Robust Preference Learning-based Reinforcement Learning

Riad Akrour

UNIVERSITE PARIS-SUD

Ecole Doctorale d'Informatique

P H D T H E S I S

to obtain the title of

PhD of Science

of the Université Paris-Sud

Specialty : Computer Science

Defended by

Riad Akrour

Robust Preference Learning-based Reinforcement Learning

Thesis Advisors: Michèle Sebag and Marc Schoenauer

TAO Team

defended on September 30th, 2014

Jury :

Reviewers :	Joelle Pineau	-	Univ. McGill, Canada
	Johannes Fürnkranz	-	Univ. Darmstädt, Germany
Advisors :	Michele Sebag	-	Cnrs, Univ Paris-Sud (TAO)
	Marc Schoenauer	-	Inria, Univ Paris-Sud (TAO)
Examiners :	Michel Beaudouin-Lafon	-	LRI, Univ. Paris-Sud
	Pierre-Yves Oudeyer	-	Inria (Flowers)
	Paolo Viappiani	-	Cnrs, Univ. Paris 6

Acknowledgments

Contents

Chapter 1

Introduction

This thesis is concerned with sequential decision making and more specifically with reinforcement learning (RL). Among the main three fields of Machine Learning, namely supervised learning, unsupervised learning and RL, the latter became an increasingly hot topic in the last decade or so, due to rapid progress of the RL theory and algorithms on the one hand, and due to applicative breakthroughs on the other hand.

RL was considered to be a most promising field ever since the early days of Artificial Intelligence. It offered ML some of its first world successes, illustrated by the TD-Gammon reaching the expertise level of world master human players [Tesauro 1995]. The similarities between RL algorithms and human cognition were and still are acknowledged [Thorndike 1911, Khamassi *et al.* 2005]. Its applications, which used to focus on games and robotics [Deisenroth *et al.* 2013], now extend to diverse problems in the industrial and medical domains, including autonomic computing [Tesauro 2007], medical prescriptions and adaptive neurological stimulations [Pineau *et al.* 2009]. It is also the case that some complex problems are cast as reinforcement learning problems, whenever they involve many interdependent decisions and sub-problems, and the optimum solution is not obtained by solving optimally and independently every sub-problem. More generally, when the *whole is not the sum of its parts*, the solvers related to every sub-problem are required to cooperate, and the global problem can be tackled as an RL problem. Such application domains range from Natural Language Processing [Chang *et al.* 2012] and Document Understanding [Maes *et al.* 2009] to Data Mining itself, gearing the great many pre-processing and processing decisions toward the global success of the application.

RL is built on a firm and well founded theory, rooted in Optimal Control and

(Approximate) Dynamic Programming [Sutton & Barto 1998a, Szepesvari 2010]. As of today, algorithmic achievements are behind theoretical advances, for scalability reasons; a significant body of work is concerned with increasing the applicability of standard RL algorithms through reducing the dimensions of RL problems, including the dimensions of the state space (e.g. using random projections [Ghavamzadeh *et al.* 2010, Milani Fard *et al.* 2013]), or the action space (e.g. defining macro-actions aka options [Brunskill & Li 2014]). Other approaches directly tackle RL as an optimization problem [Deisenroth *et al.* 2013, Peters & Schaal 2008, Stulp & Sigaud 2013]. In the recent years, another well-founded framework was proposed for RL, based on tree-structured multi-armed bandits, specifically Monte-Carlo Tree Search and the celebrated Upper Confidence Tree [Kocsis & Szepesvári 2006] inspired from the UCB [Auer *et al.* 2002]. These any-time algorithms scale up to medium size problems. They supported major ML breakthroughs in the last 10 years, illustrated by the computer-Go players [Gelly & Silver 2007, Gelly *et al.* 2012] now nearly reaching the level of world champions. The significance of these advances is best understood by contrasting the computational complexity of computer-Go and computer-chess players [Hsu 2002]; the size and complexity of the Go state space is larger by several orders of magnitude (branching factor circa 20 against ca 6), and efficient assessment functions are still to be designed.

