotators reach different conclusions for the same position. A decisive advantage (−+,+−) for one player may only be annotated as moderate advantage (∓,±) by another annotator. For this reason, we only compare annotations within the same annotated game.

Algorithm 9 Preference generation from position annotations

```
Require: database of games $\mathcal{G}$
 1: $\zeta \leftarrow \emptyset$,
 2: for all $g \in \mathcal{G}$ do
 3:     $pairs \leftarrow \emptyset$,                              ▷ Clear saved annotations every game
 4:     $seen \leftarrow \emptyset$
 5:     for all $s \in g$ do                                      ▷ Collect all annotated states
 6:         if $\exists \mathring{a} \in \mathcal{A}_S(s)$ then
 7:             $pairs \leftarrow pairs \cup \{(s, \mathring{a})\}$      ▷ Create pairs for states in the same game
 8:             $seen \leftarrow seen \cup \{s\}$
 9:         end if
10:     end for
11:     for all $s' \in seen$ do
12:         for all $s'' \in seen$ do
13:             for all $(s', \mathring{a}'), (s'', \mathring{a}'') \in pairs$ do     ▷ Create preferences for all pairs
14:                 if $\mathring{a}' \sqsupset \mathring{a}''$ then
15:                     $\zeta \leftarrow \zeta \cup \{s' \succ s''\}$
16:                 end if
17:             end for
18:         end for
19:     end for
20: end for
21: return state preferences $\zeta$
```

V.3.2 Generating Preferences from Move Annotations

The problem of move selection in a game may be viewed as a label ranking problem (cf. Section II.5.2), where the game positions are the context, and the moves are the labels that have to be ranked. However, as each move corresponds to a unique successor position, move preferences may also interpreted as position preferences between the corresponding successor positions. We chose this approach because it allows us to treat move preferences and position preferences uniformly.

Algorithm 10 shows how we generate position preferences from move annotations. First, each game $g \in \mathcal{G}$ in PGN format is parsed and all positions are generated, as in the case of state preferences. However, here we have to store position/move/annotation triplets $(s, a, \mathring{a})$ because now we only can compare successor positions of the same original position s (cf. end of Section V.2).

After collecting this data for every game, all triples containing the same FEN position are compared. The NAG symbols for all moves associated with this position are compared and the corresponding preferences between the positions resulting from these moves are added to the set

Algorithm 10 Preference generation from move annotations

```
Require: list of games $\mathcal{G}$
 1: $\zeta \leftarrow \emptyset$,
 2: for all $g \in \mathcal{G}$ do
 3:     $triples \leftarrow \emptyset$,
 4:     $seen \leftarrow \emptyset$
 5:     for all $(s, a) \in g$ do                                   ▷ Collect all annotated actions
 6:         if $\exists å \in \mathcal{A}_A(a, s)$ then
 7:             $triples \leftarrow triples \cup \{(s, a, å)\}$      ▷ Create triples for actions in the same game
 8:             $seen \leftarrow seen \cup \{s\}$
 9:         end if
10:     end for
11:     for all $s \in seen$ do
12:         for all $(a_1, a_2)$ s.t. $(s, a_1, å_1), (s, a_2, å_2) \in triples$ do ▷ Create preferences for all triples
13:             if $å_1 \sqsupset å_2$ then
14:                 $s_1 \sim \delta(s_1 \mid s, a_1)$                ▷ Obtain successor state
15:                 $s_2 \sim \delta(s_2 \mid s, a_2)$                ▷ Obtain successor state
16:                 if WhiteMove($a$) then      ▷ Consider that annotations are active player dependent
17:                     $\zeta \leftarrow \zeta \cup \{s_1 \succ s_2\}$
18:                 else if BlackMove($a$) then
19:                     $\zeta \leftarrow \zeta \cup \{s_2 \succ s_1\}$
20:                 end if
21:             end if
22:         end for
23:     end for
24: end for
25: return state preferences $\zeta$
```

of preferences. Here, we have to take care that position evaluations are always viewed from the white player. Thus, a better move for black will result in a worse position for white.

V.3.3 Position Evaluations in Computer Chess

In many cases, not only a NAG annotation is available, but also a suggested move sequence that should be followed afterwards. This usually means that not the position reached after the move is preferable, but the position at the end of the line. Consequently, in such cases we also considered following these variations until the end of the line. This is implemented as part of the transition function δ.

Chess engines are usually only evaluating quiet positions, where no capturing move is obviously preferred. For example, when the opponent initiated a piece-trading move sequence, the trade should be completed before evaluating the position. This is why such a setup has been suggested in previous works on reinforcement learning in chess (Beal and Smith 1997; Baxter et al. 1998). Hence, we further extended the transition function so that it may also perform a quiescence search, that only returns quite positions.

The utility of both, variations and quiescence search, are evaluated later in this chapter (Section V.5.2).

V.3.4 SVM-based Ranking

Once we have a set of position preferences of the form $s_{i1} \succ s_{i2}$, we can use them for learning a linear utility function, as described in Section III.3.2.c. We use a *support vector machine* (SVM) that can find a hyperplane that maximizes the utility margin, i.e., a hyperplane that maximizes the difference $d(\boldsymbol{\theta}, \zeta_i) = \boldsymbol{\theta}^T(\boldsymbol{\phi}(s_{i1}) - \boldsymbol{\phi}(s_{i2}))$ between the pairs that are closest to the decision boundary. For a description how this can be framed as an optimization problem and how these problems can be solved efficiently we refer to machine learning textbooks such as Schölkopf and Smola (2001). An efficient implementation of SVMs for ranking tasks is described in Joachims (2002). The basic idea is to formulate the constraints via the hinge-loss as explained in Section III.3.2.c. The exact problem solved by an according SVM is a regularized variant of the hinge-loss problem,

$$\begin{aligned} \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^k} \frac{1}{2}\|\boldsymbol{\theta}\|_{v_1} + C\sum_i \|\xi_i\|_{v_2}, \\ \text{s.t. } \boldsymbol{\theta}^T\left(\boldsymbol{\phi}(s_{i1}) - \boldsymbol{\phi}(s_{i2})\right) \geq 1 - \xi_i, \\ \xi_i \geq 0, \end{aligned} \tag{V.3.1}$$

with v_1 as the regularizer norm and v_2 as the loss norm. Commonly used is the squared l_2 norm for the regularizer and the l_1 norm for the loss. ξ_i is the hinge to be minimize and C a tradeoff parameter between the regularizer and the loss function. The resulting parameter vector $\boldsymbol{\theta}$ is then use to define the utility or evaluation function $U(s) = \boldsymbol{\theta}^T\boldsymbol{\phi}(s)$.

We selected SVMs for our experiments because they learn a linear evaluation function, which can be easily plugged into the chess program. It is also important for a good performance of the chess program that evaluations are fast because the more positions it can evaluate the deeper it can search. The preferences are based on the long-term expectation of a state, under an optimal policy. Hence, the obtained utility function defines a value-based utility, as explained in Section III.3.3.a. Therefore, it would be sufficient to choose the move maximizing the expectation of the next states utility (III.3.15) but this does not consider the opponents move and s/he may choose an adversarial action. It is better to use a transition function $\tilde{\delta}(s'' \mid s, a) = \delta(s' \mid s, a)\delta(s'' \mid s', \pi_O(s'))$ with $\pi_O(s)$ as the opponent's policy. $\pi_O(s)$ is un-

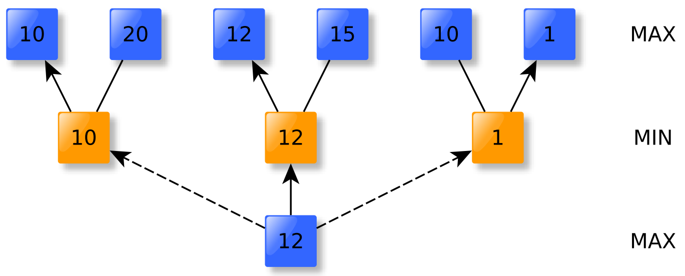


Figure V.3.2.: A minimax game tree with heuristic leaf evaluation with blue as the MAX player and orange as the MIN player.

known, but $\delta(s'' \mid s', \pi_O(s'))$ is deterministic and $A(s)$ is finite, therefore all possibly following states can be enumerated. As it is safe to assume that the opponent will select an adversarial move that minimizes our long-term expectation, i.e., utility value, we can approximate $\pi_O(s')$ for positions where we can evaluate all possible child positions. Hence, chess engines build a tree, starting at the current position and play the move that maximizes the utility of the leaf nodes. Figure V.3.2 shows an example with blue as the nodes where we can act. The numbers define the heuristic value that can be obtained in the leaf nodes. As we know that the opponent (orange) will try to minimize our utility, we can predict the move s/he is going to play and therefore also the best utility we can obtain in adversarial nodes. Hence, 12 is the best value we can obtain, because the opponent will prevent us from reaching the node with value 20.

The deeper the tree, the more opponent moves can be taken into consideration. Finally, the default evaluation function of the chess program is also linear, so that it should, in principle, be possible to achieve a good performance with learned linear functions. Nevertheless, we note that non-linear evaluation functions might achieve a better performance. Our preliminary experiments in Wirth and Fürnkranz (2012) have shown that they can outperform their linear counter-parts, at least in the fraction of correctly predicted preference constraints on unseen positions. We have, however, not tested these functions within the chess program, because the computational overhead is substantial.

V.3.5 Related Work

The basic idea of learning from annotated games was first introduced by Gomboc et al. (2004), but using a hill-climbing approach for optimizing Kendall's τ measure. Additionally, this procedure was only applied to a 11-feature subset of an evaluation function. Furthermore, our work also differs from this publication by trying to analyze and overcome some encountered problems, as well as giving insight into the effect of using more complex evaluation functions.

Our approach to using a preference-based object ranking approach for learning an evaluation function essentially follows Paulsen and Fürnkranz (2010), where a similar algorithm was applied to the problem of learning evaluation functions of different playing strengths. In that case, the training preferences were obtained by assuming that the move actually played by a player had been preferred over all other moves that had not been played.

Interestingly, the idea of training game evaluation functions based on position comparisons actually predates the field of preference learning: for early versions of the TD-Gammon Backgammon program, Tesauro (1988) proposed *comparison training* as a training procedure for evaluation function learning. He demonstrated a particularly interesting technique, where he enforced a symmetric neural network architecture, which could be de-coupled after training, so that the two identical halves can be used as a position evaluator. Utgoff and Clouse (1991) compared this approach to reinforcement learning, which became the predominant algorithm in TD-Gammon (Tesauro 1995, 2002). Later, however, variants of comparison training have also been used for training the evaluation function of the Deep Blue chess program (Tesauro 2001; Campbell et al. 2002), with the goal of learning from grandmaster moves.

Of course, approaches to learning from (expert) demonstration, most notably *inverse reinforcement learning* (IRL), are also applicable to learning from chess databases, but our focus is on learning from annotations. IRL assumes (near) perfect demonstrations, which is hardly available in many domains, as opposed to the partial, qualitative evaluations that can be found in the game annotations.

V.4 Experimental Setup

To investigate the usefulness of preference data, we train chess evaluation functions based on preferences generated from annotated chess games (Section V.4.1), and employ them in the strong open source chess engine Cuckoo (Section V.4.2). All states are represented by the heuristic features created by the position evaluation function. Training a linear model allows us to simply extract the feature weights for the linear sum function. The quality of the preferences can then be analyzed by comparing the playing strength of our re-weighted chess engine. In this section, we describe the basic experimental setup, the results of the experiments will be describe in the subsequent Section V.5.

V.4.1 Chess Database

As a data source we use the *Mega Database 2012*, which is available from *Chessbase*.[38] To the authors' knowledge, this is the largest database of professionally annotated chess games available. The annotations are commonly, but not exclusively, provided by chess grandmasters.

In the more than 5 million games contained in the database, we found 86,767 annotated games,[39] from which we computed 5.05 million position preferences and 190,000 move preferences using the algorithms sketched in Section V.3. Because of the high number of position preferences, we first only considered move preferences, and later investigated the utility of adding position preferences.

V.4.2 Cuckoo Chess Engine

We use the Cuckoo chess engine[40] for our experiments, because of its combination of high playing strength and good modifiability. It facilitates bit boards (Samuel 1959; Adelson-Velskii et al. 1970) for position representations and NegaScout (Reinefeld 1983) as search algorithm.

Most state-of-the-art chess engines use a heuristic position evaluation function while searching for the best, currently reachable position with enhanced Alpha-Beta search algorithms like NegaScout. For performance reasons, evaluation functions are commonly linear sums over abstract, manually constructed features as explained in Section V.3.4. The features are elementary properties of the reached position, such as material difference or usefulness of pieces on their current squares.

Cuckooo's evaluation function may be viewed at different levels of abstraction. Table V.1 shows its 16 top-level features. These features are already aggregated versions of more complex evaluation functions, which are not directly available, but hidden within the code of the evaluator. We also created a complex evaluation function based on this hidden code that uses the 650 features shown in Table V.2. Some values, such as *king safety* or *threat bonus* in the complex function, are multiplicative, i.e., their value is multiplied with another value. Other values depend on various values of the chess pieces. Hence, we did not learn these basic values, but instead used the default basic piece values of the Cuckoo chess engine.[41] The values of kings, pawns and knights squares are interpolated between two different values, one for the mid- and the endgame table, depending on the current material value in play. Moreover, the original Cuckoo engine also uses a separate endgame evaluation and a castle bonus, but these features

[38] `http://www.chessbase.com/`

[39] In addition to the 66k games with textual annotations, there are a few more that only contain a few annotation symbols, which we included, but which are not officially counted as "annotated".

[40] `http://web.comhem.se/petero2home/javachess/`

[41] Previous work has shown that piece values can be satisfactorily learned by self-play in various chess variants (Baxter et al. 2000; Beal and Smith 2000, 2001; Droste and Fürnkranz 2008).

Feature Type	# Features	Description
material difference	1	Difference in the sum of all piece values per player.
piece square	6	Position dependent piece values by static piece/square tables. A single value for every piece type.
pawn bonus	1	Bonus for pawns that have passed the enemy pawn line , while also considering its distance to the enemy king.
trade bonus	2	Bonus for following the "when ahead trade pieces, when behind trade pawns" rules.
castle bonus	1	Evaluates the castling possibility.
rook bonus	1	Bonus for rooks on (half-) open files.
bishops scores	2	Evaluating the bishops position by attack possibilities, if trapped and relative positioning.
threat bonus	1	Difference in the sum of all piece values under attack.
king safety	1	Evaluates the kings position relative to the rooks.

Table V.1.: Aggregated features used in the linear evaluation function of the Cuckoo chess engine.

can also not be implemented in a linear evaluation function, so that we had to omit them. Finally, it was also not possible to correctly extract the function used within the threat bonus and the king safety feature, therefore the directly available aggregated version was used.

The Cuckoo chess engine was used in a single-thread configuration. All experiments haven been executed on systems with two cores or more, ensuring independence of the available computing power for each player.

V.4.3 Learning Algorithm

There are several publicly available implementations of SVMs available, such as SVMlight[42] or LIBLINEAR.[43] We started our experiments with the former because it has a particularly fast implementation (SVMRank; Joachims 2002) tailored to ranking problems. However, later on we switched to LIBLINEAR (Fan et al. 2008) because of stability issues.

For training an evaluation function with LIBLINEAR we generate position preferences as described in Section V.3, resulting in a set of preferences ζ. Each preference difference ζ_i in this set is described with the difference of the feature vectors of the underlying position, i.e., $\boldsymbol{\phi}(\zeta_i) = \boldsymbol{\phi}(s_{i1}) - \boldsymbol{\phi}(s_{i2})$. The features have not been standardized or normalized, because they are already internally normalized to a pico-pawn scale.[44] After learning, as described in

[42] available from `http://svmlight.joachims.org/`

[43] available from `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`

[44] Chess programs typically normalize their feature values to the value of a pawn or, more commonly, to the value of a pico-pawn (one percent of a pawn).

Feature Type	# Features	Description
material difference	1	Multiplier for the material difference.
kings square table midgame	64	Position of the king in the midgame.
kings square table endgame	64	Position of the king in the endgame.
pawns square table midgame	64	Position of the pawns in the midgame.
pawns square table endgame	64	Position of the pawns in the endgame.
knights square table midgame	64	Position of the knights in the midgame.
knights square table endgame	64	Position of the knights in the endgame.
bishops square table	64	Position of the bishop.
queens square table	64	Position of the queen.
rooks square table	64	Position of the rook, multiplied with the pawn count.
queens mobility	28	Positions the queen can reach.
rooks mobility	15	Positions the rooks can reach.
bishops mobility	14	Positions the bishops can reach.
pawn bonuses	5	Values for double, island, isolated, backward and passed pawns.
trade bonuses	2	Bonus for following the ”if ahead, trade pieces, if behind, trade pawns rule”. Multiplied by pawn count and material value.
rook bonuses	3	Bonus for rooks on open or half-open files and the 2th/7th row.
bishop bonuses	2	Bonus for a pair of bishops as fixed value and pawn based multiplier.
trapped bishop penalties	2	Penalty for bishops blocked by enemy pawns.
threat bonus	1	Multiplier for the sum of pieces under attack.
king safety	1	Multiplier for the king safety value.

Table V.2.: Full feature set used in the linear evaluation function of the Cuckoo chess engine.

Section V.3.4, the feature weights $\boldsymbol{\theta}$ can be extracted from the learned linear function model, and can be subsequently passed to the Cuckoo chess engine.

Considering the example given in Section V.2, we would have to acquire constraints for all pairs in the given order 13...f×e5 $\succ$ 13...a5?! $\succ$ 13...♛×c2? $\succ$ 13...♘×c2?? . For example, the two top-ranked elements would relate to s_{i2} = Move(13...f×e5) and s_{i2} = Move(13...a5) as the positions reached by applying the moves in question. The training example is now defined by the difference between the state features $\boldsymbol{\phi}(s_{i1})$ and $\boldsymbol{\phi}(s_{i2})$ (Tables V.1 & V.2) and the corresponding label $r_i = \{>\}$.

V.4.4 Evaluation

The learned utility functions have been evaluated in several round robin tournaments, where each learned function plays multiple games against all other functions. Unless mentioned otherwise, all tournaments are played using a time control of two seconds per move.

All results are reported in terms of Elo ratings (Elo 1978), which is the commonly used rating system for rating chess players. It not only considers the absolute percentage of won games, but also takes the strength of the opponent into account. A rating difference of a 100 points approximately means that the stronger player has an expected win rate of $5/8$. It also allows to report upper and lower bounds for the playing strength of each player. For calculating the Elo values, a base value of 2600 was used, because this was the rating for Cuckoo as reported by the *Computer Chess Rating List*[45] in 2012. In 2017, the engine was rated with an Elo of 2589. It should be noted that computer engine Elo ratings are not directly comparable to human Elo ratings, because they are typically estimated on independent player pools, and thus only reflect relative strengths within each pool.

V.5 Results

In this section, we report the results of our study. Section V.5.1 summarizes the results of a preliminary study which only used move preferences and the small abstracted feature set in order to keep the learning complexity low (Wirth and Fürnkranz 2012). Building upon these results, we expand and improve the experimental procedure in several ways. In particular, we add position preferences, optimize various parameters of the experimental procedure, and report results on learning both the abstract and the complex linear evaluation functions (Section V.5.2). Upon analyzing these results, we noticed that a possible reason for the weak performance could be that annotated positions are not representative for all positions that may be encountered during this game. We investigate this hypothesis in Section V.5.3, where we show how the results can be improved by adding positions where one side has a very clear, decisive advantage. Finally, we compare the simple and the complex evaluation functions in Section V.5.4.

V.5.1 Learning from Move Preferences

In a first feasibility study, we only used action preferences and the 16 aggregated features for evaluation (Table V.1). We used all positions immediately after an annotated move, i.e., we did not follow the annotator's variations that start with this move. However, in line with previous work on learning evaluation functions, we applied quiescence search (cf. Section V.3.3). We

[45] `http://www.computerchess.org.uk/ccrl/4040/`

briefly summarize the key results of this study, for details we refer to Wirth and Fürnkranz (2012).

We learned six different evaluation functions using 5%, 10%, 25%, 50%, 75% and 100% randomly sampled games of the available move preferences. We sampled three different sets for each player (except for the 100% engine, of course), and learned one evaluation function for each training set. The learned functions were evaluated in three round robin tournaments, each including the player with the original feature weighting and a random player as baseline algorithms. The random engine picks a new set of random weights for each position evaluation. The weights are sampled uniformly from the min/max range of values observed within all learned SVMRank models. Each pairing played 100 games with a 5min time frame and no increments.

Figure V.5.3 shows the development of the rating over the percentage of used move preferences in the training data. It is recognizable that an increase in the number of used preference data leads to an improved chess engine, which we take as evidence that the game annotations provide useful information for learning an evaluation function. The playing strength is clearly above the random baseline, which reached an average Elo rating of 2332 ± 32, but, on the other hand, also well below the original player and its average Elo rating of 2966 ± 43.

The player that was trained on 5% of the data is an outlier, resulting from the comparably high variance in the training data at this point. This is illustrated in Figure V.5.4 which shows the mean and variance for the five features with the highest variation in the learned values.

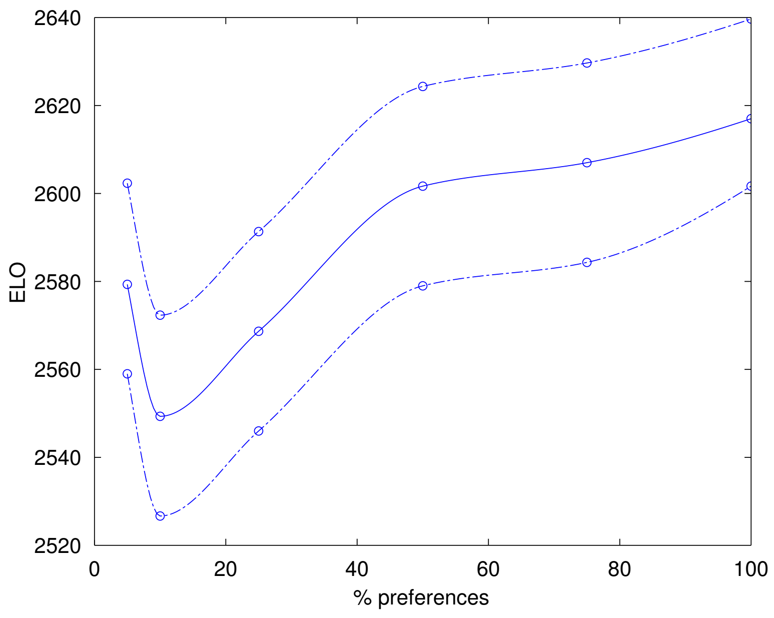


Figure V.5.3.: Playing strength in Elo rating (solid line) with lower and upper bounds (dashed lines) over percentage of move preferences used for training.