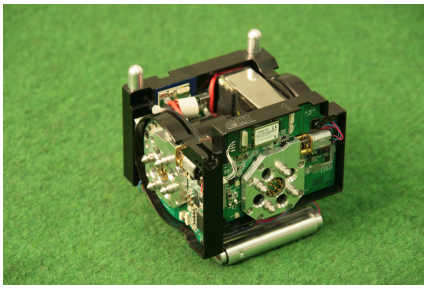
The main issue tackled in this PhD concerns the dependency of the RL algorithm on the expertise of the designer, the expert or the user. As mentioned, reinforcement learning tackles an optimization problem. In its formal setting (chapter 2), the optimization objective depends on the reward function associating an instant evaluation to each state visited by the agent. The difficulty of the optimization problem is significantly ameliorated if the reward function supports the identification of relevant milestones in the RL search space. At the other extreme, if the designer sets the reward function in such a way that it rewards the target state and no other state, then the RL problem can be thought of in terms of a *Needle in the Haystack* optimization problem: extensive exploration is required to identify the promising regions of the search space.

The first part of the manuscript is devoted to the presentation and discussion of the state of the art, starting with the formal background of Reinforcement Learning and Markov Decision Processes (chapter 2). Chapter 3 presents the so-called *Inverse Reinforcement Learning* (IRL) approach. First pioneered by Russell and Ng [Ng & Russell 2000] and by Abbeel and Ng [Abbeel & Ng 2004, Abbeel 2008], IRL alleviates (in a sense to be formalized) the RL dependency on the human expertise. Other related work, concerned with the *Human in the loop* and with Interactive Learning for Optimization (ILO), are presented and discussed in chapter 4.

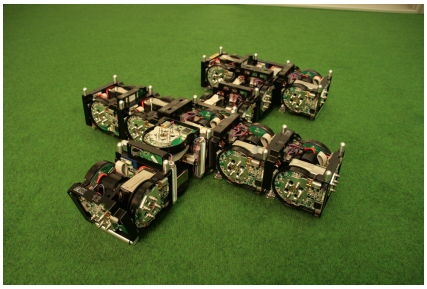
The second part of the manuscript describes our contributions and the theoretical and empirical validations thereof.
In our targeted application domain, IRL is not applicable *per se*: this PhD was funded by the European SYMBRION Integrated Project (FET IP 216342), aimed at robotic organisms. As shown on Fig. 1.1, right, the goal is to design flexible robotic organisms based on the physical docking and un-docking of a great many robotic units (Fig. 1.1, left). While robotic swarms and organisms based on the cooperation of a great number of inexpensive robotic units offer many advantages in terms of cost *vs* robustness *vs* sophistication ratios, their control raises new and challenging issues. In particular, IRL is not applicable to the control of a swarm, for two reasons. On the one hand, the expert designer does not know the desired behavior of the individual robot, yielding the desired behavior of the whole organism[1]; this issue is referred to as the inverse problem of *design of emergent systems* [Kaneko & Tsuda 2000]. On the other hand, a single expert designer would find it hard to demonstrate the expected behavior of all robotic units involved in a swarm or an organism; the demonstration would rather involve a swarm of experts – which is impractical and unrealistic in many ways[2]. Current approaches of swarm robotics are hence based on either carefully hand-crafted controllers (e.g., rule-based [Groß *et al.* 2006]), with the drawback of limited adaptive qualities, or on guided in-silico Direct Policy Search using Evolutionary Algorithms [Groß *et al.* 2006, O'Dowd *et al.* 2011], as illustrated

[1] The question is to design the controller of an ant, in such a way that the ant colony yields the desired behavior [Chevallier *et al.* 2010].

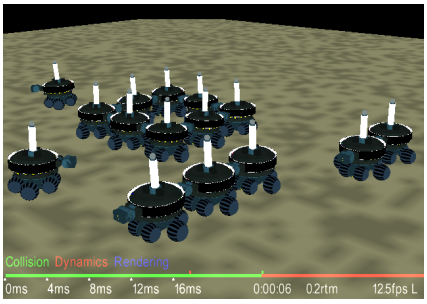
[2] Imitation enhanced Q-learning has recently been proposed at UWE [Erbas *et al.* 2011, Erbas *et al.* 2014] in which a single 'educated' robot helps other robots to rapidly improve their behaviors in a Q-learning context.



(a) individual robot



(b) robot organism made of docked individual robots



(c) Swarm-bot (2001-2005)

(d) Swarm Foraging, UWE

Figure 1.1: SYMBRION: Robot individual (a) and robot organism (b). Examples of swarms: Swarm-bot [Groß *et al.* 2006] (c) and Swarm Foraging [O'Dowd *et al.* 2011] (d).

on Fig. 1.1, bottom-right and bottom-left, that requires an ad hoc simulator.

In 2011, J. Fürnkranz, E. Hüllermeyer and their collaborators [Cheng *et al.* 2011, Hüllermeier & Fürnkranz 2013] noted that associating a numerical reward to each action, as done in standard RL, was hardly meaningful in some contexts, e.g. to account for the death issue in a medical prescription application domain. An alternative was offered by ranking state-action pairs conditionally to a policy π, e.g. ranking two state-action pairs depending on whether death occurred by following π after the first state-action pair, and not after the second one.

Simultaneously though independently, we explored a related approach, dubbed Preference-based Policy Learning (PPL). Let us note that since 2011, quite a few other works (including [Wilson *et al.* 2012, Knox & Stone 2012, Jain *et al.* 2013]) were devoted to preference learning within RL (see chapter 4).

In PPL, the expert is iteratively asked her preferences among two trajectories of

the robot[3]. The PPL algorithm, first contribution of the present PhD work (chapter 5), iteratively learns a policy return estimate through a state-of-art learning-to-rank algorithm [Joachims 2005] based on the expert's preferences. An optimization criterion, adaptively defined from the policy return estimate and an exploration term, is tackled using a Direct Policy Search algorithm [Akrour *et al.* 2011].

It soon appeared that a main issue for PPL was to reduce as much as possible the number of preference queries to the expert and thereby limit his cognitive burden. Our second contribution (chapter 6) thus investigated the use of *Active Preference Learning* [Ailon 2012] within PPL, in order to minimize the number of queries needed to elicit the expert's preference model. However, our goal differs significantly from Active Preference Learning, for the following reason. The goal of active learning is to find an accurate preference model. Quite the contrary, our goal is to find *the optimum* of this model; the model accuracy is a means, not an end. Taking inspiration from the related work of [Viappiani & Boutilier 2010], the *Active Preference-based Reinforcement Learning* (APRIL) algorithm was thus proposed [Akrour *et al.* 2012].

Along *in-situ* experimentations, a limitation of the APRIL scheme regarding its robustness w.r.t. the expert's mistakes was identified. As could have been expected, it appears that no expert (including ourselves) is mistake-free; and the impact of these mistakes on the preference model might be hard to recover. This impact is all the more detrimental to the whole approach as the expert can deliberately change her preferences along the interaction with the robot in a pedagogic curriculum, e.g. focusing on the acquisition of some walking skill before tackling the running skill.

As a third contribution, we thus proposed an algorithm aimed at coping with the expert's mistakes while interactively asking for the user's preferences (chapter 7). Like PPL and APRIL, the resulting framework called *Programming by Feedback* proceeds by iterating a two-step process: i/ presenting the user with a new

[3]In principle, the PPL principle extends to swarm trajectories as well. Due to time constraints however, only the single robot case was considered in the scope of the PhD. The extension and empirical validation of the presented approach to swarm robotics are left for further work (chapter 8).

trajectory and asking whether this trajectory improves on the previous best; ii/ updating the preference model depending on the user's preference, and searching for the next best and most informative trajectory to be displayed [Akrour *et al.* 2014].