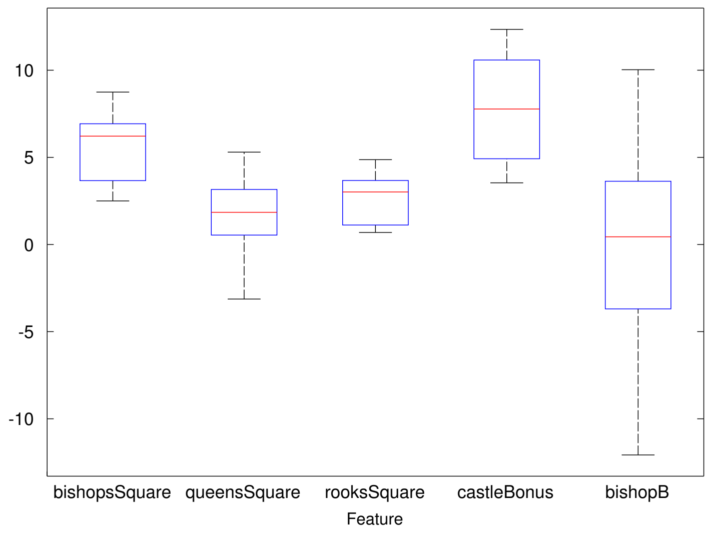
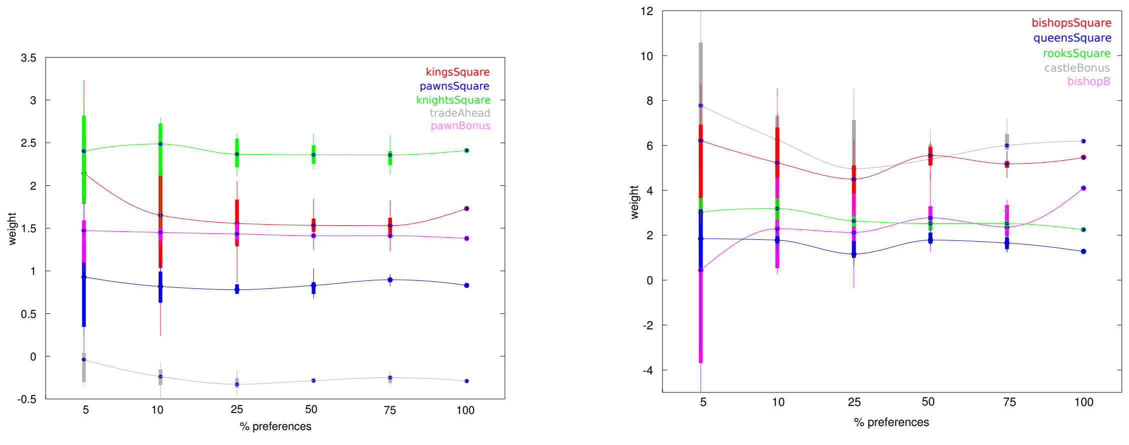


Figure V.5.4.: Variance of the learned weight for the 5 most variant features, based on 10 different random samples, each using 5% of the available data.



(a) Feature values with stable convergence (b) Feature values without stable convergence

Figure V.5.5.: Development of average and variance for selected feature weights, averaged over 10 samples per subset size. Weights are scaled to *materialDifference*= 1.

However, most features converge towards a stable average value. Figure V.5.5 shows the development of the average, standard deviation and min/max for selected features. The 5 features in Figure V.5.5a are quite stable, showing mostly stable average values, whereas the features in V.5.5b are rather unstable. The features *castleBonus* and *bishopB* only rarely have an effect, their feature value difference is 0 in 84.6% and 99.7%, respectively, of all training examples.

This could be a reason for their unstable development. The six features not shown were also stable (Wirth and Fürnkranz 2012).

V.5.2 Learning from Position Preferences

As we already noted in Section V.4.1, the database contains about an order of magnitude fewer move preferences than position preferences. Thus, it is reasonable to expect that if we move from move preferences to the much larger set of position preferences, the results reported in the previous section will further improve.

To test this, we generated both move and position preferences for training set sizes of 10%, 25%, 50%, and 100% of all games. We divided the data into k non-overlapping folds, each holding $\frac{1}{k}$-th of the data (e.g., 4 folds with 25%), and trained an evaluation function for each fold. The players that represent different values of k then, before each game, randomly selected one of the evaluation functions for their size. Thus, unlike the previous setup where separate tournaments were played and averaged after, here we played a single tournament where the results of different evaluation functions were implicitly averaged through the random selection in each game. Furthermore, we changed the tournaments timing setup to 2 sec/turn to be independent from the problem of distributing computation time over multiple moves. The second and third column of Table V.3 show the number of preferences in the training data for each configuration, averaged over all folds of the same size (we will return to columns four and five in Section V.5.3).

We first optimized various parameterizations of the experimental setup. In particular, we optimized the C-parameter of the SVM, which regulates the amount of overfitting by specifying a trade-off between an exact fit of the training data and the simplicity of the found weight vector. We tested three values, $C = 1$, $C = 0.01$ and $C = .00001$, on each of the four training set sizes, resulting in 12 different configurations.

Our second objective was to compare different ways of incorporating position evaluations. As described in Section V.3.3, one can either directly use the position to which the annotation is attached, attach the annotation to the end of the line provided by the annotator, or perform a

	Preferences			
#games	Move	Position	Augmented	Surrender
10%	0.02M	0.50M	0.34M	0.05M
25%	0.04M	1.26M	0.85M	0.12M
50%	0.09M	2.53M	1.71M	0.24M
100%	0.19M	5.05M	3.41M	0.48M

Table V.3.: Average Number of Generated Preferences (in Millions).

Follow Variations	Quiescence Search	Simple Features	Complex Features
×	—	1.0	1.7
—	—	2.5	1.8
×	×	2.5	2.5
—	×	4.0	4.0

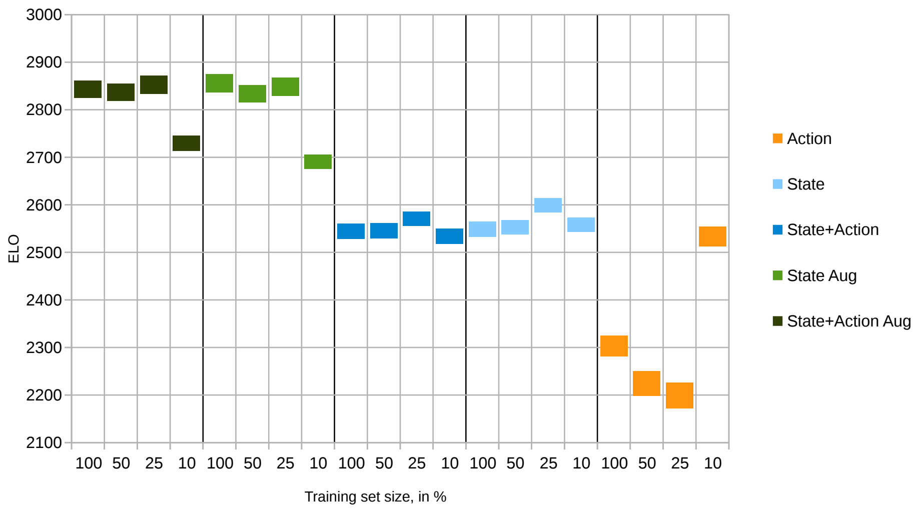
Table V.4.: Average rank of different variants of computing position preferences over 12 tournaments with 3 different SVM C values and 4 subset sizes

quiescence search as is commonly done in reinforcement learning of evaluation functions. We compared all four possible combinations of these options by conducting tournaments between them in each of the 12 settings described above. Each of the six pairings in each of the 12 tournaments was decided by playing 75 games. We repeated this for learning both, a simple evaluation function (using the features of Table V.1), and a complex evaluation function (Table V.2).

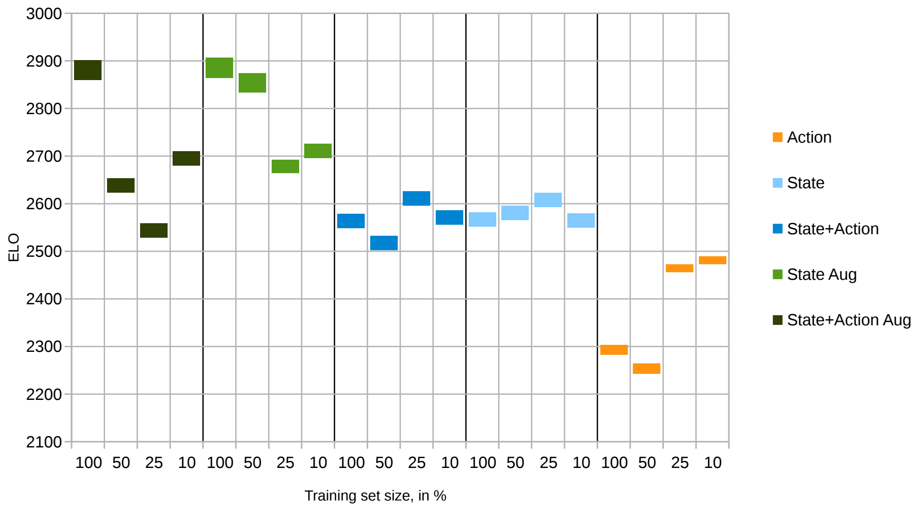
Table V.4 shows the average rank of each of the four possible combinations of using quiescence search or not and following side lines or not across these 12 tournaments. It turns out that following the annotated side lines, but without using quiescence search performs best in every instance using the simple heuristic, and is the best setting on average for the complex heuristic. Hence, we decided to run all subsequent experiments with this setup. Interestingly, the setup that we had selected in our first experiments (Section V.5.1) consistently turned out to be the worst setting.

For determining the best parameter settings of C for each subset size we let the highest ranked player of each of the 12 tournaments compete in a single tournament with 200 games. For each subset size, we selected the C-parameter value that performed best in this tournament and used it for all subsequent experiments. Again, this optimization was performed separately for the simple and the complex evaluation functions. The found optimal values differ a lot and do not exhibit a noticeable pattern. However, it should be noted that for the complex evaluation function, the difference in playing strength between two parameterizations can be quite large. We will return to this issue further below.

These optimal parameter settings were then used in a large tournament that compares the performance of different types of preference information. Figure V.5.6 shows the results. For the moment, only consider the right-most three groups in each graph, namely those representing learning from position preferences, learning from move preferences and learning from both. Clearly, learning from only move preferences yields much worse results that learning from position preferences. Moreover, adding move preferences to the position preferences does not further increase the performance.



(a) Results for the simple evaluation function



(b) Results for the complex evaluation function

Figure V.5.6.: Comparison of players learned from different types of preference information for four training set sizes with the square height as confidence interval.

Clearly, one has to keep in mind that the number of move preferences is considerably smaller than the number of position preferences, so that the values are already different with respect to training set sizes. However, we also see from the results that learning from position preferences is already saturated in the sense that there are only minor performance variations between the different training set sizes, i.e., more position preferences do not further improve performance. The same seems to be the case when adding move preferences to the position preferences. Thus, we are inclined to conclude that the move preferences do not add qualitatively different information but just add more of the same information.

V.5.3 Augmenting Position Preferences

While we have seen that learning from the much larger number of position preferences yields considerably better results than our first experiments that only used move preferences, we also still noticed a considerable performance gap between the performance of the original Cuckoo chess program and the one using our learned evaluation function. In particular, we noticed that fundamental concepts like the material value of the different pieces tends to be underestimated by the learned heuristics.

A possible reason for this could be that human players tend to only annotate “interesting” positions, i.e., positions where the evaluation is not entirely clear. Thus, the program may miss to pick up that, e.g., it is in general really bad to be a queen down because such positions are too clear to be annotated. Quite in contrary, the learner may see several examples where the side that is a queen down may nevertheless win the game because positions that involve a queen sacrifice are typically interesting.

In order to test this hypothesis, we tried to augment our preference data with preferences that involve a very clear material advantage for one side. We focused on positions that are already annotated as a decisive advantage for white (+−) or black (−+). For each such position, we generated a new position by removing a random piece, excluding pawns and kings, of the inferior side. The idea is that if we remove a piece from an already lost position, the resulting position should be extremely bad. Thus, such positions are hopefully more extreme than those that are usually annotated in game databases. This technique increased the number of state preferences by 60%-70%, as can be seen from the fourth column of Table V.3.

We compared the augmented preference sets to non-augmented sets in a tournament with 100 games per pairing. Figure V.5.6 shows the results. We can clearly see that in the case of the simple evaluation function (Figure V.5.6a), the augmented preference sets always outperform all other settings. We can also observe a similar saturation as with position preferences, but here it only occurs after 25% of the preferences have been seen.

The results for the complex evaluation function (Figure V.5.6b) shows essentially the same results, but in a much more unstable manner. Clearly, tuning a function with 650 features is much more difficult than tuning only 16 features because of more intricate dependencies between the individual features. Thus, the results between different runs also vary to a much larger extent than for the simple function. This can also be seen from the fact that in this case, the learning curve does not saturate at 25%, as for the simple heuristic, but it keeps improving with more preferences. Moreover, several of the learned weights for this function are 0.

	Mtrl Diff	Piece Squares				
		King	Pawn	Bishop	Queen	Rook
Original	1	1	1	1	1	1
Base 100	0.13	0.86	0.90	1.87	1.35	1.21
Base 50	0.13	0.87	0.90	1.88	1.34	1.19
Augm. 100	0.34	0.78	0.82	1.89	0.55	1.39
Augm. 50	0.34	0.79	0.81	1.88	0.54	1.35
	Bonuses					
	Pawn	Trade1	Bishop1	Bishop2	Threat	Safety
Original	1	1	1	1	1	1
Base 100	0.61	0.29	0.44	0.80	2.84	0.76
Base 50	0.62	0.31	0.44	0.74	2.87	0.77
Augm. 100	0.56	0.83	0.66	0.72	2.65	0.88
Augm. 50	0.57	0.85	0.66	0.71	2.68	0.88

Table V.5.: Normalized values of the learned feature weights for various configurations.

In order to see whether the augmentations really helped, we took a closer look at the features learned for the simple heuristic. Table V.5 shows the feature values (normalized to $||\boldsymbol{\theta}||_1 = 16$) of the original player along with the feature values of the players with position preferences and augmented position preferences using 50% and 100% of their preferences. All features are shown except four features that have been within $+-0.1$ of the original weight. We can see that the material difference is grossly underestimated as mentioned above. On the other hand, we can also see that the threat feature is strongly overestimated. Both can be explained by the above-mentioned expert's selection bias, because interesting and therefore annotated positions are usually those where one player is not obviously dominating by material difference, but where on the other hand a lot of tension is on the board, i.e., where several pieces are threatened.

Our augmentation procedure was able to counter this bias to some extent. In particular, the value for material difference has increased considerably. However, we also observe a few strange effects in the piece-square tables. In particular, the average value of the queen in these tables has changed from an over-estimation to a strong under-estimation. This is probably somewhat counter-balanced by the increased importance of the material balance, but the complex interactions between these features are hard to interpret. Piece square features strongly depend on realistic occurrence counts for *each* square, which can not be achieved by simply removing pieces.

Interestingly, we observe that the differences between using all or only half of the available preferences is only marginal in both settings, so that we can conclude that the learned results are stable.

Finally, we also tried to enhance the number of preferences even further by including positions where a player resigned the game. Most decisive chess games do not end with checkmate, but one of the two players resigns the game, thereby yielding the win to the other side. Clearly, such an action also indicates a strong preference for the winning side. We inserted such final position at the top (when black to move resigned) or bottom (when white to move resigned) of the preference order $\sqsupset$, with the reasoning that these positions are so hopeless that their qualitative evaluation should be worse than $-+$ (or better than $+-$ respectively).[46]

Including such surrenders increased the number of available position preferences by ca. 5% (cf. column five in Table V.3). However, these additions did not further increase the performance. In fact, the Elo rank worsened slightly in all cases, but not in a significant way (considering the Elo bounds). Explaining these results requires further evaluation, but it supports our theory that selecting the *right* preferences is quite important, pointing out one of the problems that have to be considered when dealing with human preferences.

V.5.4 Simple and Complex Heuristic

Finally, we wanted to compare the best configurations of learning the simple vs. the complex evaluation functions with the original player. Thus, we played a final tournament, also with 100 games per pairing, where we included the players based on Move+Position+Augmented preferences for both evaluation functions and the original player.

Figure V.5.7 shows the results. Essentially, we can see that the simple evaluation function outperforms the complex variant. While this may sound surprising, keep in mind that the evaluation function is not simpler in the sense that it includes less chess knowledge, but it is only simpler in the sense that less parameters have to be tuned because they only apply to pre-aggregated features. It seems to be the case that the available training information is sufficient for tuning the small set of parameters (note that the performance does not strongly improve after seeing 25% of the data), but it may not have been enough for tuning the large set parameters of the complex evaluation function. Note that in this case, the learning curve is steeply increasing from 25% to 50% and 100% of the data (we again note an outlier for 10% of the training data).

However, finally we have to note that even the best performing variant did not quite reach the performance of the hand-tuned version, although the gap has decreased in comparison to our preliminary study (cf. Section V.5.1). We have already speculated about possible reasons for this failure. In particular, as we have seen in the greatly improved performance that could be obtained by augmenting the position preferences with preferences that involved positions with strong material imbalance, it seems to be a problem that only certain types of chess positions

[46] We used a few crude heuristics for excluding cases where a game was decided by time controls or similar.

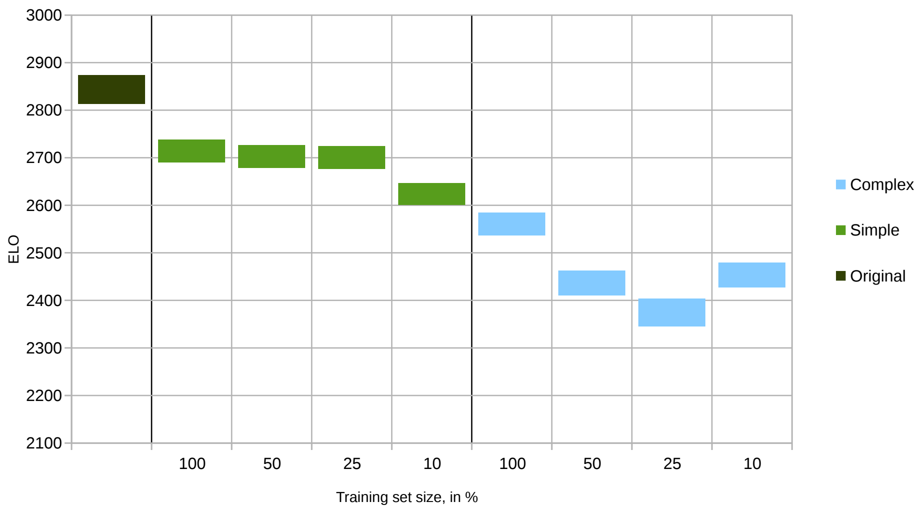


Figure V.5.7.: Comparison of the simple and complex heuristic with the original player with the square height as confidence interval.

are annotated, so that we cannot obtain that same quality of results that have been observed by reinforcement learning from self-play.

V.6 Problems with Human Preferences

We have been able to confirm that useful information can be learned from annotations, and that not all weights are converging to their expected optima. We also showed that this is not a problem of insufficient training data for the simple evaluation heuristic. One problem that we identified is that human annotators tend to focus on close and interesting positions. We could show that an augmentation of the training information with artificial preferences that result from generating bad positions by removing pieces from already losing positions leads to a significant increase in the quality of the learned function, confirmed by an evaluation of the resulting feature weights. One could also speculate whether the observed performance differences are the result of different scales used by different annotators. However, we did not generate preferences across different games, only within individual games. Thus, we cannot suffer from this problem as long as each expert is consistent with herself.

It is still unclear why there is a difference between the hand-tuned and the learned evaluation function and why the augmentation did improve the results. Several reasons are possible. As we have been able to confirm, that the amount of training data is sufficient, we can assume that the data is either too noisy or that important information is missing. By augmenting the training

	$+-$	$\pm$	$\pm$ with line below	$\mp$ with line above	$\mp$	$-+$
$+-$		68.55%	74.51%	76.52%	78.23%	82.81%
$\pm$			61.08%	68.53%	73.25%	79.52%
$\pm$ with line below				60.59%	70.99%	79.66%
$\mp$ with line above					61.16%	75.27%
$\mp$						69.27%
$-+$						

Table V.6.: Agreement between preferences obtained from the database and the handcrafted evaluation function. Each entry relates to all preferences obtained by comparing positions with the annotation symbols shown in the row and column headers.

data, we added additional information for training that was not available before. Hence, it is possible that we added some of the missing information. However, the newly generated positions can be assumed to be nearly noise free, as extreme position differences increased further, the augmentation process did also decrease the relative number of noisy preferences. Hence, we can not be sure which of the two effects are the reason for the observed improvement.

V.6.1 Noise in Human Preferences

To answer the remaining questions, we analyzed the obtained preferences and compared them with the preferences induced by the handcrafted evaluation function. Table V.6 shows the amount of identical preferences, comparing all preferences obtained from the human annotations with preferences obtained by applying the hand-tuned evaluation function to the same positions. Each field relates to all preferences obtained by comparing positions annotated with the symbols stated in the according row/column header. It can be assumed that there is a noise problem as there is a significant difference in agreement. Furthermore, the agreement increases with difference in the human evaluation, e.g., human preferences comparing decisive positions ($+-$ vs. $-+$) are more likely to be in-line with the original evaluation function than preference between closely related symbols. Hence, this also explains the improvement in playing strength by augmenting the data with new preferences derived from extreme positions. Human preferences involving extreme positions are more likely to be in-line with the evaluation function. This assumptions is also confirmed by the fact that the average agreement of preferences did rise from 70% to 82%, when adding the augmented preferences.

V.6.2 Sigmoidal Loss Function

The preference-based hinge loss only considers a linear tradeoff between violated preferences, but other loss functions better suited to deal with noise. A common assumption is that the probability of correctness of a preference scales with the utility difference (Wilson et al. 2012; Akrour et al. 2014). For example, preferences for trajectories with similar (true) utilities are more likely to be wrong. As shown by Table V.6, this assumptions holds in the chess domain. Hence, sigmoidal likelihood functions, as introduced in Section III.3.2.c, are better suited to capture the noise introduced by human annotations. The probability of correctness is 50% if the utility difference is 0 for the preference pair and increases with higher difference but levels out. In case of negative difference values, the probability of correctness approaches 0. In contrast to stepwise loss functions, it is possible to minimize the sigmoid loss efficiently using convex optimization algorithms.