All three algorithms proposed in the contribution chapters are introduced together with an analytical study of their behavior and an experimental validation comparatively to the prominent state-of-the-art methods, illustrating their respective merits and limitations, and analyzing their scalability as well as their sensitivity w.r.t. their hyper-parameters.

Finally, the last chapter of this manuscript discusses globally the contributions of the presented work and proposes several perspectives of research for further work.

Chapter 2

Reinforcement Learning

Foreword

Reinforcement Learning (RL) [Kaelbling *et al.* 1996, Sutton & Barto 1998b, Wiering & van Otterlo 2012] considers an agent perceiving the environment at discrete time steps $t = 0, 1, \ldots, H \leq \infty$ through its current state s_t (belonging in the *state space* $\mathcal{S}$). The agent has the opportunity to interact with the environment by executing an action a_t in the *action space* $\mathcal{A}$, upon which a transition from s_t to s_{t+1} will occur according to the probability distribution $\mathcal{P}(s_{t+1}|s_t, a_t, \ldots, s_0, a_0)$ ($\mathcal{P}$ is most often referred to as the *transition model*). In a first step (sections 2.1 and 2.2), the state and action spaces are assumed to be discrete, finite and ranging in sets of small to medium size. These assumptions will be relaxed in section 2.3 and thereafter.

Upon executing action a_t in state s_t, the agent gets from the environment a scalar *reward* $\mathcal{R}(s_t, a_t, s_{t+1}) \in \mathbb{R}$. Most usually the reward only depends on the current state s_t, or on both s_t and the current action a_t. Otherwise $\mathcal{R}(s_t, a_t)$ is defined as the expectation of $\mathcal{R}(s_t, a_t, s_{t+1})$, where the expectation is taken over the transition distribution ($\mathbb{E}_{s_{t+1} \sim \mathcal{P}(.|s_t, a_t)}[\mathcal{R}(s_t, a_t, s_{t+1})]$).

Let a deterministic policy π be defined as a mapping from $\mathcal{S}$ onto $\mathcal{A}$, associating an action to any state; analogously a stochastic policy π is defined as a mapping from $\mathcal{S} \times \mathcal{A}$ onto $[0, 1]$, where $\pi(s, a)$ is the probability of executing a in s.
The goal of Reinforcement Learning algorithms is to find a policy maximizing the discounted long term reward

$$\sum_{t=0}^{H} \gamma^t \mathcal{R}(s_t, \pi(s_t), s_{t+1})$$

The discount factor $\gamma \in [0, 1]$ indicates how important are future rewards compared to immediate ones. Term $1 - \gamma$ can be interpreted as the probability for the agent

to die in any time step and thereby stop receiving rewards: the lower γ, the shorter the agent lifetime on average, and the keener the agent will be to look for short term rewards. To avoid the discount of future rewards, some algorithms rather optimize the average reward

$$\frac{1}{H}\sum_{t=0}^{H}\mathcal{R}(s_t, \pi(s_t), s_{t+1})$$

thereby giving equal importance to immediate and future rewards.

This chapter introduces the formal background of Markov Decision Process and the main RL algorithms. The evolutionary RL approach is also discussed.

2.1 Markov Decision Processes

Reinforcement Learning (RL) algorithms are couched in the Markov Decision Process (MDP) formalism, specifying how the environment reacts to the actions of the agent. An MDP is fully specified by the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \mathcal{P}_0, \gamma >$. The Markovian assumption implies that the knowledge of a_t and s_t fully specifies the transition model $\mathcal{P}$:

$$\mathcal{P}(s_{t+1}|s_t, a_t, \ldots, s_0, a_0) = \mathcal{P}(s_{t+1}|s_t, a_t)$$

Distribution $\mathcal{P}_0 : \mathcal{S} \mapsto [0, 1]$ defines the probability $\mathcal{P}_0(s_0 = s)$ that the agent starts in initial state s.

Note that even though the existence of the transition model $\mathcal{P}$ is necessary to fully define an MDP, some RL algorithms only estimate it from samples (s_t, a_t, s_{t+1}, r_t) gathered by the agent wandering in the environment.

The goal of the RL algorithms is to find a policy π^* maximizing the expected sum of discounted rewards gathered by following this policy. In order to do so, one defines some target functions to be optimized, called *value functions*.

2.1.1 Value function and Bellman equation

For a policy $\pi : \mathcal{S} \mapsto \mathcal{A}$, the value function of π at state s given by Equation (2.1) is the discounted sum of rewards gathered when the agent starts at state s, selects actions according to π, moves to state s_{t+1} according to transition model $\mathcal{P}$, and

receives a reward r_t according to $\mathcal{R}$, for $t = 0, \ldots, H$.

$$V^{\pi}(s) = \mathbb{E}_{s_{t+1} \sim \mathcal{P}(s_t, \pi(s_t))} \left[\sum_{t=0}^{H} \gamma^t r_t \middle| s_0 = s \right] \tag{2.1}$$

By definition $V(\pi) = \mathbb{E}_{s_0 \sim \mathcal{P}_0}[V^{\pi}(s_0)]$ is the expectation of $V^{\pi}(s)$ along the initial state distribution $\mathcal{P}_0$.

When the time horizon H is infinite, we can recursively write the V-Value at state s as a sum of an immediate reward and the V-Values of the next states:

$$V^{\pi}(s) = \mathcal{R}\left(s, \pi\left(s\right)\right) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}\left(s' \middle| s, \pi\left(s\right)\right) V^{\pi}(s') \tag{2.2}$$

Eq. (2.2) is the Bellman equation [Bellman 1957]; its recursive structure is at the core of most RL algorithms aimed at optimizing the V-Value.

Let $\mathcal{F} = \{f : \mathcal{S} \mapsto \mathbb{R}\}$ be the space of functions mapping the state space $\mathcal{S}$ onto $\mathbb{R}$ (which includes all value functions associated to any policy π). Let us define the Bellman operator as the mapping on $\mathcal{F}$ defined as:

$$\begin{aligned} & T_{\pi} : \mathcal{F} \mapsto \mathcal{F} \\ \forall f \in \mathcal{F}, \quad & T_{\pi} f(s) = \mathcal{R}\left(s, \pi\left(s\right)\right) + \gamma \textstyle\sum_{s' \in \mathcal{S}} \mathcal{P}\left(s' | s, \pi\left(s\right)\right) f(s') \end{aligned}$$

Whatever policy π, the Bellman operator T_{π} is monotonic and is a contraction on the metric space $(\mathcal{F}, ||.||_{\infty})$ [Bertsekas & Tsitsiklis 1996]. Formally,

$$(\forall s \in \mathcal{S}, V(s) \leq V'(s)) \Rightarrow (\forall s \in \mathcal{S}, T_{\pi} V(s) \leq T_{\pi} V'(s))$$

by linearity of the expectation. Likewise, one shows that

$$||T_{\pi} f - T_{\pi} g||_{\infty} \leq \gamma ||f - g||_{\infty}$$

i.e. T_{π} is a contraction with contraction rate γ. Since V^{π} is invariant under T_{π} by construction ($T_{\pi} V^{\pi} = V^{\pi}$), it follows that, for any $f \in \mathcal{F}$:

$$||T_{\pi} f - T_{\pi} V^{\pi}||_{\infty} = ||T_{\pi} f - V^{\pi}||_{\infty} \leq \gamma ||f - V^{\pi}||_{\infty}$$

In other words, for $\gamma < 1$, applying T_{π} onto any function $f \in \mathcal{F}$ different from the V-Value of π yields a new function $T_{\pi} f$ that is closer by a factor γ to V^{π}. It follows immediately that V^{π} is the unique fixed point of T_{π} and that n successive applications of T_{π} to some f will lead to a function that is far from V^{π} by at

most γ^n times the initial distance $||f - V^\pi||_\infty$. Moreover, the Banach fixed point theorem states that successive applications of T_π starting from any initial condition will indeed converge to this fixed point. As will be seen in sections 2.2 and 2.3, the repeated application of the Bellman operator is at the core of many RL algorithms.