V.6.3 Further Results

To evaluate the sigmoid loss function, we compared the original heuristic with the a heuristic learned by the hinge loss and the sigmoid loss, based on the simple feature space with 16 dimensions. The training data contained all available preferences, the 100% set. As we have observed that the regularizer does not influence the results for this set, we used an unregularized optimization problem for this new set of experiments. This allowed us apply a commercial solver for linear programs,[47] resulting in a significant speedup when calculating the hinge loss. We used this speedup for reducing the error tolerance by several orders of magnitude and obtain more exact results, removing another possible source of error. The new experiments are not directly comparable to the old set, due to slight code changes and more performant hardware, changing the runtimes of the engine. Furthermore, we increased the tournaments to 200 games per pairing as the differences between the heuristics diminished, resulting in a higher variance as before. Figure V.6.8 shows the obtained results. The sigmoid loss function outperforms the hinge loss substantially, but is still not reaching the performance level of the original player. Hence, we can determine that noise-tolerant loss functions are more appropriate than linear losses.

However, we can not rule out the possibility that missing information is also a factor. For evaluating this theory, we created a new training set by removing all preferences where the original evaluation function disagreed with the preferences derived from the annotations and applied the hinge loss as well as the sigmoid loss for training a new heuristic evaluation function. In both cases, *the learning approach managed to recover the original heuristic*, even for a randomly

[47] `http://www.gurobi.com/`

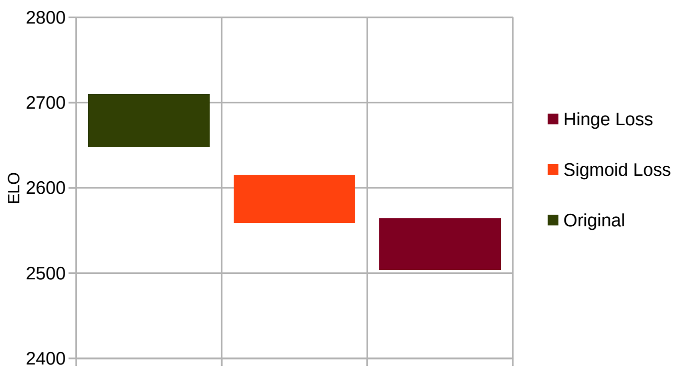


Figure V.6.8.: Comparison between the sigmoid loss and the hinge loss with the square height as confidence interval.

sampled 10% subset. We can therefore conclude that the training data is not missing information for learning a heuristic that is at least as good as the handcrafted version, but that the sigmoid loss is still not capable enough in dealing with noise.

V.7 Conclusion

We demonstrated that useful information can be learned from game annotations, in the domain of chess. The used linear loss function is suited for generalizing preference-based feedback (cf. Research Question 2.A), but we also observed that the performance of the engine does not directly scale with increasing amounts of preferences. Human evaluations introduce several possible problems that can explain this observation. We determined that noise is a relevant problem, answering Research Question 2.B. However, in other domains, the number of preferences or missing information can also be a factor, but we can rule out these factors in the used setup.

Non-linear loss functions, like a sigmoidal loss, show substantial improvements over linear approaches, supporting the theory that improved noise handling is important for dealing with preferences defined by humans, answering Research Question 2.C. Hence, we can add to our answer to Research Question 2.A that noise-tolerant loss functions are more appropriate than linear losses.

Further improvements can be observed when limiting the dimensionality of the utility function space, but this was to be expected as it simplifies the learning problem. However, we also showed that propositions from humans should be considered. Hence, we can determine that future research should consider combinations of multiple feedback types and/or demonstrations.

Chapter VI

Reinforcement Learning with Preference-based Utility Functions[49]

[49] This chapter is based on Wirth, Fürnkranz, and Neumann (2016).

The aim of this chapter is to use the obtained knowledge to create a *preference-based reinforcement learning* (PBRL) algorithm, that is potentially applicable to a wide range of domains and requires a low number of preferences for learning. In Section IV.6.3, we determined that efficient approximations for the temporal credit assignment problem are substantial for speeding up the learning process. Hence, we use the reward-based utility function method (cf. Section III.3.2.c & III.3.3.c) as it allows to generalize the obtained feedback to all state-action pairs. Furthermore, the principle allows us to decouple the *reinforcement learning* (RL) problem from the *preference learning* (PL) problem. For solving the PL problem, we use the non-linear loss function discussed in Section V.6.2, as we have determined its advantage in Section V.6.3. In contrast to the other reward-based utility function approaches (cf. Section III.3.3.c), we aim at a model-free algorithm that is applicable to non-parametric policies. The assumption of model knowledge or parametric policies restrict the applicability of existing algorithms. To reduce the number of required preferences, all obtained preferences should be as informative as possible. The aim is to quickly reduce the uncertainty over the expert's evaluation function ρ, introduced in Section III.1. As we want to obtain an algorithm that is applicable to high-dimensional problems, we can not use directed exploration criteria like Akrour et al. (2014) (cf. Section II.4.1). We need to ensure that the algorithm creates trajectories suitable for informative preference queries using an undirected exploration method. As undirected methods use controlled, random exploration, it is not sensible to query the expert for each possible preference, but we need to decided which trajectories should be used for querying the expert.

The resulting research questions are

3.A How can we incorporate a noise-tolerant loss functions into a reward-based PBRL algorithm that is able to deal with unreliable, human feedback?

3.B How can we solve the PBRL problem efficiently, without depending on a model of the transition dynamics or parametric policies?

3.C How can we use undirected exploration to explore the system dynamics as well as the expert's evaluation function, potentially allowing the algorithm to scale to higher dimensional problems?

3.D Which trajectories should be selected for preference queries to minimize the number of required expert evaluations?

VI.1 Overview

Considering the PBRL learning cycle (Figure III.2.1), we need to solve four subtasks:

1. approximation of the expert's utility function,

2. policy improvement,
3. collection of new trajectories and
4. requesting new preferences.

For learning a model of the expert's utility function, we use a method derived from the ideas of Akrour et al. (2012) and Akrour et al. (2014), because they are among the most efficient PBRL algorithms currently known. Our approach replaces the loss function with a sigmoidal loss and computes the solution to a Bayesian inference problem. The posterior distribution allows to obtain explicit information about the uncertainty of the utility function, that we can use to guide the policy improvement step. The resulting reward-based utility can be directly used within a RL algorithm where the RL algorithm is used to perform a policy improvement step.

For the policy improvement step, we propose a new algorithm called *actor critic relative entropy policy search* (AC-REPS), which allows us to directly control the exploration-exploitation trade-off by bounding the relative entropy between the old and the new policy. Traditional exploration techniques such as softmax or ϵ-greedy action selection can control this trade-off only indirectly and are therefore much harder to tune. Bounding the relative entropy is a well-known strategy in policy search (Peters et al. 2010) for limiting the amount of change, as explained in Section II.4.

Following the new stochastic policy, we can now sample a new set of trajectories that is potentially superior to the current set of undominated trajectories and request new preference.

Details of this procedure and how we realize the elements of this cycle are given in the following sections.

VI.2 Inferring a Reward-based Utility Function from Preferences

The problem of approximating the utility function from preferences is closely related to *inverse reinforcement learning* (IRL). Both settings have access to predefined trajectories. In the IRL case, the trajectories are given by the expert and usually assumed to be near optimal, i.e. the shown trajectories are implicitly preferred over most unseen trajectories, defining implicit preferences. In the PBRL case, the trajectories are generated by the algorithm itself with pairwise preferences requested from a human expert, i.e., both trajectories of a preference pair are explicitly known. Hence, both approaches can be formalized by similar algorithms that use the available preferences as constraints for the utility function that we want to estimate.

VI.2.1 From IRL to PBRL

Ng and Russell (2000) presented the first algorithm for IRL. It is based on the idea that the value of the demonstrated policy V^{π^*} should be higher than for any other policy V^{π_i}. They assume that the reward function $r(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$ is linear in a given feature space. Due to this linearity, the resulting value V^π is also linear in the *feature expectations* $\boldsymbol{\phi}(\pi)$, i.e.,

$$\hat{V}^\pi = \boldsymbol{\theta}^T \boldsymbol{\phi}(\pi), \tag{VI.2.1}$$

where $\boldsymbol{\phi}(\pi)$ is defined as

$$\boldsymbol{\phi}(\pi) = \mathbb{E}_{\pi,\delta}\left[\sum_{t=0}^{|\boldsymbol{\tau}|} \gamma^t \boldsymbol{\phi}(s_t)\right]. \tag{VI.2.2}$$

The action dependence of the reward function is disregarded in this definition, but could be easily added by using $\boldsymbol{\phi}(s_t, a_t)$ instead of $\boldsymbol{\phi}(s_t)$. The constraints $V^{\pi^*} > V^{\pi_i}$ now translate into linear constraints on the parameter vector $\boldsymbol{\theta}$, i.e, $\boldsymbol{\theta}^T \boldsymbol{\phi}(\pi^*) > \boldsymbol{\theta}^T \boldsymbol{\phi}(\pi_i)$.

IRL can be easily extended to the PBRL case. First, we do not compare policies but trajectories, i.e., instead of using feature expectations over policies we use feature averages over trajectories $\boldsymbol{\phi}(\boldsymbol{\tau})$ (III.3.16). Note that feature averages over trajectories do not require knowledge of the system dynamics δ, unlike the feature expectations of IRL.

We also don't have access to optimal trajectories $\boldsymbol{\tau}^{\pi^*}$, but only to pairwise feedback $\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2}$. Therefore, we have to rewrite the constraints as $\boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{\tau}_{i1}) > \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{\tau}_{i2})$. We can now define an optimization problem, based on the mentioned constraints. Because of the binary feedback, we want to minimize the 0-1 loss

$$\min_{\boldsymbol{\theta}} \sum_{i=0}^{|\zeta|} \mathbb{1}(d(\boldsymbol{\theta}, \zeta_i) \leq 0), \tag{VI.2.3}$$

with $d(\boldsymbol{\theta}, \zeta_i)$ as the utility difference function (III.3.11). The result of the optimization is a new realization of $\boldsymbol{\theta}$ that specifies the utility $U(s) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$. We do not use the term reward in the PBRL setting, as explained in Section III.3.2.c.

However, this formulation is subject to three problems: (i) it should be possible to approximate the utility with a linear function, (ii) the value difference of a preference can become arbitrary small, and (iii) the problem can have multiple solutions. For overcoming the first problem, we utilize a tabular model for the feature space in discrete domains. In this case, the linear function is not an approximation, but an exact representation (Geramifard et al. 2013; cf. Section II.3.1). In continuous domains, we use a squared exponential kernel (II.3.28) for defining our feature space.

To ensure a selection of *basis functions* that is able to capture the properties of the sampled parts of the feature space but stays tractable, we use a subset of observed state-action samples as centers. Hence, we can assumed good approximation capabilities, but not ensure optimality. Problem two and three can be overcome by suitable loss functions, as explained in the next section.

VI.2.2 Bayesian Estimation of the Utility Function

As mentioned, minimizing the 0-1 loss is not sufficient for determining a solution to the preference problem. To prevent arbitrary small differences and determine a single, best solution, we follow the method of Ng and Russell (2000), also utilized in Akrour et al. (2012) and Akrour et al. (2014) and most other PBRL approaches. Our loss function is not a indicator method but continuous and the loss value scales with the utility difference. Furthermore, the loss saturates with the utility difference, as mentioned in Section III.3.2.c. The loss is rephrased as a likelihood function, allowing the computation of a posterior distribution, given a suitable prior. Obtaining a posterior distribution has the advantage of encapsulating information about the uncertainty of the utility function, allowing to guide the policy optimization and preference query selection process accordingly.

We consider the problem of computing the weight vector $\boldsymbol{\theta}$ via the given preferences ζ, which can be solved via Bayesian inference (III.3.13), as introduced in Section III.3.2.c. For $\boldsymbol{\theta}$, we utilize a multivariate, Gaussian prior $\mathcal{N}(\boldsymbol{\theta} \mid \mathbf{0}, \sigma^2 \boldsymbol{I})$. To reduce the number of free parameters, we fixed the covariance matrix to a uniformly scaled identity matrix. Our likelihood function

$$p_{\boldsymbol{\theta}}(\zeta_i) = \frac{1}{1 + \exp(-m \cdot d(\boldsymbol{\theta}, \zeta_i))}, \tag{VI.2.4}$$

is a sigmoid function, as shown in Figure III.3.4. The shape factor m defines the steepness of the sigmoid function. The sigmoid function is well suited for noisy preferences, as explained in Section V.6.2. However, we may not encounter noisy preferences in a given setting, it is still relevant to consider trade-offs between multiple preferences as we may not be able to obtain the true, optimal utility function with the chosen representation (cf. Section VI.2.1). Hence, it is possibly not able to satisfy all preference with a given representation. The proposed formulation is difficult to solve analytically, because we want to access the complete posterior distribution, not only the mean or maximum likelihood. However, it is possible to employ *elliptical slice sampling* (ESS; Murray et al. 2010).[50] ESS is a *Markov chain Monte Carlo* (MCMC) strategy that allows sampling from a posterior distribution, given a Gaussian prior and an arbitrary likelihood function.

[50] http://homepages.inf.ed.ac.uk/imurray2/pub/10ess/

We can now guide the policy into areas of the state-action space with promising utility values by employing the idea of *upper confidence bounds* (UCB; Auer et al. 2002). As we are interested in continuous action spaces, we can not use the (action) visit counts for computing the confidence interval, but use the standard deviation, for obtaining our utility function

$$\begin{aligned} U(s) &= \mu_{\boldsymbol{\theta}}^T \boldsymbol{\phi}(s) + c\sqrt{\mathbb{E}_{\Pr(\boldsymbol{\theta}\,|\,\zeta)}\left[\left((\boldsymbol{\theta} - \mu_{\boldsymbol{\theta}})^T \boldsymbol{\phi}(s)\right)^2\right]}, \\ \mu_{\boldsymbol{\theta}} &= \mathbb{E}_{\Pr(\boldsymbol{\theta}\,|\,\zeta)}\left[\boldsymbol{\theta}\right]. \end{aligned} \tag{VI.2.5}$$

The expectations can be approximated with samples obtained from the ESS procedure and c is a parameter. The resulting utility function does not only exploit the expectation, but does also try to explore uncertain parts of the utility space.

VI.3 Actor Critic Relative Entropy Policy Search

In order to perform a policy improvement step, we have to deal with the following requirements. We do not want to assume a known (or approximated) model of the *Markov decision process with preferences* (MDPP) because such an assumption is limiting in many settings. Moreover, we want to be able to use our policy improvement step for continuous valued policies with a large, maybe even infinite number of parameters, such as a *Gaussian process* (GP), as introduced in Section II.3.3.

We will resort to random exploration strategies that can be implemented by a stochastic policy $\pi(a\,|\,s)$. Therefore, we developed a new actor critic algorithm that permits the use of non-parametric policies such as GPs. Our algorithm is based on the *relative entropy policy search* (REPS) algorithm (Peters et al. 2010), but employs a critic that computes a Q-function. Hence, we will call our algorithm *actor critic relative entropy policy search* (AC-REPS).

Our algorithm consists of three steps, which are described in detail in the following sections:

1. Estimate the Q-function using the current estimate of the utility function.
2. Compute sample probabilities for a new policy that maximize the Q-values while limiting the *Kullback-Leibler* (KL) divergence in between the old and the new policy. This technique limits the greediness of the new policy.
3. Fit a new policy to these probabilities.

VI.3.1 Estimating the Q-Function

For estimating the Q-function, we reuse all the observed state transitions (s, a, s') from the environment. We first compute the new utility $U = \boldsymbol{\theta}^T \boldsymbol{\phi}(s)$ for all transitions. Subsequently, we generate new on-policy actions for all successor states in our data set, i.e., $a' \sim \pi(\cdot \mid s)$. Given these pre-processed transition data and a feature representation of the Q-function, i.e., $Q(s, a) = \boldsymbol{\theta}^T \boldsymbol{\phi}(s, a)$, the parameter vector $\boldsymbol{\theta}$ of the Q-function can be estimated by the *least-squares temporal difference learning* (LSTD) algorithm (Boyan 1999).

To increase the robustness of LSTD, we use the regularization method, introduced in Section II.4.2, and include a bias term. We also reuse all samples, independent from the sampling policy. Hence, we have to correct the occurrence probabilities, as explained in Section II.2.2.c. For LSTD, the relevant sampling distribution for a sample (s, a, s', a', r) is $\pi_i(a' \mid s')$ but not $\pi_i(a \mid s)$ (Lagoudakis and Parr 2003a). Hence, we can simply obtain the correct samples by computing a new action $a' \sim \pi_i(a' \mid s')$, which can be performed offline as it does not dependent on the transition dynamics. However, a single, new action does not allow good approximations, therefore we copy each sample 4 times and obtain 4 independent samples for a'. This is possible because LSTD is computational inexpensive in contrast to the other parts of the algorithm.

VI.3.2 Actor Critic REPS

The policy update of AC-REPS is inspired by the episodic REPS algorithm (Kupcsik et al. 2013). We want to find a policy $\pi_{i+1}(a \mid s)$ that optimizes the expected Q-value, but at the same time has a limited KL distance to the old policy $\pi_i(a \mid s)$. We optimize over the joint state-action distribution $\mathrm{Pr}^{\pi_{i+1}}(s, a) = \mathrm{Pr}^{\pi_{i+1}}(s)\pi_{i+1}(a \mid s)$ and require that the estimated state distribution $\mathrm{Pr}^{\pi_{i+1}}(s)$ is the same as the state distribution $\mathrm{Pr}^{\pi_i}(s)$ of the current policy, i.e., $\mathrm{Pr}^{\pi_{i+1}}(s) = \mathrm{Pr}^{\pi_i}(s), \forall s$. This set of constraints is implemented by matching feature averages of the distributions $\mathrm{Pr}^{\pi_{i+1}}(s)$ and $\mathrm{Pr}^{\pi_i}(s)$ (Daniel et al. 2012), i.e., $\int \mathrm{Pr}^{\pi_{i+1}}(s)\boldsymbol{\phi}(s)\, ds = \hat{\boldsymbol{\phi}}$, where $\hat{\boldsymbol{\phi}}$ is the average feature vector of all state samples. Summarizing all constraints, we obtain the following constraint optimization problem

$$\begin{aligned} &\underset{\mathrm{Pr}^{\pi_{i+1}}}{\arg\max} \int \mathrm{Pr}^{\pi_{i+1}}(s, a) Q(s, a)\, \mathrm{d}s\, \mathrm{d}a, \\ &\quad \text{s.t. } \mathrm{KL}(\mathrm{Pr}^{\pi_{i+1}}(s, a) \,||\, \mathrm{Pr}^{\pi_i}(s, a)) \leq \epsilon, \\ &\quad\quad \int \mathrm{Pr}^{\pi_{i+1}}(s)\boldsymbol{\phi}(s)\, ds = \hat{\boldsymbol{\phi}}, \quad \int \mathrm{Pr}^{\pi_{i+1}}(s, a)\, \mathrm{d}s\, \mathrm{d}a = 1, \end{aligned} \tag{VI.3.6}$$

where $\Pr^{\pi_i}(s,a) = \Pr^{\pi_i}(s)\pi_i(a|s)$ is the current state-action distribution. The constraint optimization problem can be solved in closed form by the method of Lagrangian multipliers and has the solution

$$\Pr^{\pi_{i+1}}(s,a) = \Pr^{\pi_{i+1}}(s)\pi_{i+1}(a|s) \propto \Pr^{\pi_i}(s,a)\exp\left(\frac{Q(s,a)-V(s)}{\eta}\right), \qquad \text{(VI.3.7)}$$

where $V(s) = \mathbf{v}^T\boldsymbol{\phi}(s)$ is a state dependent baseline. The parameters η and $\mathbf{v}$ are Lagrangian multipliers that can be obtained efficiently by minimizing the dual function $g(\eta,\mathbf{v})$ of the primal optimization problem

$$g(\eta,\mathbf{v}) = \epsilon\eta + \mathbf{v}^T\hat{\boldsymbol{\phi}} + \eta\log\sum_{i=0}^{N}\frac{1}{N}\exp\left(\frac{Q(s_i,a_i)-\mathbf{v}^T\boldsymbol{\phi}(s_i)}{\eta}\right),$$

where we already replaced the integrals with a sum over samples.

The optimization problem is similar to the one of the contextual REPS algorithm presented by Kupcsik et al. (2013), with the difference that we want to maximize the Q-values that depend on the state instead of returns that dependent on the context. For details of the derivation of the given equations, we refer to the above-mentioned papers and the survey by Deisenroth et al. (2013).

VI.3.3 Obtaining a new Exploration Policy

Effectively, the optimization problem given in the previous paragraph is only solvable given a finite set of state-action pairs s_i, a_i and their corresponding Q-values Q_i. For these samples, we can obtain a desired probability $\Pr(s_i,a_i) = \Pr(s_i)\pi(a_i|s_i)$ from Equation VI.3.7. Our goal is now to generalize this sample-based representation to the whole state-action space with a new parametric (or non-parametric) policy $\tilde{\pi}$. A standard technique to obtain a generalizing distribution $\tilde{\pi}$ from samples is to minimize the KL between π and $\tilde{\pi}$, i.e.,

$$\begin{aligned}\mathbb{E}_s\left[\mathrm{KL}\left(\pi(a|s)\,\|\,\tilde{\pi}(a|s)\right)\right] &= \int \Pr(s,a)\log\frac{\pi(a|s)}{\tilde{\pi}(a|s)}\,\mathrm{d}s\,\mathrm{d}a,\\ &\approx -\frac{1}{N}\sum_{i=0}^{N}\frac{\Pr^{\pi_{i+1}}(s_i,a_i)}{\Pr^{\pi_i}(s_i,a_i)}\log\tilde{\pi}(a_i|s_i) + \text{const},\\ &= -\frac{1}{N}\sum_{i=0}^{N}\exp\left(\frac{Q(s_i,a_i)-V(s_i)}{\eta}\right)\log\tilde{\pi}(a_i|s_i), \qquad \text{(VI.3.8)}\end{aligned}$$

where we replaced the integral by samples, which have been generated by our sampling distribution $\Pr^{\pi_i}(s,a)$. Note that Equation VI.3.8 is equivalent to the weighted negative log-likelihood of policy $\tilde{\pi}$ given the state-action pairs s_i and a_i with weighting

$$w_i = \exp\left(\frac{Q(s_i,a_i) - V(s_i)}{\eta}\right).$$

Hence, minimizing the expected KL is equivalent to a weighted maximum likelihood (ML) estimate of $\tilde{\pi}$. Note that, for the AC-REPS algorithm it is sufficient to have access to samples from $\Pr^{\pi_i}(s,a)$. Hence, we can reuse all samples obtained from any iteration. However, we have to re-weight the samples using importance sampling, as explained in Section II.2.2.c. In contrast to the LSTD step, we do not use resampling as AC-REPS is computational expensive. In case of discrete action spaces, this can be improved by creating samples for each action for each observed state, regardless of the observed action. These samples have then to be weighted by $\pi(a\,|\,s)$.