Let us define the greedy policy π' with respect to V^π as follows:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}\left(s'|s,a\right) V^\pi(s')$$

It follows that, from the knowledge of the MDP gathered through the computation of V^π, a new greedy policy π' can be defined with a higher V-Value ($V^{\pi'} \geq V^\pi$, see below).

2.1.2 The Q-Value function

A most convenient concept for RL algorithms is the Q-Value function, mapping the state-action space $\mathcal{S} \times \mathcal{A}$ onto $\mathbb{R}$ as follows

$$Q^\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) Q^\pi(s', \pi(s'))$$

For a deterministic policy π, the Q-value function is closely related to V^π as $V^\pi(s) = Q^\pi(s, \pi(s))$. However, many RL algorithms choose to estimate Q^π instead of the value function, because it allows a straightforward computation of the associated greedy policy: the definition of the greedy policy based on V^π requires to know both the transition probabilities $\mathcal{P}$ of the MDP and the reward function $\mathcal{R}$. Quite the contrary, the greedy policy π' can be defined from the Q-Value function as

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q^\pi(s,a)$$

and does not require the knowledge of $\mathcal{P}$ or $\mathcal{R}$. The fact that the greedy policies defined from V^π or Q^π improve on π is known as the *Policy Improvement Theorem* [Sutton & Barto 1998b], and its proof is straightforward:

Proof. From the definition of the greedy policy π', it follows that

$$\forall s \in \mathcal{S}, V^\pi(s) \leq Q^\pi\left(s, \pi'\left(s\right)\right).$$

The result follows from successive applications of this inequality:

$$\begin{aligned}
\forall s \in \mathcal{S}, V^{\pi}(s) &\leq Q^{\pi}\left(s, \pi'(s)\right) \\
&= \mathbb{E}_{s' \sim \mathcal{P}(.|s,\pi'(s))}\left[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')\right] \\
&\leq \mathbb{E}_{s' \sim \mathcal{P}(.|s,\pi'(s))}\left[\mathcal{R}(s,a,s') + \gamma Q^{\pi}(s', \pi'(s'))\right] \\
&= \mathbb{E}_{s' \sim \mathcal{P}(.|s,\pi'(s))}\left[\mathcal{R}(s,a,s') + \gamma \mathbb{E}_{s'' \sim \mathcal{P}(.|s',\pi'(s'))}\left[\mathcal{R}(s',a,s'') + \gamma V^{\pi}(s'')\right]\right] \\
&= \mathbb{E}_{s' \sim \mathcal{P}(.|s,\pi'(s)), s'' \sim \mathcal{P}(.|s',\pi'(s'))}\left[\mathcal{R}(s,a,s') + \gamma \mathcal{R}(s',a,s'') + \gamma^2 V^{\pi}(s'')\right] \\
&\vdots \\
&\leq V^{\pi'}(s) \qquad \square
\end{aligned}$$

Moreover, if $\exists s \in \mathcal{S}, Q^{\pi}(s, \pi'(s)) > V^{\pi}(s)$ then $V^{\pi} \neq V^{\pi'}$, and hence π' strictly improves on π.

2.1.3 Optimal policy and value function

After the theory of Dynamic programming [Bellman & Dreyfus 1962], for any MDP there exists at least one optimal policy that is both deterministic and stationary. Let us denote this policy by π^*. A policy is optimal if at each state, there is no policy having a strictly higher V-Value. Letting V^* denote the (optimal) V-Value of such a policy, we have $V^*(s) = \max_{\pi} V^{\pi}(s)$ for all $s \in \mathcal{S}$. Analogous to T^{π}, the Bellman optimality operator $T : \mathcal{F} \mapsto \mathcal{F}$ associates to any function $f : \mathcal{S} \mapsto \mathbb{R}$ defined on the state space, the function $Tf : \mathcal{S} \mapsto \mathbb{R}$ defined by:

$$Tf(s) = \max_{a \in \mathcal{A}} \left\{ \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}\left(s'\middle|s,a\right) f(s') \right\}$$

Operator T has the same monotonic and contraction properties than T^{π}; its successive applications on any function f defined from $\mathcal{S}$ onto $\mathbb{R}$ converge to the optimal value function V^*.