Weighted ML estimates can be obtained in closed form for many types of distributions. We will use a GP policy. In order to keep the computation tractable, we adapt the weighted formulation of a GP given by Kober et al. (2010) to the sparse GP case (Snelson and Ghahramani 2006).

VI.3.4 Policy Improvement with Utility Functions

In contrast to reward-based RL algorithms, we have to consider that the utility estimate may change between iterations. Hence, the quality of an already evaluated policy has to be re-evaluated after an utility update and we can no longer assume that our current policy $\pi_i(a\,|\,s)$ is an improvement over older policies. All approaches to PBRL disregard this problem or perform multiple policy improvement steps after a new utility function is obtained. Both methods are suboptimal as several updates, and therefore policy evaluations, are required to move the policy into a direction that maximizes the new utility function. Hence, either a high number of trajectory samples or model-based approaches are used.

We don't start our AC-REPS step with the current policy, but use the policy maximizing $\mathbb{E}_{\mu(s)}\hat{V}(s)$, out of all already obtained policies. The average state value over all initial states, based on the updated utility function, is computed using the LSTD procedure from Section VI.3.1. Hence, we start our improvement step with the policy that is closest to the expected maximum.

VI.4 Creating new Preference Queries

The new, stochastic REPS policy is now used to generate n new trajectories as additional transition samples. However, as we want to minimize the number of required preference queries, only one preference per iteration is requested. This is a direct result of Section IV.6.1, where we determined that it is beneficial to update the policy before requesting additional preferences. For requesting new preferences, we compare four different selection strategies. As we are foremost interested in finding highly ranked trajectories (cf. Section III.2), we maintain a currently undominated trajectory $\boldsymbol{\tau}^*$. Preference queries are created by computing a selection criterion for all pairings of undominated trajectories with any other observed trajectory and posing a query with the pair that maximizes the criterion. This technique keeps the selection process computationally tractable, as we don't need to compute the criterion for $|\Upsilon|^2$ pairings, but only $|\Upsilon|$ times, as we only have one undominated trajectory, according to the total order assumption introduced in Section III.1. Furthermore, the additional information obtained by the preference query reduces the uncertainty concerning the expected, optimal trajectory, as it is involved in the query.

Some of the proposed criteria require access to an updated posterior. Sampling from the real posterior is computational expensive. Hence, we only approximate the new posterior by reweighing the already obtained samples, based on the new likelihood function. We use the importance sampling explained in Section II.2.2.c, but that may introduce the mentioned approximation errors, in case the distribution changes significantly.

VI.4.1 Expected Improvement

Considering that we assume the updated utility estimate to be more accurate, it is sensible to select the trajectory with the highest, expected utility

$$\boldsymbol{\tau} = \arg\max_{\boldsymbol{\tau} \in \Upsilon} \mathbb{E}_\zeta \left[U_\zeta(\boldsymbol{\tau}) \right]. \tag{VI.4.9}$$

This *expected improvement* (EI) criterion does only select a single trajectory that should be compared to the currently undominated trajectory $\boldsymbol{\tau}^*$. Hence, the method evaluates if the expert confirms the expected improvement. This criterion tries to exploit the expert's preference function ρ, but disregards exploration. However, as we are not maximizing the selection criterion but only select out of a given sample set, we still explore partially.

VI.4.2 Query by Disagreement

As we have access to the posterior distribution of the utility function (cf. Section VI.2), we can use ideas derived from active learning of classifiers (Seung et al. 1992; Freund et al. 1997). Wilson et al. (2012) proposed to use the samples obtained from the posterior to formulate a *query by disagreement* (QBD) method concerning the expected preferences.

$$\begin{aligned} \boldsymbol{\tau} &= \arg\min_{\boldsymbol{\tau}\in\Upsilon} |\Pr(\boldsymbol{\tau} \succ \boldsymbol{\tau}^* \mid U) - 0.5|, \\ \Pr(\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j \mid U) &= \mathbb{E}_\zeta \left[\mathbb{1}\big(U_\zeta(\boldsymbol{\tau}_i) - U_\zeta(\boldsymbol{\tau}_j) > 0\big)\right], \end{aligned} \tag{VI.4.10}$$

determines the probability of preference $\Pr(\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j \mid U)$, based on the current utility difference. In contrast to Wilson et al. (2012), we don't use a distance value, but only the binary expectation of the preference symbol. The pairing where samples disagree the most, as the expectation is closest to 0.5, is selected. This criterion tries to maximize the part of the utility distribution that can be invalidate with a single preference query.

VI.4.3 Expected Utility of Selection

A drawback of the QBD method is that it does not consider the actual improvement over the current, best trajectory. The new preference may influence the utility distribution, but not specific $\boldsymbol{\theta}$ used to derive a new utility function, e.g., the expectation of the distribution may not change. Hence, Akrour et al. (2011, 2012) suggested to use the *expected utility of selection* (EUS) as a selection criterion.

$$\begin{aligned} \boldsymbol{\tau} &= \arg\max_{\boldsymbol{\tau}\in\Upsilon} \tilde{U}(\boldsymbol{\tau}), \\ \tilde{U}(\boldsymbol{\tau}) &= \Pr(\boldsymbol{\tau} \succ \boldsymbol{\tau}^* \mid U)\mathbb{E}_{\zeta\cup\{\boldsymbol{\tau}\succ\boldsymbol{\tau}^*\}}\left[U(\boldsymbol{\tau})\right], \\ &\quad + (1 - \Pr(\boldsymbol{\tau} \succ \boldsymbol{\tau}^* \mid U))\mathbb{E}_{\zeta\cup\{\boldsymbol{\tau}\prec\boldsymbol{\tau}^*\}}\left[U(\boldsymbol{\tau}^*)\right], \end{aligned} \tag{VI.4.11}$$

computes the expected utility $\tilde{U}$ when adding the result of the potential query to the preference set. In case the new trajectory dominates the current, best one, the utility for the new trajectory is used. If the trajectory is not improving over the current best, the new utility of the old optimal one is used. The probability of preference is estimated in the same way as QBD. EUS can be seen as an improved version of EI as it also considers the probability of improvement.

VI.4.4 Expected Belief Change

The EUS method considers the actual improvement to be expected, and tries to exploit that, but it does not try to explore uncertain parts of the utility space. For overcoming this problem, Wilson et al. (2012) suggested to use the *expected belief change* (EBC) as a selection criterion, based on an approximation of the variational distance between two policies. We propose to use the expected KL distance

$$\begin{aligned} &\mathrm{KL}(U_\zeta \,\|\, U_{\zeta\cup\zeta_i}),\\ &= \Pr(\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2} \mid U)\mathrm{KL}(U_\zeta \,\|\, U_{\zeta\cup\{\boldsymbol{\tau}_{i1}\succ\boldsymbol{\tau}_{i2}\}}),\\ &\quad + (1 - \Pr(\boldsymbol{\tau}_{i1} \succ \boldsymbol{\tau}_{i2} \mid U))\mathrm{KL}(U_\zeta \,\|\, U_{\zeta\cup\{\boldsymbol{\tau}_{i1}\prec\boldsymbol{\tau}_{i2}\}}), \end{aligned} \qquad \text{(VI.4.12)}$$

over the utility distribution as a measure. The KL distance can be computed using samples from the Bayesian estimate of $\Pr(U \,\|\, \zeta)$ (cf. Section VI.2.2). Like the EUS criterion, it considers the probability of preference $\Pr(\boldsymbol{\tau} \succ \boldsymbol{\tau}^* \mid U)$ but does measure the change induced to the utility distribution. Like in the QBD approach, the idea is to reduce the uncertainty as quickly as possible.

VI.5 Experiments

As an empirical validation, we implemented the proposed algorithm and tested it on four common RL tasks. We compare the different query strategies proposed in Section VI.4 and show the advantage of the EBC method. We also analyze the effects of tuning the UCB-c parameter (VI.2.5) and the AC-REPS-ϵ (VI.3.6) parameter. Both parameters control the interdependent exploration of the system dynamics and the expert's preference function. Hence, they are important for obtaining good results and we can also demonstrate the effects of our policy selection method, introduced in Section VI.3.4. Furthermore, we evaluate the algorithm's behavior when obtaining noisy feedback.

In our experiments, the learner does not have access to the true reward signal but is given preference feedback based on the return. All reported results are averaged over 30 trials. We always collect 10 trajectories per iteration and request 1 preference per iteration. For the ESS, we use $200k$ samples but discard the first $50k$ samples as burn-in phase before the sampling strategy correctly approximates the posterior. The LSTD regularization factors, the RBF bandwidth and number of basis function, are only tuned once per domain, to determine a reasonable setup. The ϵ bound of AC-REPS is manually tuned on the preference-based tasks. The sigmoid shape parameter m and the variance of the prior σ are fixed in advance, without significant tuning. The GP hyper-parameter are automatically tuned, using a covariance matrix adaptation evolution strategy (CMA-ES; Hansen and Ostermeier 2001) after each iteration. The optimization target is to minimize the training set error, so that no additional data was required.

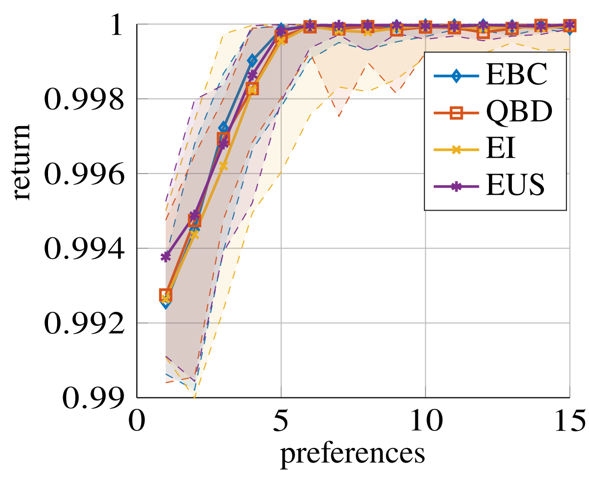
In the following graphs, the solid lines define the median while the shaded areas and the dashed lines show the 25% and 75% percentile of the policy's return. The x-axis shows the number of obtained preferences, which is identical with the number of iterations.

VI.5.1 Preference Selection Criteria

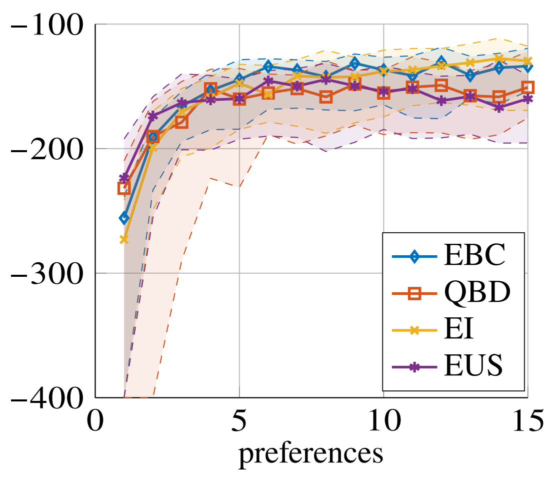
In this section, we compare the *expected belief change* (EBC), *query by disagreement* (QBD), *expected improvement* (EI) and *expected utility of selection* (EUS) preference query selection criteria proposed in Section VI.4. The first task is the bicycle balance task, as explained in Section A.2, where we use continuous states and actions with a GP policy. 400 RBF centers are used for the GP and the utility function approximation. As defined by Akrour et al. (2014), we use the *average* reward as preference signal. However, this setup is not exactly identical to the work of Akrour et al. (2014), because we do not require a generative model and use a Gaussian process policy instead of a neural network. In this domain, all selection criteria perform nearly identical, considering the median and percentile plots in Figure VI.5.1a, as explained in Section VI.5. This is in line with Wilson et al. (2012), who evaluated a very similar task and also determine that the query criterion does not substantially influence the result. The number of required preferences is already extremely low, as five expert queries are usually sufficient to achieve convergence.

As a second testing domain, we show the results from the acrobot setup as Figure VI.5.1b, introduced in Appendix A.1. In contrast to the variant explained in the appendix, we use a continuous state and action space. The kernels for the utility function and the GP use 300 centers. As in the bicycle balance domain, we can only see minor differences between the different selection strategies. However, the QBD strategy and the EUS method are not able to obtain a policy that is on par with the other two setups, but the difference is not significant. In comparison to the *every-visit preference Monte Carlo* (EPMC) approach (Section IV.2), we only require up to a dozen preference queries, instead of hundreds. However, part of the improvement can probably be contributed to the GP function approximator used for the policy.

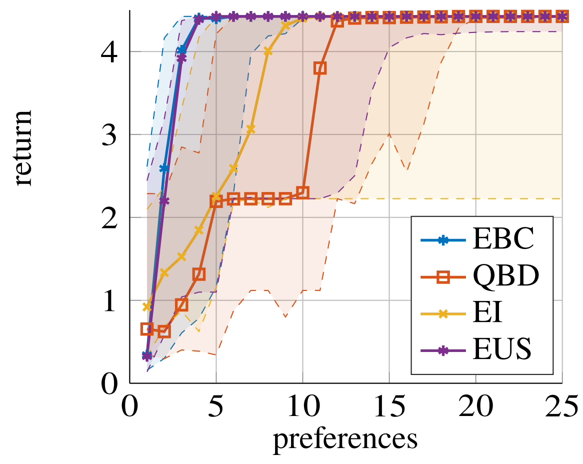
Clearer results can be obtained in the grid world domain (Figure VI.5.1c), defined by Akrour et al. (2014). A detailed description can be found in Appendix A.4. For allowing a comparison to the *programming by feedback* (PF) algorithm by Akrour et al. (2014), we don't use a GP, but a tabular representation. Figure VI.5.2 shows the results. The EUS and the EBC strategy clearly outperform the two alternatives. Both methods consider the expected preferences symbol as well as the expected change induced by adding a new preference. However, it seems that using the expected improvement as a measure of change can be problematic. Not only the EI strategy use



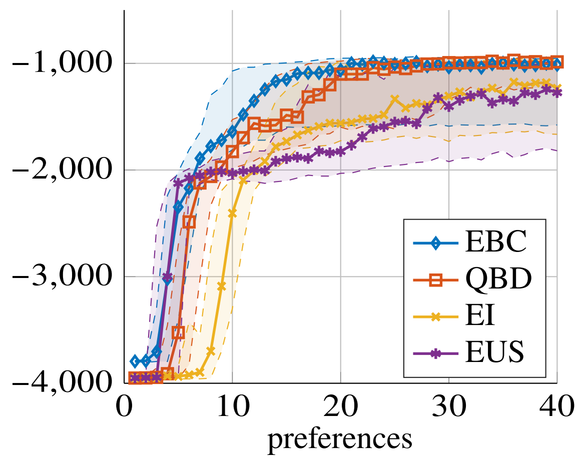
(a) Bicycle Balance



(b) Acrobot



(c) Grid world



(d) Swing Up

Figure VI.5.1.: Comparison of preference query selection criterions

this measure, but also the EUS method and both techniques struggle with convergence in some cases, as can be seen by the lower quartile.

The last and most difficult domain is the swing up from Appendix A.6, with the results shown in Figure VI.5.1d. Here, we use 700 RBF centers. We can again observe the problem from the grid world domain, that the expected improvement-based methods struggle with convergence. All methods approach a plateau of −2000 quickly, but only the QBD and the EBC methods are able to overcome this level fast.

We conclude that exploitative techniques like EI and EUS are not sufficient as a selection criterion, because they disregard exploration. However, it is useful to consider the probability of preference in combination with a measure of change, as can be seen by the results from EUS

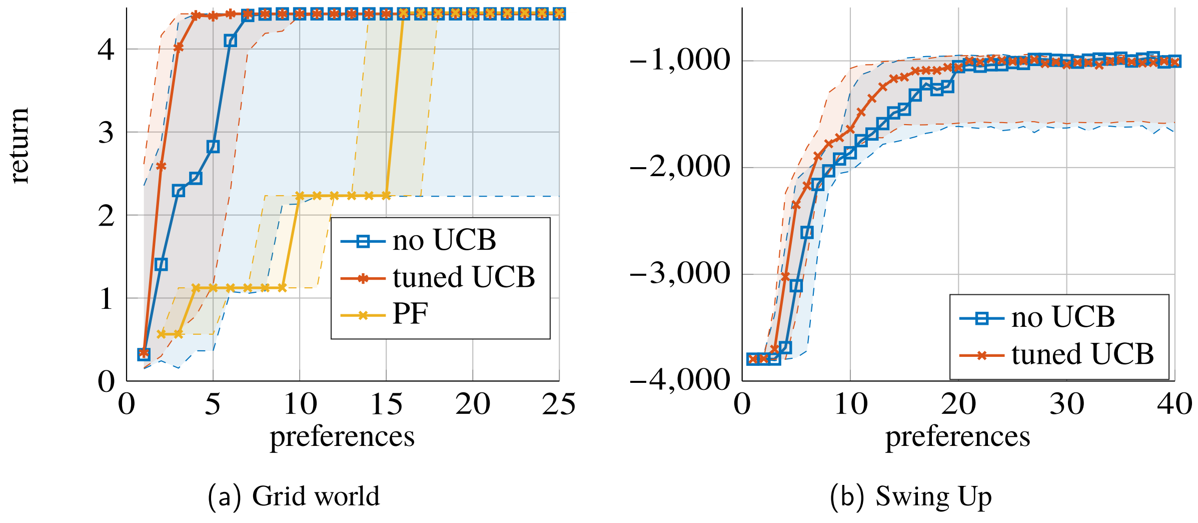


Figure VI.5.2.: The influence of UCB on utility space exploration

and EBC. Hence, the EBC criterion is the most useful, because it combines the probability of preference with a measure that enables exploration.

VI.5.2 UCB-based Exploration

As described in Section VI.2.2, we use UCB to guide the policy optimization process into areas with promising utility values. When only exploring the transition dynamics and purely exploiting the utility function, we may not be able to obtain trajectories that allow use to derive preference queries that are capable of reducing the uncertainty quickly. However, as we already use exploring query selection methods, this additional exploration may not be necessary. For evaluating this problem, we show a comparison between a setup with a tuned UCB c value and disabled UCB in the setting with the EBC preference query strategy. We choose the grid world and the swing up domain for this comparison as they seem to be dependent on selecting good preference learning strategies. In the grid world, as shown by Figure VI.5.2a, we can see that UCB helps in obtaining optimal policies. Without the confidence interval, the policy may converge to a suboptimal solution. The lower quartile of the no UCB approach is worse than the one of a well tuned c parameter in the grid world domain. Furthermore, the UCB approach leads to a speedup in convergence. In the swing up domain (Figure VI.5.2b), we can only observe minor differences between both versions. However, the UCB approach converges slightly faster.

Figure VI.5.2a also shows a comparison to the PF algorithm by Akrour et al. (2014) in the grid world domain. The PF algorithm uses directed exploration, but this seems to be not necessary as our undirected method outperforms this approach.

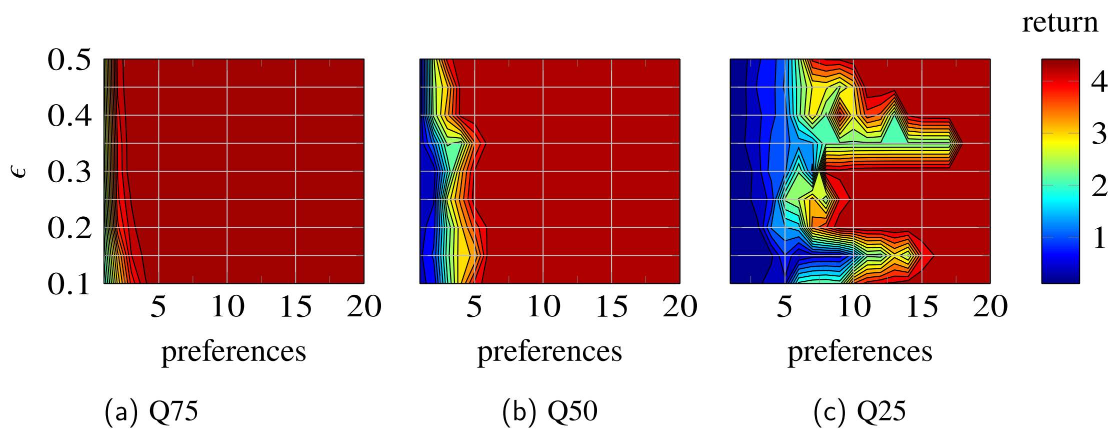


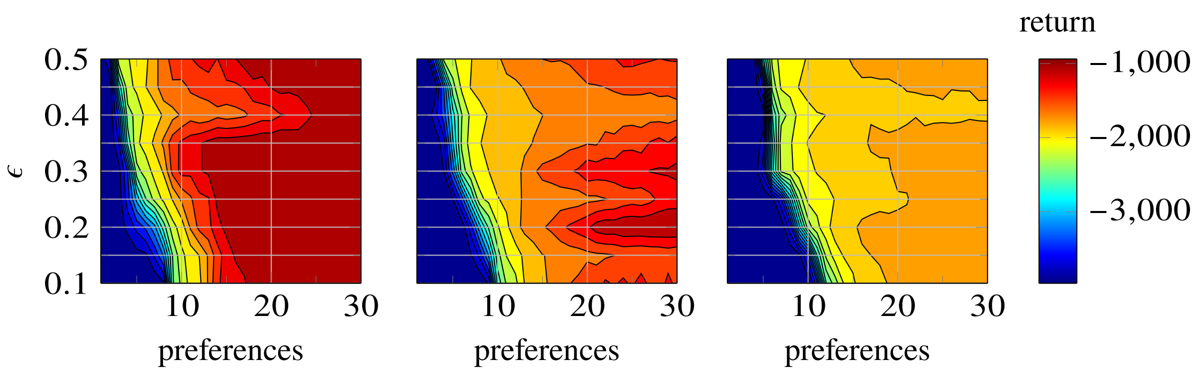
Figure VI.5.3.: Influence of ϵ in the grid world domain

ϵ	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
mean correction	2.64	2.71	3.9	3.4	3.39	3.47	4.08	4.32	4.43
std correction	3.42	3.28	4.24	3.8	3.69	3.86	4.25	4.49	4.38

Table VI.1.: Policy iteration index correction amount in the grid world domain

VI.5.3 REPS-based Exploration

After analyzing the UCB-based preference feedback exploration, we also take a closer look at the system dynamics exploration problem solved by the AC-REPS system, introduced in Section VI.3. The following contour plots show the median and quartiles of the obtained trajectory return, as in Section IV.6.2. The x-axis is the number of obtained preferences and the y-axis relates to the hyper-parameter ϵ. Figure VI.5.3 shows the results from the grid world domain, where we can see that the approach is able to converge independently of the ϵ value. The median (Figure VI.5.3b) and upper quartile plot (Figure VI.5.3a) shows that higher ϵ values slightly speed up convergence. This was to be expected, because this domain is rather simple from a RL point of view and therefore, we can use a more greedy approach. When considering the lower quartile (Figure VI.5.3c), no clear trend is visible. This can be explained by the preference function exploration problem. A greedy approach is only useful when we can derive a correct utility function quickly. If we are not able to access trajectories that enable a fair approximation of the utility function, we require trajectories that contain information for efficiently reducing uncertainty. Meaningful trajectories should differ from the already obtained set (high ϵ), but must also be variant enough to let the selector choose efficiently (low ϵ). Hence, we have to select a good tradeoff with an $\epsilon \sim 0.3$. In case the parametrization is much too greedy ($\epsilon > 0.35$), our correction method explained in Section VI.3.4 kicks in. This can be seen in Table VI.1, showing the mean and standard deviation of the difference of the index between the current policy and



(a) Q75 (b) Q50 (c) Q25

Figure VI.5.4.: Influence of ϵ in the swing up domain

ϵ	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
mean correction	1.14	0.88	0.92	0.52	0.58	0.62	0.77	0.9	0.89
std correction	2.32	1.88	1.55	1.02	1.02	1.05	1.23	1.5	1.46

Table VI.2.: Policy iteration index correction amount in the swing up domain

the policy selected by the correction method. Except for the outlier at $\epsilon = 0.2$, we can see only slight increases of the influence of the correction method until $\epsilon = 0.35$ where the policy starts to degrade substantially. For higher ϵ, the policy selection method induces major corrections, as can be seen by the mean and standard deviation of the correction. As we fall back to old policies, if required, we achieve additional exploration, moving the results closer to a good tradeoff.

In the swing up domain, Figure VI.5.4, we can observe a similar behavior. Results improve from $\epsilon = 0.1$ to $\epsilon = 0.2$ until the approach becomes to greedy. The arising problem is first visible in the lower quartile (Figure VI.5.4c), but also becomes prominent in the other plots when approaching $\epsilon = 0.4$. Increasing ϵ further is again much too greedy and the $\mathbb{E}_{\mu(s)} \hat{V}(s)$ based correction results in a better tradeoff. Table VI.2 shows the difference in the iteration index between the selected and the current policy, as in Table VI.1. Again, we can observe an increase in correction when surpassing $\epsilon = 0.35$, but in contrast to the grid world, the correction also increases when ϵ falls below 0.25. This behavior can be explained when considering the standard deviation, where we observe much higher values for low ϵ. When not updating greedily enough, the information contained in new trajectories is not sufficient to derive a substantially different utility function. Several, small updates must accumulate till the utility function changes substantially and the correction induces a large setback. For too high ϵ, the correction happens more often as single updates already result in greatly different trajectories. In the acrobot domain (Figure VI.5.5), all settings for ϵ approach a good policy quickly, but struggle with further improvements. This is especially visible when considering the lower quartile. The ϵ value does

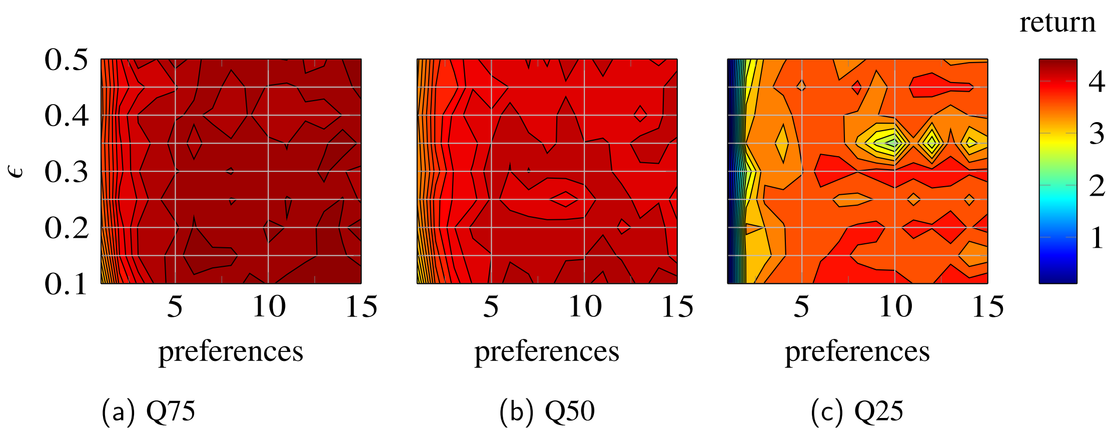


Figure VI.5.5.: Influence of ϵ in the acrobot domain

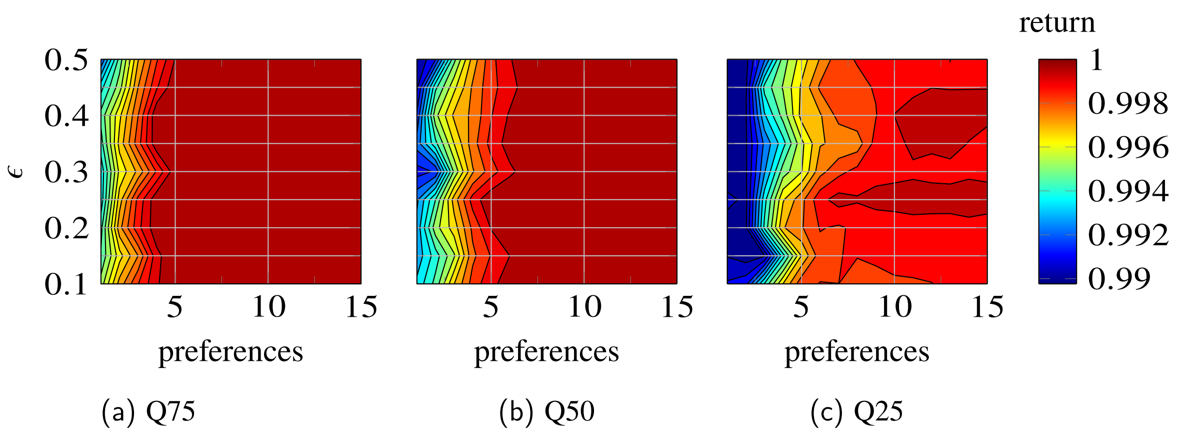
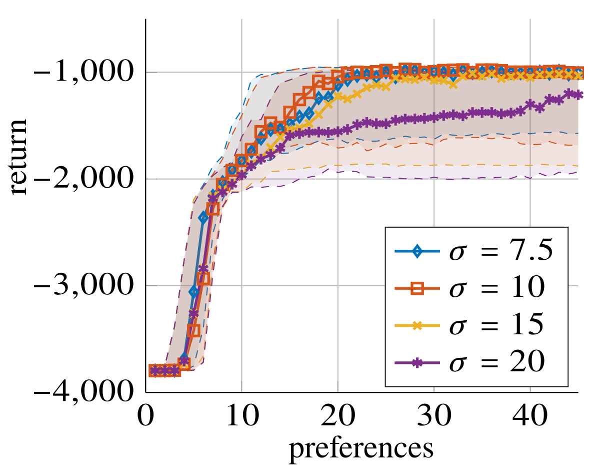


Figure VI.5.6.: Influence of ϵ in the bicycle balance domain

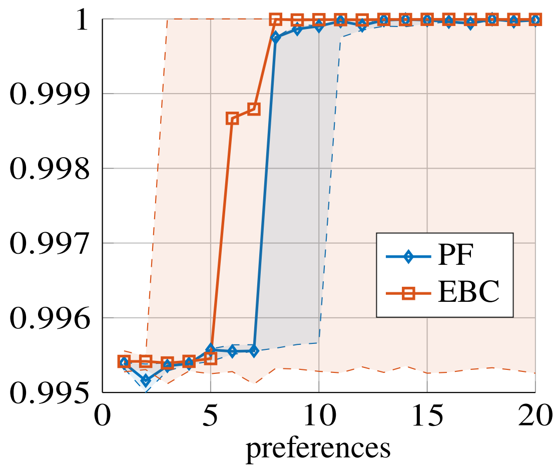
not influence the result substantially. Hence, we determine that it is difficult to approximate a utility function in this domain. Figure VI.5.6 shows the results from the bicycle balance domain. We can not observe a substantial influence of the ϵ parameter considering convergence speed. However, the lower quartile shows slight differences with a sweet spot around $\epsilon = 0.25$. The similarity of results further confirms the findings in Section VI.5.1 and by Wilson et al. (2012) that exploration methods are not important in that domain. We argue that the learning phase is mainly limited by number of obtained samples and efficient use.

VI.5.4 Noise

To confirm the noise robustness of our algorithm, we use a noisy version of the swing up domain. The swing up is difficult from a PBRL point of view because parts of the state space are not accessible without a good policy (cf. Section VI.5.1). Hence, it is difficult to learn a utility



(a) Swing Up, Gaussian Noise



(b) Bicycle Balance, Rigid Noise

Figure VI.5.7.: Effect of noisy preferences

function for these states, because only later trajectories will carry the relevant information that allow to get into these regions. We use the same swing up setup as before, but add a normal distributed noise value to the reward before computing the preferences. Figure VI.5.7a shows the results obtained with different values for the standard deviation of the reward noise. The algorithm is stable against small variations of the reward. This was to be expected, as preferences are indifferent to reward changes unless they flip the sign of the return difference. However, the algorithm is also robust against changing preferences as can be seen by $\sigma = 20$. The number of required preferences increases substantially, but the algorithm still converges.

For an additional comparison to the PF algorithm, we reimplemented the exact same bicycle balance domain and obtained the according results from Akrour et al. (2014). In difference to the bicycle balance experiments in the other sections, trajectories are not terminated after the bike falls over but the bike will remain in its last state. Hence, the average reward can be substantially lower because this last state dominates the calculation in early iterations. We also use the noise model introduced in the according publication. The rigid noise model

$$\begin{aligned} \Pr(\boldsymbol{\tau}_i \succ \boldsymbol{\tau}_j) &= \max(0, \min(1, f(R(\boldsymbol{\tau}_i), R(\boldsymbol{\tau}_j)))), \\ f(R_i, R_j) &= \frac{(R_i - R_j) + 0.1}{2.1}, \end{aligned} \tag{VI.5.13}$$

defines a probability of preference depending on the true return difference. In Figure VI.5.7b, we can observe that our new algorithm using EBC and UCB outperforms the PF approach considering the median and the upper quartile. However, in contrast to the algorithm by Akrour et al. (2014), the lower quartiles show convergence problems. This can be explained by the used exploration method. Fully optimizing a directed exploration criterion creates trajectories that may differ greatly. Trajectories created by undirected exploration are usually more similar. Therefore, the preference queries created by directed exploration often compare trajectories that differ to a higher degree than the ones created with the undirected exploration method. In turn, the probability of a wrong preference is lower when using directed exploration. This theory is supported by the fact that we can observe a higher number of wrong preferences when using our undirected exploration method in combination with the rigid noise model. However, it should also be considered that we evaluate based on 30 trials, whereas Akrour et al. (2014) only uses 10 trials. Hence, the PF results may not be exact enough and the algorithm could suffer from similar problems.

VI.5.5 Runtime

The ESS requires $80 - 90$ sec. for sampling in the grid world domain and $100 - 150$ sec. in the three other domain. However, the GP calculation dominates the required CPU time and a single iteration can take between 1 and 30min, depending on the domain and the already obtained preferences. With a runtime of several minutes per iteration, this algorithm is computational expensive. Applying the same algorithm to larger problem spaces probably requires methods for speeding-up the calculations.

VI.6 Conclusion

We have demonstrated that it is possible to use PBRL in an online manner, even in a non-parametric, model-free setting with continuous state action spaces. Hence, we successfully solved Research Question 3.B. We also incorporated a noise-tolerant loss function (cf. Research Question 3.A) by embedding a sigmoidal loss-based utility calculation with the policy optimization. However, the noise tolerance can still be improved.

Our results also show that complex directed exploration that are often used for the preference case might be unnecessary. Random exploration is sufficient if the exploration/exploitation trade-off can be controlled efficiently. Furthermore, undirected exploration is sufficient for exploring the expert's evaluation function when coupled with an efficient preference query selection criterion, answering Research Question 3.C. In some domains, further improvements are possible when using UCB as additional exploration criterion, but this can also be realized in combination with the proposed undirected exploration method.

Additional improvements can be observed when adapting to the problem of a changing utility function. In classic RL we can assume steady policy improvements, because the value of observed policies does not change. However, in the PBRL setting, it is beneficial to re-evaluate old policies, based on improved utility estimates and adapt the policy iteration framework accordingly.

Concerning a good preference query selection criterion, it is required to use a measure of change, that allows sufficient exploration. Exploitative methods are usually not sufficient. However, it is usually beneficial to use a strategy that also considers the probability of a preference. EBC satisfies both requirements and should therefore be used as a selection method considering the analyzed methods, answering Research Question 3.D.

Chapter VII

Wrap Up

VII.1 Summary

Throughout this thesis, we answered several specific research questions. In this section, we recap the questions and the obtained answers and show how they relate to the problems and characteristics of *preference-based reinforcement learning* (PBRL) algorithms, presented in Chapter III. Table VII.1 shows a compact representation of this section. The references in title lines point to the description of the characteristic/problem whereas the other references relate to the relevant experiments. In general, we focus on trajectory preferences, to move complexity from the expert to the algorithm, as explained in Section III.3.1. Furthermore, we use undirected exploration (cf. Section II.4.1) as these techniques usually scale better to higher-dimensional problem spaces.

In Chapter III, we introduced the different design principles for PBRL algorithms. This allowed us to define a unifying framework and categorization for surveying the field. Furthermore, advantages, assumptions and drawbacks of different algorithms are described explicitly, enabling us to discuss the relation to our methods in Section VII.2.

The goal of Chapter IV was to analyze the problems introduced by assuming preference feedback. Research question 1.A was how to approximate a solution for the temporal credit assignment problem in PBRL, where we presented two methods. Both use heuristic approximations for intermediate preferences, but we could show that the incorporation of prior information improves the results. These methods also allowed us to determine the impact of the temporal credit assignment problem as well as the binarization of feedback, due to preferences. We analyzed the impact by comparing with a classic *reinforcement learning* (RL) method to answer research the according questions 1.B and 1.C. We also compared the methods in term of trajectory and preference sample efficiency (Research Question 1.D). We found out that efficient approximations are probably most important for efficient PBRL. The binary feedback does not seem to be a substantial problem. Furthermore, a well-defined exploration/exploitation tradeoff is of importance, as in classic RL.

Following the insight from Chapter IV, we decided to switch to utility-based approximations for the temporal credit assignment problem, as they are possibly more efficient. However, they may result in large approximation errors, as explained in Section III.3.2.c. This can be especially problematic when dealing with preferences defined by humans as they are possibly unreliable. Therefore, we compared methods for learning from human preferences within Chapter V. Research question 2.A dealt with the problem of learning a utility-function within a setting that allowed us to discard most classic RL problems. The obtained results show that it is possible to efficiently learn from human preferences, despite some problems that humans introduce. Following the insight, we tried to determine the problems that occur when using human preferences as Research Question 2.B. We determined that missing information, number of preferences and

	Pref. (III.3.1)	L. Problem (III.3.2)	Temp. Credit (III.3.3)	Traj. Gen. (III.3.4.a)	Pref. Gen. (III.3.4.b)	Dynamics (III.3.6)
Chapter IV	traj. (III.3.1.c)	pref. model (III.3.2.b)	intermediate	homogen	exhaustive	model-free
1.A, approximate temporal credit assignment solutions			IV.6.1, IV.6.3			
1.B, impact of approximate temporal credit assignment			IV.6.1, IV.6.3			
1.C, impact of binary feed-back			IV.6.3			
1.D, sample efficiency				IV.6.1	IV.6.2	
Chapter V	state (III.3.1.b)	utility (III.3.2.c)	value-based (III.3.3.a)	-	-	model-based
2.A, preference-based utility learning		V.5, V.6.3				
2.B, problems induced by hu-man preferences		V.6.1, V.6.2				
2.C, relevance of induced problems		V.6.3				
Chapter VI	traj. (III.3.1.c)	utility (III.3.2.c)	reward-based (III.3.3.c)	homogen	interleaved	model-free
3.A, noise-tolerant PBRL		VI.5.4				
3.B, model-free, non-parametric PBRL						VI.5.1
3.C, undirected exploration for trajectory generation				VI.5.2, VI.5.3		
3.D, efficient query selecting strategies					VI.5.1	

Table VII.1.: Research questions in relation to PBRL categorization and experimental results

noise are potential problems. However, further studies showed that noise is the severest problem and should be dealt with explicitly (Research Question 2.C). Hence, we implemented a noise-tolerant method for learning a utility function. We have been able to show that non-linear loss functions are better suited for learning from human preferences than linear loss functions.

We then used the obtained insight to define a novel, model-free, non-parametric, noise-tolerant PBRL algorithm with undirected exploration in Chapter VI. The general task of creating an according algorithm is Research Question 3.B. To our knowledge, all other available algorithms are model-based and/or require a parametric policy space. We showed that the new algorithm is very preference-efficient, outperforming the current state of the art in several domains. Research question 3.C considers the problem of how to generate trajectories that are well suited for exploring the transition dynamics as well as the preference function space, in an undirected manner. It seems to be beneficial to guide the policy improvement step in a way that increases the probability for trajectories with potentially high return while also considering the uncertainty of the utility function. We also analyzed how to reduce the number of required preference queries by adding intelligent query selection method as Research Question 3.D. Several query selection methods have been proposed in the literature and we compared different concepts to validate existing ideas and give further insight. Efficient methods should consider the expected probability for each preference symbol while also computing a measure of expected change. Methods based on expected improvement have difficulties to escape local optima. Finally, we also confirmed that a sigmoid-like loss function is also suitable for handling noise when embedded into this newly proposed algorithm (Research Question 3.A).

VII.2 Discussion

When learning with human feedback, the human work-time is usually considered more expensive than computation time. Therefore, most of the state-of-the art research in PBRL aims at learning with a low number of preferences. This usually requires to focus on three tasks:

1. Maximizing the information contained in obtained preferences
2. Efficiently using and generalizing the obtained information
3. Handling the unreliability of human feedback

In the following, we discuss the related results obtained in this thesis as well as the work that has been published on using human feedback in PBRL.

VII.2.1 Optimizing Preference Query Creation

A first method to maximizing the information content was presented by Akrour et al. (2011), where they introduce an exploration/exploitation tradeoff for searching in the preference space. The results show a substantial advantage for the intelligent preference query method. Results by Akrour et al. (2012), Wilson et al. (2012), and Akrour et al. (2014) further support the need for well-defined query selection methods. However, most methods completely ignore the problem and assume exhaustive preference sampling or let the human expert decide which preference feedback to pose, as explained in Section III.3.4. We showed that the basic principle of several of the mentioned selection strategies are also applicable to other algorithms (cf. Section VI.5.3). Hence, a significant number of publications that support the relevance of intelligent preference query strategies are now available. All available publications show that it is useful to consider the expected probability for a preference symbol (cf. Section VI.4.10) as part of the query strategy. However, it is currently still unclear how to combine the preference space exploration method with the exploration of the transition dynamics. Akrour et al. (2012) used an exploration method that independently gathers information from the transition dynamics, till the preference space exploration criterion is maximized. However, this is not required and can be suboptimal as the preference criteria are usually based on expectations that suffer from high variance when using only few preferences. Hence, it can be beneficial to not fully maximize the criteria, but obtain new preferences earlier. Akrour et al. (2011, 2012) and Akrour et al. (2014) terminate the maximization process prematurely while Wilson et al. (2012) and Wirth et al. (2016) select a limited number of trajectories already available from the dynamics exploration process, as explained in Section III.3.4.b. Enhancing the exploration process with information over the uncertainty of the utility function can further speed up the PBRL process (cf. Section VI.5.2).

VII.2.2 Efficient Preference Usage

Efficiently using and generalizing obtained preferences requires a well-defined function space for the preference learning problem. This can either be a parametric policy space or a state-action feature space (cf. Section III.3.2). The dominant problem for efficiently learning from preferences is the dimensionality of the function space. Low dimensional spaces have few, free parameters and can therefore be trained with few preferences, but may not allow a sufficient approximation of the optimal function. Hence, the approximation is usually a tradeoff between using expert knowledge that allows to define low-dimensional spaces and general methods like kernels, as explained in Section II.3. General methods are less demanding for the expert but usually result in a higher number of free parameters and therefore require a higher number of preferences. In several domains, especially robotics (Deisenroth et al. 2013), parametric policy spaces are available, which allow low-dimensional approximations using expert knowledge. Wilson et al. (2012) and Kupcsik et al. (2015) use this information to define efficient PBRL algorithms, but they are restricted to Monte Carlo-based evaluation procedures. As explained in Section IV, value-based methods are possibly more sample-efficient but may suffer from approximation errors. Akrour et al. (2011, 2012) show that it is sensible to use intelligent mapping strategies for trajectory features that allow to obtain linear function approximations while also limiting the dimensionality of the function space. However, they use a mapping strategy that prevents the use of a reward-based utility, as the trajectory features are not separable into state-action features. Hence, value function methods for RL are not applicable. Akrour et al. (2013), Akrour et al. (2014), and Wirth et al. (2016) did overcome this problem by using the principle of feature averages (III.3.16). This gave rise to a class of PBRL algorithms that can separate the *preference learning* (PL) problem form the RL problem by computing a state-action utility from preferences. Therefore, value-based RL methods can be used to solve the policy evaluation/improvement problem, as described by Akrour et al. (2014) and Wirth et al. (2016). Due to their efficiency, they define the current state-of-the-art for PBRL without parametric polices. However, the algorithm by Akrour et al. (2014) requires an approximation of the transition dynamics and is therefore not model-free.

VII.2.3 Dealing with Human Feedback

When dealing with human feedback, multiple problems can arise. Noise can be an issue, because a human evaluation can not be assumed to be clear cut and can be stochastic. The common assumption is that the probability of noise scales with the expert's utility difference. Preferences concerning trajectories that are of similar quality are more likely to be wrong. Furthermore, it may be impossible to approximate the optimal utility function perfectly, due to limitations of the function space. In this case, approximation errors for similar trajectories usually have a smaller influence on the optimality of the resulting policy, because the expected realization probability is similar anyhow. Hence, loss functions that scale with the utility difference are preferable, as used by Wilson et al. (2012), Akrour et al. (2013), Akrour et al. (2014), and Kupcsik et al. (2015) and in Chapter V & VI. However, the assumption was only analyzed and validated by Akrour et al. (2013) and Akrour et al. (2014) and in Chapter V of this thesis. As described in Section V.6, additional problems can arise when allowing the human to choose the trajectories to evaluate, because the evaluation can be biased. Hence, human-guided preference selection (cf. Section III.3.4.b) can be problematic.