Likewise, a Bellman optimality operator is defined for the Q-Value functions. Letting T be the operator defined on functions from $\mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ in the following way:

$$Tf(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}\left(s'\middle|s,a\right) \max_{a' \in \mathcal{S}} f(s', a')$$

then T is a contraction mapping leading to $Q^* = \max_{\pi} Q^{\pi}$ after successive applications of T on any initial function $f : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$.

2.2 Algorithms for MDP

2.2.1 Value and Policy Iteration

The Value Iteration algorithm [Puterman 1994] aims at computing the optimal value function V^*. The algorithm proceeds by iteratively applying the Bellman optimality

Algorithm 1 Value Iteration

$t \leftarrow 0$

$V_0 \leftarrow$ arbitrary function defined on $\mathcal{S} \mapsto \mathbb{R}$

repeat

  $t \leftarrow t + 1$

  $V_t \leftarrow TV_{t-1}$

until $||V_t - V_{t-1}||_\infty \leq \varepsilon$

return V_t

operator, eventually leading to V^*. Its stopping criterion is when two successive value functions are sufficiently close to each other ($||V_t - V_{t-1}||_\infty < \varepsilon$, where ε is a user-defined parameter), thereby yielding an upper bound on the distance between V_t and V^* [Bertsekas & Tsitsiklis 1996]:

$$||V_t - V^*||_\infty \leq \frac{\gamma}{1-\gamma}\varepsilon$$

The algorithm can also apply on the Q-Value function, thereby facilitating the construction of the greedy policy (section 2.1.2).

Algorithm 2 Policy Iteration

$t \leftarrow 0$

$\pi_0 \leftarrow$ arbitrary policy mapping $\mathcal{S}$ onto $\mathcal{A}$.

1: **repeat**

2: $t \leftarrow t + 1$

3: $\pi_t \leftarrow$ greedy policy w.r.t. Q_{t-1}

4: $Q_t \leftarrow Q^{\pi_t}$

5: **until** $Q_t = Q_{t-1}$

return π_t

A second well known algorithm for finding an optimal policy of an MDP is the Policy Iteration algorithm [Puterman 1994]. It alternates two steps: the policy

evaluation step (line 4 of Algorithm 2), and the construction of the greedy policy (line 3) also called the policy improvement step. The policy evaluation step aims at the Q-Value of policy π_t; it proceeds as in Value Iteration except that the Bellman optimality operator T is replaced by T_{π_t}, the Bellman operator of π_t. The greedy policy π_t (line 3) is derived by greedily following the previously computed $Q^{\pi_{t-1}}$. If the current Q-Value does not improve on the previous one, the algorithm terminates and the current policy π_t is returned; π_t is optimal because, as seen in section 2.1.2, $Q_t = Q_{t-1}$ implies that $\nexists s \in \mathcal{S}, Q_{t-1}(s, \pi_{t-1}(s)) \neq \max_{a \in \mathcal{A}} Q_{t-1}(s, a)$. This implies that Q_{t-1} is stable to the application of T, and hence $Q_t = Q_{t-1} = Q^*$.