VII.3 Conclusion

In this thesis, we have been able to show that it is important to obtain an efficient, approximate solution to the temporal credit assignment problem. Reward-based utility methods are especially interesting as they allow to employ value-function-based RL methods like *least-squares temporal difference learning* (LSTD). Value-based methods are preferable for PBRL, in case no well-defined parametric policy space is available beforehand. Due to the human involvement in the evaluation, low sample counts are usually more important than highly optimal solutions. Hence, the current state-of-the-art either uses reward-based methods or parametric policies. In contrast to the prior state-of-the-art, we have shown a method that is model-free as well as non-parametric and can limit the number of required trajectory samples. The method of feature averages offers a general solution to the reward-based utility problem, that can be coupled with different PL methods while separating the RL problem. For computing a utility function, we have shown that it is beneficial to incorporate prior knowledge, in form of assumptions over the utility distribution. Especially Bayesian methods are well suited for the problem, because they allow more flexible priors and allow the computation of uncertainty estimates. Uncertainty estimates can be used to efficiently guide the trajectory creation and preference query selection. This guidance is important, because it can substantially reduce the number of required preference queries and therefore minimize the human workload. We have also shown, that it is preferable to quickly update policies and directly validate the possibly improved version, instead of obtaining more preferences for a more exact policy evaluation. Furthermore, an early re-evaluation strategy is also preferable to maximizing a query selection criterion. This is possibly related to the high variance of query criteria, induced by a low number of preferences. Additionally, the uncertainty over the transition dynamics and the utility function should be considered independently as this allows to minimize both uncertainties efficiently. For creating preference queries, it seems most important to consider the expected change to the utility function or distribution. However, we have been able to also demonstrate the drawbacks of methods based on expected improvement. Considering the problems induced by human feedback, we have further substantiated the claim that sigmoid-like loss functions are well suited to deal with noise problems. Besides all the empirical results, we also presented a unified framework to PBRL where we point out the different problems and methods that should be considered. Especially interesting are the different techniques for defining the PBRL problem. It can be seen why utility-based methods define the current state-of-the-art, because they allow efficient generalization and sample reuse. However, utility-based methods are usually computationally expensive, because they require to solve two different optimization problems. Preference model-based methods can be computed faster, but usually require a higher number of preferences.

VII.4 Future Research

Future research in the field of PBRL should focus on improving the solution for the exploration/exploitation problem as well as the temporal credit assignment problem, in order to improve scalability. For improved exploration, we sketch an idea that directly incorporates the uncertainty of the utility function into the policy optimization problem in Section VII.4.1. Hence, a unified exploration criterion could be defined. For approximating a solution to the temporal credit assignment problem, it is required to focus on non-linear functions because linear functions are too restrictive, if not using domain knowledge. However, non-linear methods are subject to a high number of free parameters. In Section VII.4.2, we explain the general idea and give two directions for further improvements. In Section VII.4.3, we explain how to generalize RL further by incorporating different types of feedback into the same learning architecture. The aim is to increase the applicability by allowing a feedback signal of the expert's choosing.

VII.4.1 Improved Exploration

All available algorithms for PBRL either use undirected exploration of the transition dynamics to obtain preference samples or use directed exploration criteria, as described in Section III.3.4.a. Undirected exploration methods that consider the transition dynamics as well as the uncertainty of the preference feedback have not been evaluated. Bayesian methods provide us with uncertainty estimates for the learned utility function that could be incorporated into the sampling process. We use this information to select good preference queries, but still assume that samples from the undirected policy space exploration provide sufficient information. The policy search my converge to a suboptimal solution, if we can not obtain samples that allow us learn the optimal utility function. Hence, the uncertainty of the utility function should influence the stochasticity of the undirected exploration policy.

The basic idea is to use a Q-function that does not only define the expectation (II.2.4), but a probability distribution

$$\begin{aligned}\Pr(Q^\pi \mid s_0, a_0) &= \int \Pr(U \mid \boldsymbol{\tau}) \Pr^\pi(\boldsymbol{\tau}) d\boldsymbol{\tau},\\ &= \int_S \delta(s' \mid s_0, a_0) \Big(\Pr(U \mid s_0, a_0, s')\\ &\quad + \gamma \int_{A(s')} \pi(a' \mid s') \Pr(Q^\pi \mid s', a')\, \mathrm{d}a'\Big)\, \mathrm{d}s',\end{aligned} \tag{VII.4.1}$$

subject to a *utility distribution* $\Pr(U \mid \boldsymbol{\tau})$ or $\Pr(U \mid s, a, s')$. The utility distribution can be obtained with Bayesian inference, as explained in Section VI.2. The Q-distribution can be sampled with Monte Carlo methods or by adapting temporal difference learning to distributions (cf. Section II.2.3.a & II.2.3.b). Policy optimization can then be performed by obtaining n samples from

Algorithm 11 Preference-based Policy Improvement with Coupled Uncertainty

```
Require: current policy Q-distribution $\Pr(Q^{\pi_i} \mid s, a)$, trajectories $\Upsilon$, sample limit $n$
 1: for $j = 0$ to $n$ do
 2:     $\mathcal{Q} = \emptyset$
 3:     for $\forall(s, a)$ in $\Upsilon$ do                          ▷ For all observed samples
 4:         $Q(s, a) \sim \Pr(Q^{\pi_i} \mid s, a)$                  ▷ Draw sample from Q-Distribution
 5:         $\mathcal{Q} \leftarrow Q(s, a)$
 6:     end for
 7:     $\pi_{i+1,j}$ = sampleBasedPolicyImprovement($\mathcal{Q}$)   ▷ Compute improved policy
 8: end for
 9: $\pi_{i+1} = \prod_{j=0}^{n} \pi_{i+1,j}$                          ▷ Compute mixture
10: return improved policy $\pi_{i+1}$
```

$\Pr(Q^{\pi} \mid s, a)$ for each observed state-action sample, as shown in Algorithm 11. We have n sets of samples and can perform an independent policy optimization step on each of the sample sets to obtain n different, improved policies. The true resulting policy is then a mixture of all obtained policies. For policy improvement, any sample-based procedure can be used, such as the *relative entropy policy search* (REPS) variant introduced in Section VI.3. A single policy optimization step should introduce reduced exploration, compared to pure expectation based methods, as we obtain additional exploration based on the mixture. This exploration is then correlated with the uncertainty of the long-term expectation, the Q-function, and therefore also with the uncertainty of the utility function.

VII.4.2 Non-linear Reward-based Utility

When dealing with a reward-based utility, we obtain constraints for a trajectory utility $U(\boldsymbol{\tau})$, but want to compute a state-action utility $U(\boldsymbol{\tau}(t))$. Due to the Markov property (cf. Section II.2.1), we have to respect the constraint

$$U(\boldsymbol{\tau}) = \sum_{t=0}^{|\boldsymbol{\tau}|-1} U(s_t, a_t). \qquad \text{(VII.4.2)}$$

Hence, we need a mapping F from state-actions to trajectories that satisfies $\sum_{t=0}^{|\boldsymbol{\tau}|} U(s_t, a_t) = U(F(\boldsymbol{\tau}))$. In the case of a linear utility function, this is trivial

$$\begin{aligned} \sum_{t=0}^{|\boldsymbol{\tau}|} U(s_t, a_t) &= U(F(\boldsymbol{\tau})), \\ \sum_{t=0}^{|\boldsymbol{\tau}|} \boldsymbol{\theta}^T \boldsymbol{\phi}(s_t, a_t) &= \boldsymbol{\theta}^T F(\boldsymbol{\tau}), \\ \sum_{t=0}^{|\boldsymbol{\tau}|} \boldsymbol{\phi}(s_t, a_t) &= F(\boldsymbol{\tau}), \end{aligned} \tag{VII.4.3}$$

and we obtain the feature averages (III.3.16). However, the assumption of a linear utility function is rather restrictive. Even when achieving non-linearity by using kernels, as explained in Section II.3.2, we have to use a high number of basis functions to compute complex functions. This, in turn, results in a high-dimensional utility function, requiring a high number of examples for learning. However, we don't want to require the expert to define a high number of preferences. Hence, we prefer to keep the dimensionality low. This could be achieved by intelligent center selection methods that have already been developed in the area of RBF networks (Mao 2002). Alternatively, non-kernel-based, non-linear utility functions that are able to approximate the expert's evaluation criterion could be of interest. As an example, *deep neural networks* (DNNs) (cf. Section II.3.4) could be used as function approximators. However, this would require to compute F for non-linear U or to explicitly incorporate the constraint (VII.4.2).

VII.4.3 Multiple Feedback Methods

Currently, RL methods are usually restricted to one type of feedback, either numeric, ordinal, or one of the preference-based types, introduced in Section III.3.1. Learning from advice (Section III.2.2) allows to provide additional feedback, but not replacing the numeric rewards, only speeding up the learning process. Recently, Kupcsik et al. (2015) introduced the idea of allowing the expert to define preferences and/or trajectory return feedback. Neither feedback is required and it is also possible to use both forms simultaneously. They define a likelihood function

$$\Pr(\boldsymbol{\theta} \mid \mathscr{D}) \propto \Pr(\boldsymbol{\theta}) \prod_{i=1}^{|\zeta|} p_{\boldsymbol{\theta}}(\zeta_i) \prod_{j=1}^{|\Upsilon|} p_{\boldsymbol{\theta}}(\Upsilon_i), \tag{VII.4.4}$$

for computing the parameters $\boldsymbol{\theta}$ of a return-based utility function. $\mathscr{D}$ is the combined set of trajectory preferences ζ and expected trajectory return feedback Υ. In contrast to Definition III.3.13, it contains an additional likelihood term

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \mathcal{N}(\tilde{R} \,|\, \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{\tau}), \sigma_R), \tag{VII.4.5}$$

that determines the likelihood of the expected return $\tilde{R}$ for trajectory $\boldsymbol{\tau}$, given a variance parameter σ_R. The setting of Kupcsik et al. (2015) is restricted to parametric policies and uses return-based utility, as explained in Section III.3.3.b. However, the general idea is applicable to reward-based utility learning (cf. Section III.3.3.c) by using the method of feature averages (III.3.16). This idea enables us to define a likelihood function

$$p_{\boldsymbol{\theta}}(s, a) = \mathcal{N}(\tilde{r} \,|\, \boldsymbol{\theta}^T \boldsymbol{\phi}(s, a), \sigma_r), \tag{VII.4.6}$$

for expected rewards $\tilde{r}$ for a state-action pair $\{s, a\}$, as it is possible to evaluate the utility function based on trajectories as well as state-action pairs. For ordinal feedback, likelihood functions over state-actions or trajectories can be defined accordingly. Albert and Chib (1993) suggested to use the ordinal classes as unknown intervals on a continuous scale. Assuming the unknown intervals to be latent variables $\mathbf{l}$ and using an ordered probit regression model allows to compute the likelihood function

$$p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) = \prod_{i=0}^{|\tilde{O}|} \left(\Phi\left(l_{\tilde{O}_i} - \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{\tau})\right) - \Phi\left(l_{\tilde{O}_{i-1}} - \boldsymbol{\theta}^T \boldsymbol{\phi}(\boldsymbol{\tau})\right)\right) \exp\left(\mathbb{1}(\tilde{o} = \tilde{O}_i)\right) \Pr(\mathbf{l}), \tag{VII.4.7}$$

with $\tilde{o} \in \tilde{O}$ as the ordinal class of trajectory $\boldsymbol{\tau}$ and a suitable prior $\Pr(\mathbf{l})$. The function Φ is the c.d.f. with respect to the normal distribution and $\mathbb{1}(\tilde{o} = \tilde{O}_i)$ indicates if the given sequence is expected to belong to the i-th ordinal class. We can also use ordinal state-action feedback by defining the function relative to $\boldsymbol{\theta}^T \boldsymbol{\phi}(s, a)$, as in the reward case (VII.4.6).

Short-term action preferences, as mentioned in Section III.3.1.a, also define a preference between two state-action pairs. Hence, we can introduce

$$d(\boldsymbol{\theta}, (s_i, a_{i1}) \succ (s_i, a_{i2})) = \boldsymbol{\theta}^T \left(\boldsymbol{\phi}(s_i, a_{i1}) - \boldsymbol{\phi}(s_i, a_{i2})\right), \tag{VII.4.8}$$

comparable to the trajectory difference (III.3.11) and apply any preference loss from Table III.1.

What remains is the question how to define likelihood functions for state preferences, long-term preferences and advice. All three feedback types depend on an optimal policy. State-preferences can be phrased as $\forall a_{i2} \in A(s_{i2})\{(s_{i1}, \pi^*(s_{i1})) \succ (s_{i2}, a_{i2})\}$, as explained in

Section III.3.1.b. Long-term state and action preferences are trajectory preferences, as they depend on a policy that is followed after the encountered state or action. A fact already used by Fürnkranz et al. (2012). Learning from advice can be mapped to long-term preferences, as explained in Section III.2.2.a.

Appendix A

Domains

A.1 Acrobot

The acrobot domain is a two-link swing up problem with the goal to rise the tip over a certain level. Torque can only be applied at the second joint, as shown in Figure A.1.1. As parametrization, we use the setup of Sutton and Barto (1998), defined as

$$\begin{aligned} s &\in [\mathbb{R}; |\mathbb{R}| < 4\pi; \mathbb{R}; |\mathbb{R}| < 9\pi], \\ a &\in [-1, 0, 1], \end{aligned} \tag{A.1.1}$$

with the joint angles (ω_1, ω_2) and angular velocities $(\dot{\omega}_1, \dot{\omega}_2)$ of the two joints as the state space. The action is the torque that should be applied. The transition function

$$\begin{aligned} s' &= \left[\omega_1 + \dot{\omega}_1 t; \dot{\omega}_1 + \ddot{\omega}'_1 t; \omega_2 + \dot{\omega}'_2 t; \dot{\omega}_2 + \ddot{\omega}_2 t\right], \\ \dot{\omega}'_1 &= \dot{\omega}_1 + \ddot{\omega}_1 t, \\ \dot{\omega}'_2 &= \dot{\omega}_2 + \ddot{\omega}_2 t, \\ \ddot{\omega}_1 &= -d_1^{-1}(d_2 \ddot{\omega}_2 + d_3), \\ \ddot{\omega}_2 &= \left(1.25 - \frac{d_2^2}{d_1}\right)^{-1} \left(a + U(-0.2, 0.2) + \frac{d_2}{d_1}\omega_1 - 0.5\dot{\omega}_1{}^2 \sin\omega_2 - d_4\right), \\ d_1 &= \cos\omega_2 + 3.5, \\ d_2 &= 0.5\cos\omega_2 + 1.25, \\ d_3 &= -0.5\dot{\omega}_2{}^2 \sin\omega_2 - \dot{\omega}_1\dot{\omega}_2 \sin\omega_2 + 1.5g\cos(\omega_1 - \pi/2) + d_4, \\ d_4 &= 0.5g\cos(\omega_1 + \omega_2 - \pi/2), \end{aligned} \tag{A.1.2}$$

is a parametrization of the dynamical system with unit length, mass and moments of inertia for both links. $g = 9.8m/s^2$ is the gravity and $t = 0.05s$ the duration of a step. $U(-0.2, 0.2)$ is a uniform distributed random noise for the action. The reward is -1 for all states, but the episodes

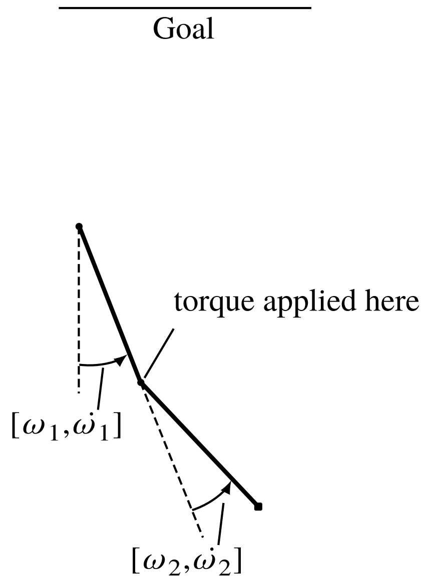


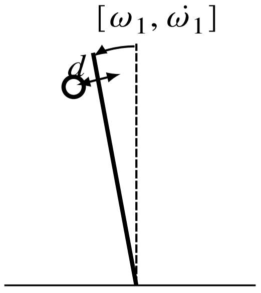
Figure A.1.1.: The acrobot domain

ends once the goal is reached. The initial state is $\omega_1 = \dot{\omega}_1 = \omega_2 = \dot{\omega}_2 = 0$ and the goal is reached if the 2nd joint tip is above the goal line $-(\cos\omega_1 + \sin(\pi/2 - \omega_1 - \omega_2)) > 1$. For obtaining a tabular representation of the $\tilde{Q}$ function, we discretize the state space into 10 equal width bins for each dimension, obtaining 10000 features. Episodes are created with 500 time-steps.

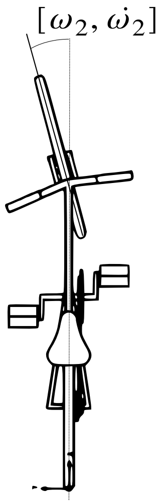
A.2 Bicycle Balance

The task in the bicycle balance domain is to keep a bicycle upright and was introduced by Randløv and Alstrøm (1998). The agent is able to displace the center of mass by moving d cm to the side of the saddle, relative to current angle of the bike, as shown in Figure A.2.2a. It is also possible to steer the front tire by applying torque τ, influencing the angle ω_2 in Figure A.2.2b. The state space is periodic and defined by the angles and velocities of angle ω_1 and ω_2, resulting in

$$\begin{aligned} s &\in [|\mathbb{R}| < \pi; \mathbb{R}; |\mathbb{R}| < \pi; \mathbb{R}], \\ a &\in [|\mathbb{R}| < 0.02; |\mathbb{R}| < 2]. \end{aligned} \tag{A.2.3}$$



(a) The bicycle from behind with a circle as center of mass



(b) The bicycle from above, with ω_2 as the front tyre angle

The first action dimension is the displacement d and the second entry is the torque τ. The state and action spaces are continuous, as used by Akrour et al. (2014), and the initial states is the upright position. The transition dynamics are governed by

$$\begin{aligned}
s' =& \left[\omega_1 + \dot{\omega}'_1 t; \dot{\omega}_1 + \ddot{\omega}_2 t; \omega_2 + \dot{\omega}'_2 t; \dot{\omega}_2 + \ddot{\omega}_2 t\right], \\
\dot{\omega}'_1 =& \dot{\omega}_1 + \ddot{\omega}_1 t, \\
\dot{\omega}'_2 =& \dot{\omega}_2 + \ddot{\omega}_2 t, \\
\ddot{\omega}_1 =& \frac{1}{149.69} \left(70.5 g \sin\left(\phi\right)\right) - \cos\left(\phi\right) \left(1.60556 \dot{\omega}_2 + \operatorname{sign}\left(\omega_2\right)\right. \\
& \left.\cdot 2.\overline{7} \left(\frac{0.578}{\frac{1.11}{|\sin \omega_2|}} + \frac{0.578}{\frac{1.11}{|\tan \omega_2|}} + \frac{70.5}{\left(0.2025 + \frac{1.2321}{(\tan \omega_2)^2}\right)^{\frac{1}{2}}} \right)\right), \\
\ddot{\omega}_2 =& \frac{\tau - 2.40833 \dot{\omega}_1}{0.09826}, \\
\phi =& \omega + \arctan\left(\frac{d}{0.94}\right).
\end{aligned} \tag{A.2.4}$$

The dynamics assume a forward motion of $10 km/h$, $g = 9.8 m/s^2$ and $t = 0.01 s$ to approximate the changes to steering and bike angles. The reward is the squared angel deviation form the upright position $r(s) = 1 - \omega_1^2$, also used by Akrour et al. (2014). Trajectories end in case the bike falls over ($|\omega_1| > \frac{\pi}{15}$). For an explanation, including the computation of all constants, we refer the reader to Randløv and Alstrøm (1998). We trained on episodes with 30000 time-steps with the C implementation by Lagoudakis and Parr (2003a).[51]

A.3 Cart Pole

The cart pole task is also known as inverted pendulum and was defined by Wang et al. (1996). The goal is to balance a pole on a cart and keep it in an upright position by moving the cart left and right. The state space is defined by the angle ω and velocity $\dot{\omega}$ of the pole with three discrete actions defining the force that can be applied to the cart:

$$\begin{aligned} &s \in \mathbb{R}^2, \\ &a \in [-50, 0, 50]. \end{aligned} \tag{A.3.5}$$

The system, as shown in Figure A.3.3, is governed by the transition function

$$s' = \left[\omega + \dot{\omega}'; \dot{\omega} + \frac{g \sin s - 0.1\dot{\omega}^2 \sin(2\omega)/2 - 0.1 cos(\omega)\,(a + U(-10, 10))}{2/3 + 0.1 \cos^2 s} \right], \tag{A.3.6}$$

witch is the physical system as parameterized by Dimitrakakis and Lagoudakis (2008). $g = 9.8m/s^2$ defines the gravity and $U(-10, 10)$ is a transition noise. The reward is -1 in case the pole falls over and 0 otherwise. The tabular representation is again based on 10 bins per dimension, resulting in 100 features. The start state is the upright position and the goal is to keep the pole above the horizontal line.

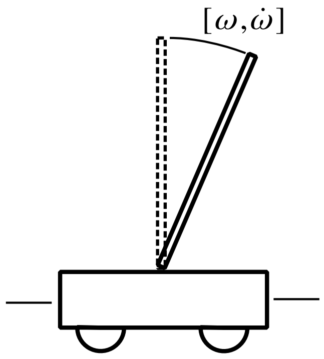


Figure A.3.3.: The cart pole domain

[51] https://www.cs.duke.edu/research/AI/LSPI/

$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$	1
$\frac{1}{32}$	$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$
$\frac{1}{64}$	$\frac{1}{32}$	▨ $\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{4}$
$\frac{1}{128}$	$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{16}$	$\frac{1}{8}$
$\frac{1}{256}$	$\frac{1}{128}$	$\frac{1}{64}$	$\frac{1}{32}$	$\frac{1}{16}$

Table A.1.: The grid world domain

A.4 Grid world

The grid world variant defined by Akrour et al. (2014), is a simple 5×5 grid with different rewards per field, as shown by Table A.1. The grey field is the starting position. The state space s has 25 dimensions, one binary entry per field. The action space a has one dimension with five possible actions: {UP, DOWN, LEFT, RIGHT, STAY}. The transition function is non deterministic, with a chance of 50% to perform the selected action. Each trajectory has a length of 300 steps. This setting is simple from a *reinforcement learning* (RL) point of view, if the rewards are known. However, as the rewards are fairly similar, it is difficult to compute a exact utility function when only obtaining trajectory preferences.

A.5 Mountain Car

The problem in the mountain car domain is to drive an underpowered car from a valley up a mountain, as shown in Figure A.5.4. As the cars engine is not powerful enough to achieve this task directly, it is required to obtain additional force by using gravity. Hence, multiple runs through the valley are required. The environment is as defined by Sutton and Barto (1998), with a 2d state space and a 1d action space

$$\begin{aligned} s &\in [-1.2 < \mathbb{R} < 0.6; |\mathbb{R}| < 0.07], \\ a &\in [-1, 0, 1], \end{aligned} \tag{A.5.7}$$

where s_1 is the position, s_2 the velocity and the actions are full throttle, zero throttle and full reverse. The transition function is defined as

$$s' = \left[s_1 + s_2'; s_2 + 0.001\,(a + U(-0.2, 0.2)) - 0.0023\cos(3s_1)\right], \tag{A.5.8}$$

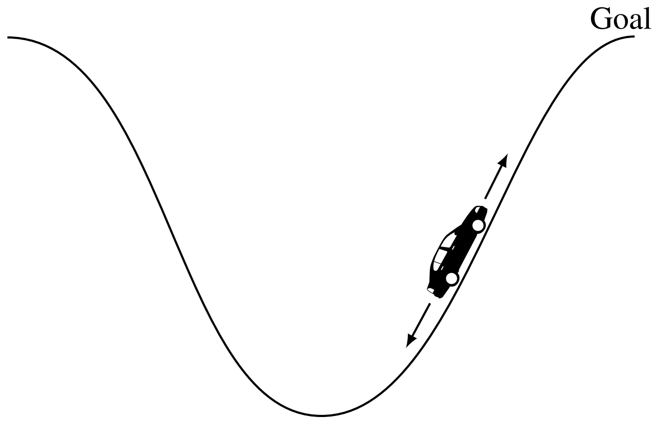


Figure A.5.4.: The mountain car domain

where the cosine term describes the slope and $U(-0.2, 0.2)$ is a uniform distributed random noise for increasing the difficulty. The reward is 0 for all states, except the goal state where a reward of 1 is obtained. For obtaining a tabular representation of the $\tilde{Q}$ function, we discretize the state space into 10 equal width bins for each dimension, obtaining 100 features. Start states are randomly sampled from from the complete state space and the goal is reached when $s_1 > 0.5$. The horizon for this setting is set to 500.

A.6 Swing Up

The swing up domain combines the problems of the mountain car and the cart pole domain. As in the cart pole domain, the task is to bring a pendulum to an upright position, but it hangs down in the initial position, as shown in Figure A.6.5. The task was originally defined by Åström and Furuta (2000), but they neglected the friction of the system. As in the mountain car domain, the torque that can be applied to the pendulum is not sufficient to bring it up directly, hence a swinging movement is required to gain additional momentum, using the gravitation. The state space is the angel (periodic) and its velocity

$$\begin{aligned} s &\in [|\mathbb{R}| < \pi; \mathbb{R}], \\ a &\in |\mathbb{R}| < 30. \end{aligned} \tag{A.6.9}$$

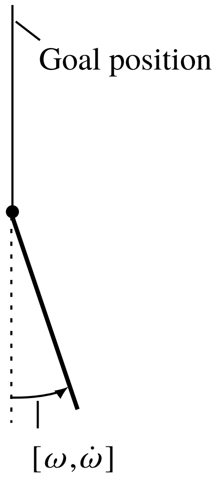


Figure A.6.5.: The swing up domain

The action a is the torque applied to the angle. The transition dynamics

$$\begin{aligned} s' &= \left[\omega + \dot{\omega}'t; \dot{\omega} + \ddot{\omega}t\right], \\ \dot{\omega}' &= \dot{\omega} + \ddot{\omega}t, \\ \ddot{\omega} &= \frac{5g}{0.8\overline{3}}\sin(\omega) + \frac{a}{0.8\overline{3}} - 0.3\dot{\omega}, \end{aligned} \tag{A.6.10}$$

describe the system with an inertia of $0.8\overline{3}$ and a friction of 0.3. $g = 9.8m/s^2$ is again the gravity and $t = 1e^{-4}$ the time discretization. The angle of the initial, hanging position is sampled randomly from $\omega = \pi + -0.2$. The reward is the squared angle deviation from the upright position $r(s) = -\omega^2$. Simulations are run for 60 time steps.

Bibliography

Adelson-Velskii, G. M., V. L. Arlazarov, A. R. Bitman, A. A. Zhivotovskii, and A. V. Uskov. 1970. “Programming a Computer to Play Chess.” *Russian Mathematical Surveys* 25 (2): 221–262. (Cit. on p. 115).

Agarwal, A., O. Dekel, and L. Xiao. 2010. “Optimal Algorithms for Online Convex Optimization with Multi-Point Bandit Feedback.” In *Proceedings of the 23rd Conference on Learning Theory (COLT-10)*, 28–40. (Cit. on p. 51).

Aiolli, F., and A. Sperduti. 2010. “A Preference Optimization based Unifying Framework for Supervised Learning Problems.” In Fürnkranz and Hüllermeier 2010a, 19–42. (Cit. on p. 40).

Akrour, R., M. Schoenauer, and M. Sebag. 2011. “Preference-Based Policy Learning.” In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-11)*, 6911:12–27. LNCS. (Cit. on pp. 61, 65, 69–74, 78, 141, 157, 158).

– . 2012. “APRIL: Active Preference Learning-Based Reinforcement Learning.” In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD-12)*, 7524:116–131. LNCS. (Cit. on pp. 61, 65, 69–74, 78, 133, 135, 141, 157, 158).

– . 2013. “Interactive Robot Education.” In *Proceedings of the ECML/PKDD Workshop on Reinforcement Learning with Generalized Feedback: Beyond Numeric Rewards*. (Cit. on pp. 61, 66, 158).

Akrour, R., M. Schoenauer, M. Sebag, and J.-C. Souplet. 2014. “Programming by Feedback.” In *Proceedings of the 31nd International Conference on Machine Learning (ICML-14)*, 32:1503–1511. (Cit. on pp. 61, 62, 66, 69–71, 73–76, 78, 129, 132, 133, 135, 143, 145, 149, 150, 157, 158, 167, 169).

Albert, J. H., and S. Chib. 1993. “Bayesian analysis of binary and polychotomous response data.” *Journal of the American Statistical Association* 88 (422): 669–679. (Cit. on p. 163).

Amodei, D., C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. 2016. “Concrete Problems in AI Safety.” *CoRR* abs/1606.06565. (Cit. on p. 3).

Andrieu, C., N. de Freitas, A. Doucet, and M. I. Jordan. 2003. “An Introduction to MCMC for Machine Learning.” *Machine Learning* 50 (1-2): 5–43. (Cit. on pp. 55, 62, 74, 78).

Åström, K. J., and K. Furuta. 2000. “Swinging up a pendulum by energy control.” *Automatica* 36 (2): 287–295. (Cit. on p. 170).

Audibert, J.-Y., and S. Bubeck. 2009. “Minimax Policies for Adversarial and Stochastic Bandits.” In *Proceedings of the 22nd Conference on Learning Theory (COLT-09)*, 773–818. (Cit. on p. 92).

Auer, P., N. Cesa-Bianchi, and P. Fischer. 2002. “Finite-time Analysis of the Multiarmed Bandit Problem.” *Machine Learning* 47 (2-3): 235–256. (Cit. on pp. 94, 136).

Auer, P., N. Cesa-Bianchi, Y. Freund, and R. E. Schapire. 1995. “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)*, 322–331. (Cit. on pp. 69, 92).

– . 2003. “The Nonstochastic Multiarmed Bandit Problem.” *SIAM Journal on Computing* 32 (1): 48–77. (Cit. on p. 92).

Auger, A. 2005. “Convergence results for the $(1,\lambda)$-SA-ES using the theory of ϕ-irreducible Markov chains.” *Theoretical Computer Science* 334 (1–3): 35–69. (Cit. on pp. 74, 78).

Bagnell, J. A., and J. G. Schneider. 2003. “Covariant Policy Search.” In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 1019–1024. (Cit. on p. 31).

Barto, A. G. 2013. “Intrinsic Motivation and Reinforcement Learning.” In *Intrinsically Motivated Learning in Natural and Artificial Systems*, 17–47. Springer Berlin Heidelberg. (Cit. on p. 4).

Baxter, J., A. Tridgell, and L. Weaver. 1998. “Knightcap: A chess program that learns by combining TD(λ) with game-tree search.” In *Proceedings of the 15th International Conference on Machine Learning, (ICML-98)*, 28–36. (Cit. on p. 112).

– . 2000. “Learning to play chess using temporal differences.” *Machine Learning* 40 (3): 243–263. (Cit. on pp. 105, 115).

Beal, D. F., and M. C. Smith. 1997. “Learning piece values using temporal differences.” *Journal of The International Computer Chess Association* 20 (3): 147–151. (Cit. on p. 112).

– . 2000. “Temporal difference learning for heuristic search and game playing.” *Information Sciences* 122 (1): 3–21. (Cit. on p. 115).

– . 2001. “Temporal difference learning applied to game playing and the results of application to shogi.” *Theoretical Computer Science* 252 (1): 105–119. (Cit. on pp. 105, 115).

Bellman, R. 1957. *Dynamic Programming*. 1st ed. Princeton University Press. (Cit. on p. 22).

Berenson, D., S. Srinivasa, D. Ferguson, and J. Kuffner Jr. 2009. “Manipulation Planning on Constraint Manifolds.” In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-09)*, 625–632. (Cit. on pp. 76, 78).

Bianchini, M., and F. Scarselli. 2014. “On the Complexity of Neural Network Classifiers: A Comparison Between Shallow and Deep Architectures.” *IEEE Transactions on Neural Networks and Learning Systems* 25 (8): 1553–1565. (Cit. on p. 29).

Bishop, C. M. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press. (Cit. on pp. 58, 78).

– . 2006. *Pattern Recognition and Machine Learning*. Springer. (Cit. on pp. 24–28).

Bowling, M., J. Fürnkranz, T. Graepel, and R. Musick. 2006. “Machine learning and games.” *Machine learning* 63 (3): 211–215. (Cit. on p. 105).

Boyan, J. 1999. “Least-Squares Temporal Difference Learning.” In *Proceedings of the 16th International Conference on Machine Learning (ICML-99)*, 49–56. (Cit. on pp. 33, 69, 75, 78, 137).

Brafman, R. I., and M. Tennenholtz. 2001. “R-MAX - A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning.” In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 953–958. (Cit. on pp. 15, 32).

Busa-Fekete, R., and E. Hüllermeier. 2014. “A Survey of Preference-based Online Learning with Bandit Algorithms.” In *Proceedings of the 25th Algorithmic Learning Theory International Conference (ALT-14)*, 8776:18–39. LNCS. (Cit. on p. 57).

Busa-Fekete, R., B. Szörényi, P. Weng, W. Cheng, and E. Hüllermeier. 2013. “Preference-based Evolutionary Direct Policy Search.” In *Proceedings of the ICRA Workshop on Autonomous Learning*. (Cit. on pp. 51, 54, 56, 57, 68, 70, 74, 76, 78).

– . 2014. “Preference-based reinforcement learning: evolutionary direct policy search using a preference-based racing algorithm.” *Machine Learning* 97 (3): 327–351. (Cit. on pp. 51, 54, 56, 57, 68–70, 74, 76, 78).

Campbell, M., A. J. Hoane, and F.-H. Hsu. 2002. “Deep blue.” *Artificial intelligence* 134 (1): 57–83. (Cit. on p. 114).

Christiano, P., J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. 2017. “Deep Reinforcement Learning from Human Preferences.” *CoRR* abs/1706.03741. (Cit. on pp. 61, 64, 66, 68, 69, 71, 74, 75, 78).

Chu, W., and Z. Ghahramani. 2005. “Preference learning with Gaussian processes.” In *Proceedings of the 22nd International Conference on Machine Learning (ICML-05)*, 137–144. (Cit. on p. 64).

Coughlin, P. J. 2008. *Probabilistic Voting Theory*. Cambridge University Press. (Cit. on p. 6).

Daniel, C., G. Neumann, and J. Peters. 2012. "Hierarchical Relative Entropy Policy Search." In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, 273–281. (Cit. on p. 137).

– . 2013. "Learning Sequential Motor Tasks." In *Proceedings of 2013 IEEE International Conference on Robotics and Automation (ICRA 2013)*. (Cit. on p. 32).

Daniel, C., M. Viering, J. Metz, O. Kroemer, and J. Peters. 2014. "Active Reward Learning." In *Proceedings of Robotics: Science & Systems X (R:SS-14)*. (Cit. on p. 49).

Deisenroth, M. P., G. Neumann, and J. Peters. 2013. "A Survey on Policy Search for Robotics." *Foundations and Trends in Robotics* 2:1–142. (Cit. on pp. 13, 20, 21, 138, 157).

Deisenroth, M. P., and C. E. Rasmussen. 2011. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search." In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 465–472. (Cit. on p. 36).

Devroye, L. 1986. *Non-uniform random variate generation*. Springer-Verlag. (Cit. on p. 18).

Dimitrakakis, C., and M. G. Lagoudakis. 2008. "Rollout sampling approximate policy iteration." *Machine Learning* 72 (3): 157–171. (Cit. on pp. 69, 168).

Bayesian Multitask Inverse Reinforcement Learning. 2011. Vol. 7188. LNCS. Springer. (Cit. on p. 50).

Droste, S., and J. Fürnkranz. 2008. "Learning the Piece Values for Three Chess Variants." *International Computer Games Association Journal* 31 (4): 209–233. (Cit. on pp. 105, 115).

Duchi, J., L. W. Mackey, and M. I. Jordan. 2010. "On the Consistency of Ranking Algorithms." In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 327–334. (Cit. on p. 47).

Duda, R. O., and P. E. Hart. 1973. *Pattern Recognition and Scene Analysis*. John Wiley / Sons. (Cit. on p. 65).

Edwards, S. J. 1994. *Portable game notation*. Accessed on 14.06.2012. `http://www6.chessclub.com/help/PGN-spec`. (Cit. on pp. 104, 106).

Elo, A. E. 1978. *The Rating of Chessplayers, Past and Present*. 2nd. Arco. (Cit. on p. 118).

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. 2008. "LIBLINEAR: A Library for Large Linear Classification." *Journal of Machine Learning Research* 9:1871–1874. (Cit. on p. 116).

Faust, A., H.-T. Chiang, N. Rackley, and L. Tapia. 2016. “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)*, 484–490. (Cit. on p. 49).

Flach, P. 2012. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press. (Cit. on p. 41).

Freund, Y., H. S. Seung, E. Shamir, and N. Tishby. 1997. “Selective Sampling using the Query by Committee Algorithm.” *Machine Learning* 28:133–168. (Cit. on p. 141).

Friedman, M., and L. J. Savage. 1952. “The Expected-Utility Hypothesis and the Measurability of Utility.” *Journal of Political Economy* 60. (Cit. on p. 59).

Fudenberg, D. 1998. *The Theory of Learning in Games*. MIT Press. (Cit. on p. 6).

Fürnkranz, J. 1996. “Machine Learning in Computer Chess: The Next Generation.” *International Computer Chess Association Journal* 19 (3): 147–161. (Cit. on p. 105).

– . 2010. “Machine Learning and Game Playing.” In *Encyclopedia of Machine Learning*, 633–637. Springer-Verlag. (Cit. on p. 105).

Fürnkranz, J., and E. Hüllermeier, eds. 2010a. *Preference Learning*. Springer-Verlag. (Cit. on p. 6).

– . 2010b. “Preference Learning: An Introduction.” In Fürnkranz and Hüllermeier 2010a, 1–17. (Cit. on pp. 38, 39).

– . 2010c. “Preference Learning and Ranking by Pairwise Comparison.” In Fürnkranz and Hüllermeier 2010a, 65–82. (Cit. on p. 41).

Fürnkranz, J., E. Hüllermeier, W. Cheng, and S.-H. Park. 2012. “Preference-based Reinforcement Learning: A Formal Framework and a Policy Iteration Algorithm.” Special Issue of Selected Papers from ECML/PKDD-11, *Machine Learning* 89 (1-2): 123–156. (Cit. on pp. 9, 52, 57, 58, 67, 68, 75, 76, 78, 80–82, 97, 164).

Fürnkranz, J., E. Hüllermeier, and S. Vanderlooy. 2009. “Binary Decomposition Methods for Multipartite Ranking.” In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD-09)*, 1:359–374. 2009. (Cit. on pp. 39, 41).

Fürnkranz, J., and M. Kubat, eds. 2001. *Machines that Learn to Play Games*. Vol. 8. Advances in Computation: Theory and Practice. Nova Science Publishers. (Cit. on p. 105).

Geramifard, A., T. J. Walsh, S. Tellex, N. Roy, and J. P. How. 2013. “A Tutorial on Linear Function Approximators for Dynamic Programming and Reinforcement Learning.” *Foundations and Trends in Machine Learning* 6 (4): 375–451. (Cit. on p. 134).

Ghavamzadeh, M., A. Lazaric, O. Maillard, and R. Munos. 2010. "LSTD with Random Projections." In *Advances in Neural Information Processing Systems 23 (NIPS-10)*, 721–729. (Cit. on p. 35).

Ghory, I. 2004. "Reinforcement learning in board games." *Department of Computer Science, University of Bristol, Tech. Rep*. (Cit. on p. 105).

Golovin, N., and E. Rahm. 2004. "Reinforcement learning architecture for Web recommendations." In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC-04)*, vol. 1, 398–402 Vol.1. (Cit. on p. 5).

Gomboc, D., T. A. Marsland, and M. Buro. 2004. "Evaluation Function Tuning via Ordinal Correlation." In *Advances in Computer Games*, 135:1–18. IFIP - The International Federation for Information Processing. Springer US. (Cit. on pp. 105, 113).

Griffith, S., K. Subramanian, J. Scholz, C. Isbell, and A. L. Thomaz. 2013. "Policy Shaping: Integrating Human Feedback with Reinforcement Learning." In *Advances in Neural Information Processing Systems 26 (NIPS-13)*, 2625–2633. (Cit. on p. 49).

Gritsenko, A., and D. Berenson. 2014. "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*, 1:48–6. 41.7. (Cit. on pp. 63–65, 68, 75, 76, 78).

Grünewälder, S., G. Lever, L. Baldassarre, M. Pontil, and A. Gretton. 2012. "Modelling transition dynamics in MDPs with RKHS embeddings." In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, 535–542. (Cit. on p. 76).

Gu, S., T. P. Lillicrap, I. Sutskever, and S. Levine. 2016. "Continuous Deep Q-Learning with Model-based Acceleration." In *Proceedings of the 33nd International Conference on Machine Learning, (ICML-16)*, 2829–2838. (Cit. on p. 37).

Hansen, N., and A. Ostermeier. 2001. "Completely derandomized self-adaptation in evolution strategies." *Evolutionary Computation* 9 (2): 159–195. (Cit. on p. 143).

Hansen, N., and S. Kern. 2004. "Evaluating the CMA Evolution Strategy on Multimodal Test Functions." In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN-04)*, 3242:282–291. LNCS. (Cit. on pp. 69, 78).

Hastie, T., and R. Tibshirani. 1997. "Classification by Pairwise Coupling." In *Advances in Neural Information Processing Systems 10 (NIPS-97)*, 10:507–513. (Cit. on p. 86).

Heidrich-Meisner, V., and C. Igel. 2009. "Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search." In *Proceedings of the 26th International Conference on Machine Learning (ICML-09)*, 401–408. (Cit. on p. 57).

Herbrich, R., T. Graepel, and K. Obermayer. 1999. “Support vector learning for ordinal regression.” In *Proceedings of the 9th International Conference on Artificial Neural Networks (ICANN-99)*, 1:97–102. (Cit. on pp. 61, 78).

Hinton, G. E., and R. R. Salakhutdinov. 2006. “Reducing the Dimensionality of Data with Neural Networks.” *Science* 313 (5786): 504–507. (Cit. on p. 29).

Hoeffding, W. 1963. “Probability Inequalities for Sums of Bounded Random Variables.” *Journal of the American Statistical Association* 58 (301): 13–30. (Cit. on p. 69).

Hoffman, M., A. Lazaric, M. Ghavamzadeh, and R. Munos. 2012. “Regularized Least Squares Temporal Difference Learning with Nested l2 and l1 Penalization.” In *Recent Advances in Reinforcement Learning*, 7188:102–114. LNCS. Springer. (Cit. on pp. 21, 33, 35, 36).

Howard, R. A. 1988. “Decision Analysis: Practice and Promise.” *Management Science* 34 (6): 679–695. (Cit. on p. 68).

Hüllermeier, E., J. Fürnkranz, W. Cheng, and K. Brinker. 2008. “Label Ranking by Learning Pairwise Preferences.” *Artificial Intelligence* 172 (16–17): 1897–1916. (Cit. on pp. 39, 41, 57).

Jain, A., B. Wojcik, T. Joachims, and A. Saxena. 2013. “Learning Trajectory Preferences for Manipulators via Iterative Improvement.” In *Advances in Neural Information Processing Systems 26 (NIPS-13)*, 575–583. (Cit. on pp. 62, 65, 68–70, 75, 76, 78).

Jain, A., S. Sharma, T. Joachims, and A. Saxena. 2015. “Learning preferences for manipulation tasks from online coactive feedback.” *International Journal of Robotics Research* 34 (10): 1296–1313. (Cit. on pp. 62, 65, 68–70, 75, 76, 78).

Joachims, T. 2002. “Optimizing Search Engines Using Clickthrough Data.” In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02)*, 133–142. (Cit. on pp. 40, 61, 78, 112, 116).

Jong, N. K., and P. Stone. 1976. “Decisions with Multiple Objectives: Preferences and Value Tradeoffs.” In *Proceedings of the ICML-06 Workshop on Kernel Methods in Reinforcement Learning*. (Cit. on p. 68).

– . 2007. “Model-Based Function Approximation for Reinforcement Learning.” In *The 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-07)*. (Cit. on p. 33).

Kakade, S. 2001. “A Natural Policy Gradient.” In *Advances in Neural Information Processing Systems 14 (NIPS-01)*, 1531–1538. (Cit. on p. 31).

Kamishima, T., H. Kazawa, and S. Akaho. 2010. “A Survey and Empirical Comparison of Object Ranking Methods.” In Fürnkranz and Hüllermeier 2010a, 181–201. (Cit. on pp. 38, 47).

Knox, W. B., and P. Stone. 2009. “Interactively Shaping Agents via Human Reinforcement: The TAMER Framework.” In *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP-09)*, 9–16. (Cit. on p. 50).

– . 2010. “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)*, 5–12. (Cit. on p. 49).

– . 2012. “Reinforcement learning from simultaneous human and MDP reward.” In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-12)*, 475–482. (Cit. on pp. 49, 52).

Kober, J., K. Mulling, O. Kroemer, C. H. Lampert, B. Schölkopf, and J. Peters. 2010. “Movement templates for learning of hitting and batting.” In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-10)*, 853–858. (Cit. on p. 139).

Kober, J., J. A. Bagnell, and J. Peters. 2013. “Reinforcement learning in robotics: A survey.” *The International Journal of Robotics Research* 32 (11): 1238–1274. (Cit. on p. 56).

Kreps, D. 1988. *Notes on the theory of choice*. Underground classics in economics. Westview Press. (Cit. on pp. 37, 38).

Kuleshov, V., and D. Precup. 2014. “Algorithms for the multi-armed bandit problem.” *CoRR* abs/1402.6028. (Cit. on p. 82).

Kupcsik, A., M. P. Deisenroth, J. Peters, A. P. Loh, P. Vadakkepat, and G. Neumann. 2014. “Model-based contextual policy search for data-efficient generalization of robot skills.” *Artificial Intelligence* 247:415–439. (Cit. on pp. 32, 69, 74, 78).

Kupcsik, A., M. P. Deisenroth, J. Peters, and G. Neumann. 2013. “Data-Efficient Generalization of Robot Skills with Contextual Policy Search.” In *Proceedings of the 27th National Conference on Artificial Intelligence (AAAI-13)*. (Cit. on pp. 137, 138).

Kupcsik, A., D. Hsu, and W. S. Lee. 2015. “Learning Dynamic Robot-to-Human Object Handover from Human Feedback.” In *Proceedings of the 17th International Symposium on Robotics Research (ISRR-15)*. (Cit. on pp. 49, 61, 62, 64, 65, 68–70, 74, 77, 78, 158, 162, 163).

Lagoudakis, M. G., and R. Parr. 2003a. “Least-Squares Policy Iteration.” *Journal of Machine Learning Research* 4:1107–1149. (Cit. on pp. 3, 70, 137, 168).

– . 2003b. “Reinforcement Learning as Classification: Leveraging Modern Classifiers.” In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 424–431. (Cit. on p. 81).

Langville, A. M., and C. D. Meyer. 2012. *Who’s #1? The Science of Rating and Ranking*. Princeton University Press. (Cit. on p. 6).

LaValle, S. M., and J. Kuffner Jr. 1999. “Randomized kinodynamic planning.” In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-99)*, 473–479. (Cit. on pp. 69, 76, 78).

Likhachev, M., G. Gordon, and S. Thrun. 2003. “ARA*: Anytime A* Search with Provable Bounds on Sub-Optimality.” In *Advances in Neural Information Processing Systems 16 (NIPS-03)*, 767–774. (Cit. on pp. 70, 76, 78).

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2015. “Continuous control with deep reinforcement learning.” *CoRR* abs/1509.02971. (Cit. on p. 37).

Liu, C., X. Xu, and D. Hu. 2015. “Multiobjective Reinforcement Learning: A Comprehensive Overview.” *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 45 (3): 385–398. (Cit. on p. 5).

Lu, C. X., Z. Y. Sun, Z. Z. Shi, and B. X. Cao. 2016. “Using Emotions as Intrinsic Motivation to Accelerate Classic Reinforcement Learning.” In *Proceedings of the International Conference on Information System and Artificial Intelligence (ISAI-16)*, 332–337. (Cit. on p. 4).

Maclin, R., J. W. Shavlik, L. Torrey, T. Walker, and E. W. Wild. 2005. “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)*, 819–824. (Cit. on p. 49).

Mannor, S., D. Simester, P. Sun, and J. N. Tsitsiklis. 2004. “Bias and Variance in Value Function Estimation.” In *Proceedings of the 21th International Conference on Machine Learning (ICML-04)*, 72–. (Cit. on p. 30).

Mao, K. Z. 2002. “RBF Neural Network Center Selection Based on Fisher Ratio Class Separability Measure.” *IEEE Transactions on Neural Networks* 13 (5): 1211–1217. (Cit. on p. 162).

Marden, J. I. 1995. *Analyzing and Modeling Rank Data*. CRC Press. (Cit. on p. 6).

Mayeur, B., R. Akrour, and M. Sebag. 2014. “Direct Value Learning: a Rank-Invariant Approach to Reinforcement Learning.” In *Proceedings of the NIPS Workshop on Autonomously Learning Robots*. (Cit. on p. 50).

Meunier, D., Y. Deguchi, R. Akrour, E. Suzuki, M. Schoenauer, and M. Sebag. 2012. “Direct Value Learning: A Preference-based Approach to Reinforcement Learning.” In *Proceedings of the ECAI Workshop on Preference Learning: Problems and Applications in AI*, 42–47. (Cit. on p. 50).

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. 2016. “Asynchronous Methods for Deep Reinforcement Learning.” In *Proceedings of the 33rd International Conference on Machine Learning (ICML-16)*, 1928–1937. Proceedings of Machine Learning Research. (Cit. on pp. 69, 78).

Mnih, V., et al. 2015. “Human-level control through deep reinforcement learning.” *Nature* 518 (7540): 529–533. (Cit. on p. 37).

Murray, I., R. P. Adams, and D. J. C. MacKay. 2010. “Elliptical slice sampling.” In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS-10)*, 9:541–548. (Cit. on pp. 62, 78, 135).

Ng, A. Y., D. Harada, and S. J. Russell. 1999. “Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping.” In *Proceedings of the 16th International Conference on Machine Learning (ICML-99)*, 278–287. (Cit. on p. 3).

Ng, A. Y., and S. J. Russell. 2000. “Algorithms for inverse reinforcement learning.” In *Proceedings of the 17th International Conference on Machine Learning (ICML-00)*, 663–670. (Cit. on pp. 50, 61, 66, 134, 135).

Nguyen-Tuong, D., and J. Peters. 2011. “Model learning for robot control: a survey.” *Cognitive Processing* 12 (4): 319–340. (Cit. on p. 76).

Paulsen, P., and J. Fürnkranz. 2010. “A Moderately Successful Attempt to Train Chess Evaluation Functions of Different Strengths.” In *Proceedings of the ICML-10 Workshop on Machine Learning and Games*. (Cit. on p. 114).

Peters, J., K. Muelling, and Y. Altun. 2010. “Relative Entropy Policy Search.” In *Proceedings of the 24th National Conference on Artificial Intelligence (AAAI-10)*, 1607–1612. (Cit. on pp. 31, 69, 75, 78, 133, 136).

Price, D., S. Knerr, L. Personnaz, and G. Dreyfus. 1994. “Pairwise Neural Network Classifiers with Probabilistic Outputs.” In *Advances in Neural Information Processing Systems 7 (NIPS-94)*, 7:1109–1116. (Cit. on p. 86).

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann. (Cit. on pp. 64, 78).

Randløv, J., and P. Alstrøm. 1998. “Learning to Drive a Bicycle Using Reinforcement Learning and Shaping.” In *Proceedings of the 15th International Conference on Machine Learning (ICML-98)*, 463–471. (Cit. on pp. 166, 168).

Rao, C. R. 1945. “Information and accuracy attainable in the estimation of statistical parameters.” *Bulletin of Calcutta Mathematical Society* 37:81–91. (Cit. on p. 31).

Rao, V. R., P. E. Green, and J. Wind. 2007. *Applied Conjoint Analysis*. SAGE Publications. (Cit. on p. 6).

Rasmussen, C. E. 2004. “Gaussian Processes in Machine Learning.” In *Advanced Lectures on Machine Learning: ML Summer Schools 2003*, 63–71. Springer. (Cit. on pp. 26, 27, 36).

Rasmussen, C. E., and M. Kuss. 2004. “Gaussian Processes in Reinforcement Learning.” In *Advances in Neural Information Processing Systems 16 (NIPS-04)*, 751–759. (Cit. on p. 36).

Reinefeld, A. 1983. “An Improvement to the Scout Tree-Search Algorithm.” *International Computer Chess Association Journal* 6 (4): 4–14. (Cit. on p. 115).

Rosenblatt, F. 1958. “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological Review* 65 (6): 386–408. (Cit. on p. 28).

Rossi, F., K. B. Venable, and T. Walsh. 2011. *A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers. (Cit. on p. 6).

Rothkopf, C. A., and C. Dimitrakakis. 2011. “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*, 6913:34–48. Lecture Notes in Computer Science. (Cit. on p. 50).

Roy, B., and D. Bouyssou. 2002. *Aiding Decisions with Multiple Criteria: Essays in Honor of Bernard Roy*. International Series in Operations Research & Management Science. Springer. (Cit. on p. 6).

Rummery, G. A., and M. Niranjan. 1994. *On-Line Q-Learning Using Connectionist Systems*. Tech. rep. CUEF/F-INFENG/TR 166. Cambridge University Engineering Department. (Cit. on pp. 22, 94).

Runarsson, T. P., and S. M. Lucas. 2014. “Preference Learning for Move Prediction and Evaluation Function Approximation in Othello.” *IEEE Transactions on Computational Intelligence and AI in Games* 6 (3): 300–313. (Cit. on pp. 61, 65, 68, 75, 76, 78).

Runarsson, T. P., and S. M. Lucas. 2012. “Imitating play from game trajectories: Temporal difference learning versus preference learning.” In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG-12)*, 79–82. (Cit. on pp. 61, 65, 68, 75, 76, 78).

Samuel, A. L. 1959. “Some studies in machine learning using the game of checkers.” *IBM Journal on Research and Development* 3:210–229. (Cit. on p. 115).

Schölkopf, B., and A. J. Smola. 2001. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press. (Cit. on p. 112).

Schulman, J., S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz. 2015. “Trust Region Policy Optimization.” In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 1889–1897. (Cit. on pp. 32, 37, 69, 75, 78).

Seung, H. S., M. Opper, and H. Sompolinsky. 1992. “Query by Committee.” In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory (COLT-92)*, 287–294. (Cit. on p. 141).

Shacter, R., and M. Peot. 1989. “Simulation Approaches to General Probabilistic Inference on Belief Networks.” In *Uncertainty in Artificial Intelligence*. Morgan Kaufmann. (Cit. on p. 19).

Shamir, O. 2017. “An Optimal Algorithm for Bandit and Zero-Order Convex Optimization with Two-Point Feedback.” *Journal of Machine Learning Research* 18:1–11. (Cit. on p. 51).

Shun-ichi, A. 2012. *Differential-geometrical methods in statistics*. Vol. 28. Springer Science & Business Media. (Cit. on p. 31).

Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. 2014. “Deterministic Policy Gradient Algorithms.” In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 387–395. (Cit. on p. 37).

Singh, S. P., A. G. Barto, and N. Chentanez. 2004. “Intrinsically Motivated Reinforcement Learning.” In *Advances in Neural Information Processing Systems 17 (NIPS-04)*, 17:1281–1288. 2. (Cit. on p. 4).

Singh, S. P., R. L. Lewis, and A. G. Barto. 2009. “Where do rewards come from.” In *Proceedings of the 31th Conference of the Cognitive Science Society (CogSci-09)*, 2601–2606. (Cit. on p. 4).

Skiena, S. S. 1986. “An overview of machine learning in computer chess.” *International Computer Chess Association Journal* 9 (1): 20–28. (Cit. on p. 105).

Snelson, E., and Z. Ghahramani. 2006. “Sparse Gaussian Processes using Pseudo-inputs.” In *Advances in Neural Information Processing Systems 18 (NIPS-06)*, 1257–1264. (Cit. on p. 139).

Sokolov, A., J. Kreutzer, C. Lo, and S. Riezler. 2016. “Learning Structured Predictors from Bandit Feedback for Interactive NLP.” In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL-16)*. (Cit. on p. 51).

Spaan, M. T. J. 2012. “Partially Observable Markov Decision Processes.” In Wiering and van Otterlo 2012, 387–405. (Cit. on p. 13).

Sugiyama, H., T. Meguro, and Y. Minami. 2012. “Preference-learning based inverse reinforcement learning for dialog control.” In *Proceedings of the 13th Annual Conference of the International Speech Communication Association (INTERSPEECH-12)*, 222–225. (Cit. on pp. 61, 66, 68, 75, 76, 78).

Sutton, R. S. 1988. “Learning to Predict by the Methods of Temporal Differences.” *Machine Learning* 3:9–44. (Cit. on pp. 21, 105).

Sutton, R. S., and A. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press. (Cit. on pp. 13, 16, 17, 21, 22, 25, 94, 165, 169).

Tesauro, G. 1988. “Connectionist Learning of Expert Preferences by Comparison Training.” In *Advances in Neural Information Processing Systems 1 (NIPS-88)*, 1:99–106. (Cit. on p. 114).

– . 1992. “Practical issues in temporal difference learning.” In *Reinforcement Learning*, 33–53. Springer. (Cit. on p. 105).

– . 1995. “Temporal Difference Learning and TD-Gammon.” *Communications ACM* 38 (3): 58–68. (Cit. on p. 114).

– . 2001. “Comparison training of chess evaluation functions.” In *Machines that learn to play games*, 117–130. Nova Science Publishers, Inc. (Cit. on p. 114).

– . 2002. “Programming backgammon using self-teaching neural nets.” *Artificial Intelligence* 134 (1): 181–199. (Cit. on pp. 105, 114).

Thomaz, A. L., and C. Breazeal. 2006. “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)*, 1000–1006. (Cit. on p. 52).

Thurstone, L. L. 1927. “A Law of Comparative Judgement.” *Psychological Review* 34:278–286. (Cit. on pp. 6, 37).

Torrey, L., and M. E. Taylor. 2013. “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)*, 1053–1060. (Cit. on p. 49).

Utgoff, P., and J. Clouse. 1991. “Two kinds of training information for evaluation function learning.” In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, 596–600. (Cit. on p. 114).

Van Hasselt, H. 2012. “Reinforcement Learning in Continuous State and Action Spaces.” In Wiering and van Otterlo 2012, 207–243. (Cit. on pp. 13, 21).

Van Hasselt, H., and M. Wiering. 2007. “Reinforcement learning in continuous action spaces.” In *Proceedings of the IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-07)*, 272–279. (Cit. on p. 17).

Van Otterlo, M., and M. Wiering. 2012. “Reinforcement Learning and Markov Decision Processes.” In Wiering and van Otterlo 2012, 3–39. (Cit. on p. 13).

Vembu, S., and T. Gärtner. 2010. “Label Ranking Algorithms: A Survey.” In Fürnkranz and Hüllermeier 2010a, 45–64. (Cit. on p. 39).

Von Neumann, J., and O. Morgenstern. 1944. *Theory of Games and Economic Behavior*. Princeton University Press. (Cit. on p. 59).

Waegeman, W., and B. De Baets. 2010. “A Survey on ROC-based Ordinal Regression.” In Fürnkranz and Hüllermeier 2010a, 127–154. (Cit. on p. 40).

Wang, H. O., K. Tanaka, and M. F. Griffin. 1996. “An approach to fuzzy control of nonlinear systems: stability and design issues.” *IEEE Transactions on Fuzzy Systems* 4 (1): 14–23. (Cit. on p. 168).

Wang, Y., and I. H. Witten. 1997. “Induction of model trees for predicting continuous classes.” In *Poster papers of the 9th European Conference on Machine Learning (ECML-97)*. (Cit. on pp. 63, 78).

Watkins, C. J. C. H. 1989. “Learning from Delayed Rewards.” PhD thesis, King’s College. (Cit. on pp. 16, 22).

Weng, P. 2011. “Markov Decision Processes with Ordinal Rewards: Reference Point-Based Preferences.” In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-11)*. (Cit. on p. 49).

Weng, P., R. Busa-Fekete, and E. Hüllermeier. 2013. “Interactive Q-Learning with Ordinal Rewards and Unreliable Tutor.” In *Proceedings ECML/PKDD Workshop on Reinforcement learning from Generalized Feedback: Beyond Numerical Rewards*. (Cit. on p. 45).

Werbos, P. 1974. “Beyond regression: New tools for prediction and analysis in the behavioral sciences.” PhD thesis, Harvard University. (Cit. on p. 29).

Wiering, M., and M. van Otterlo, eds. 2012. *Reinforcement Learning: State-of-the-Art*. Springer.

Williams, C. K. I., and C. E. Rasmussen. 1996. “Gaussian Processes for Regression.” In *Advances in Neural Information Processing Systems 8 (NIPS-96)*, 514–520. (Cit. on pp. 26, 36).

Williams, R. J. 1987. “A class of gradient-estimating algorithms for reinforcement learning in neural networks.” In *Proceedings of the IEEE First International Conference on Neural Networks (ICNN-87)*, 601–608. (Cit. on p. 21).

– . 1992. “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning.” *Machine Learning* 8 (3-4): 229–256. (Cit. on p. 21).

Wilson, A., A. Fern, and P. Tadepalli. 2012. “A Bayesian Approach for Policy Learning from Trajectory Preference Queries.” In *Advances in Neural Information Processing Systems 25 (NIPS-12)*, 1142–1150. (Cit. on pp. 54, 55, 57, 69–72, 74, 76, 78, 129, 141–143, 148, 157, 158).

Wirth, C., and J. Fürnkranz. 2013a. “A Policy Iteration Algorithm for Learning from Preference-based Feedback.” In *Advances in Intelligent Data Analysis XII: 12th International Symposium (IDA-13)*, 8207:427–437. LNCS. (Cit. on pp. 9, 58, 67–70, 75, 76, 78, 84).

– . 2013b. “EPMC: Every Visit Preference Monte Carlo for Reinforcement Learning.” In *Proceedings of the 5th Asian Conference on Machine Learning, (ACML-13)*, 29:483–497. JMLR Proceedings. (Cit. on pp. 9, 58, 67–70, 75, 76, 78, 86).

Wirth, C., J. Fürnkranz, and G. Neumann. 2016. “Model-Free Preference-based Reinforcement Learning.” In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2222–2228. (Cit. on pp. 10, 61, 62, 64, 66–73, 75, 76, 78, 131, 157, 158).

Wu, T.-F., C.-J. Lin, and R. C. Weng. 2004. “Probability Estimates for Multi-class Classification by Pairwise Coupling.” *Journal of Machine Learning Research* 5:975–1005. (Cit. on p. 86).

Yue, Y., J. Broder, R. Kleinberg, and T. Joachims. 2012. “The K-armed Dueling Bandits Problem.” *Journal of Computer System Sciences* 78 (5): 1538–1556. (Cit. on p. 51).

Zhao, Y., M. Kosorok, and D. Zeng. 2009. “Reinforcement learning design for cancer clinical trials.” *Statistics in Medicine* 28 (26): 3294–3315. (Cit. on p. 5).

Zhifei, S., and E. Meng Joo. 2012. “A survey of inverse reinforcement learning techniques.” *International Journal of Intelligent Computing and Cybernetics* 5 (3): 293–311. (Cit. on pp. 45, 50).

Ziebart, B. D., A. Maas, J. A. Bagnell, and A. Dey. 2008. “Maximum Entropy Inverse Reinforcement Learning.” In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, 1433–1438. (Cit. on p. 50).

Zimin, A., and G. Neu. 2013. “Online learning in episodic Markovian decision processes by relative entropy policy search.” In *Advances in Neural Information Processing Systems 26 (NIPS-13)*, 1583–1591. (Cit. on p. 32).

Zucker, M., J. A. Bagnell, C. Atkeson, and J. Kuffner Jr. 2010. “An Optimization Approach to Rough Terrain Locomotion.” In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-10)*, 3589–3595. (Cit. on pp. 53, 61, 65, 66, 68, 70, 75, 76, 78).

Own Publications

Journals

Joppen, T., M. Moneke, N. Schröder, C. Wirth, and J. Fürnkranz. 2017. "Informed Hybrid Game Tree Search." *IEEE Transactions on Computationl Intelligence and AI in Games* PP (99): 1–1.

Wirth, C., R. Akrour, G. Neumann, and J. Fürnkranz. 2017. "A Survey of Preference-Based Reinforcement Learning Methods." *Journal of Machine Learning Research* (accepted). (Cit. on pp. 9, 43).

Wirth, C., and J. Fürnkranz. 2015. "On Learning from Game Annotations." *IEEE Transactions on Computational Intelligence and AI in Games* 7 (3): 304–316. (Cit. on pp. 9, 53, 61, 65, 68, 75, 76, 78, 103).

Conferences

Wirth, C., and J. Fürnkranz. 2013a. "A Policy Iteration Algorithm for Learning from Preference-based Feedback." In *Advances in Intelligent Data Analysis XII: 12th International Symposium (IDA-13)*, 8207:427–437. LNCS. (Cit. on pp. 9, 58, 67–70, 75, 76, 78, 84).

– . 2013b. "EPMC: Every Visit Preference Monte Carlo for Reinforcement Learning." In *Proceedings of the 5th Asian Conference on Machine Learning, (ACML-13)*, 29:483–497. JMLR Proceedings. (Cit. on pp. 9, 58, 67–70, 75, 76, 78, 86).

Wirth, C., J. Fürnkranz, and G. Neumann. 2016. "Model-Free Preference-based Reinforcement Learning." In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI-16)*, 2222–2228. (Cit. on pp. 10, 61, 62, 64, 66–73, 75, 76, 78, 131, 157, 158).

Workshops

Wirth, C., and J. Fürnkranz. 2012. "First Steps Towards Learning from Game Annotations." In *Proceedings of the ECAI Workshop on Preference Learning: Problems and Applications in AI*, 53–58. (Cit. on pp. 9, 53, 61, 65, 68, 75, 76, 78, 113, 118, 119, 121).

– . 2013c. “Learning from Trajectory-Based Action Preferences.” In *Proceedings of the ICRA 2013 Workshop on Autonomous Learning*. (Cit. on p. 9).

– . 2013d. “Preference-Based Reinforcement Learning: A Preliminary Survey.” In *Proceedings of the ECML/PKDD-13 Workshop on Reinforcement Learning from Generalized Feedback: Beyond Numeric Rewards*. (Cit. on pp. 9, 43).

Wissenschaftlicher Werdegang des Verfassers[53]

2011 - 2017 PhD Computer Science, Technische Universität Darmstadt

Thesis title: *Efficient Preference-based Reinforcement Learning*
Conventional reinforcement learning methods are confined to deal with numerical feedback that is not always available. Therefore, this project aims at realizing a high performance framework for learning from more general feedback like preferences.

2007 - 2010 MSc Computer Science, Technische Universität Darmstadt

Thesis title: *Determining Move Effects Without Explicit State Construction*
Reasoning in complex domains is costly because of interleaved rules. This work shows how to speed up reasoning using automatically created, independent rules.

2003 - 2007 BSc Computer Science, Technische Universität Darmstadt

Thesis title: *Interactive Splineapproximation for Procedural-generated Terrain Data*

Ehrenwörtliche Erklärung[55]

Hiermit erkläre ich, die vorgelegte Arbeit zur Erlangung des akademischen Grades „Doctor rerum naturalis“ mit dem Titel *„Efficient Preference-based Reinforcement Learning: Using Surrogates for Solving Markov Decision Processes with Preferences“* selbständig und ausschließlich unter Verwendung der angegebenen Hilfsmittel erstellt zu haben. Ich habe bisher noch keinen Promotionsversuch unternommen.

Darmstadt, den 27.04.2017

Christian Wirth

[53] Gemäß §20 Abs. 3 der Promotionsordnung der Technischen Universität Darmstadt
[55] Gemäß § 9 Abs. 1 der Promotionsordnung der Technischen Universität Darmstadt.

Index