Since Policy Iteration evaluates the policy in its inner loop, one iteration of Policy Iteration generally takes longer than in Value Iteration. However, Policy Iteration requires in general fewer iterations to converge [Bertsekas & Tsitsiklis 1996]. In addition, when $|\mathcal{S}|$ is small enough, it becomes more advantageous to compute the value function using a matrix inversion:

$$\begin{aligned} V^{\pi} &= \mathcal{R}_{\pi} + \sum_{t=1}^{\infty} \gamma^t \mathcal{P}_{\pi}^t \mathcal{R}_{\pi} \\ &= (I - \gamma \mathcal{P}_{\pi})^{-1} \mathcal{R}_{\pi} \end{aligned}$$

$\mathcal{R}_{\pi}$ and $\mathcal{P}_{\pi}$ are respectively an $|\mathcal{S}|$ dimensional vector and an $|\mathcal{S}| \times |\mathcal{S}|$ dimensional matrix where for any state s, $\mathcal{R}_{\pi}(s) = \mathcal{R}(s, \pi(s))$ and $\mathcal{P}_{\pi}(s, s') = \mathcal{P}(s'|s, \pi(s))$.

Besides Value Iteration and Policy Iteration, there are more general algorithms that encompass both of them as special cases, such as λ-Policy Iteration [Bertsekas & Ioffe 1996] and Modified Policy Iteration [Puterman 1994]. For a comprehensive overview see [Thiéry 2010].

2.2.2 Model-Free RL

The previous algorithms solve the problem of optimally behaving in an MDP by using the exact Bellman (optimality) operator. This operator in turn requires the full knowledge of the reward function $\mathcal{R}$ and the dynamics of the environment given as the transition model $\mathcal{P}(.|s, a)$.

Model-free algorithms relax these requirements and assume that the learning agent only has access to a dataset $\mathcal{D} = \{(s, a, r, s')_i, i = 1 \dots N\}$ of transition samples, where tuple (s, a, r, s') indicates that upon executing action a in state s, the

agent reaches state s' and receives reward r. An important distinction between the algorithms is the level of control they require on how the samples in $\mathcal{D}$ are collected. Two categories are distinguished: On-policy algorithms assume the samples are generated by the agent following its (evolving) policy; off-policy algorithms use samples that have been generated by any policy. A representative algorithm of the on-policy category (respectively off-policy category) is SARSA [Rummery & Niranjan 1994] (resp. Q-Learning [Watkins 1989]).

Q-Learning is an online off-policy, model-free RL algorithm. As opposed to batch algorithms, it does not need to actually build $\mathcal{D}$; it incrementally processes the transition samples[1]. Furthermore the samples do not need to be sequential (the ending state of a sample being the starting state of another one). Q-learning starts with an arbitrarily initialized Q-Value $Q_0 : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. In each i-th iteration, upon seeing (s, a, r_i, s'_i) the Q-Value is updated as follows:

$$Q_i(s,a) = Q_{i-1}(s,a) + \alpha_i(s,a)\left(r_i + \gamma \max_{a' \in \mathcal{A}} Q_{i-1}(s'_i, a') - Q_{i-1}(s,a)\right)$$

The update can be interpreted as computing the difference between the current Q-Value at state (s, a) and an estimate of the Bellman optimality operator T applied to Q (where the estimate considers a single sample), and the difference is used to refine the Q-value with learning rate α_i. Note that Q-learning can be interpreted as an empirical Value Iteration algorithm: assuming that[2] $\alpha_i(s,a) = \frac{1}{i}$ and Q_0 the null vector, then by solving the recursion it comes:

$$Q_N(s,a) = \frac{1}{N}\sum_{i=1}^{N} r_i + \gamma \frac{1}{N}\sum_{i=1}^{N} \max_{a' \in \mathcal{A}} Q_{i-1}(s'_i, a')$$

SARSA is very similar to Q-Learning, with the difference that, instead of using $\max_{a' \in \mathcal{A}} Q_{i-1}(s'_i, a')$ for an estimate of future rewards, it uses $Q_{i-1}(s'_i, a'_i)$ for a'_i the action executed in s'_i by the policy of the agent, generating the sample (s, a, r, s'_i, a'_i) (and giving its name to the algorithm). For these reasons, SARSA needs to be on-policy to learn the Q-Value of the policy of the agent, and the agent can in turn use this Q-Value to improve its policy as in Policy Iteration.

[1] Although it does not need to store the samples, it is however commonplace to store and reuse the samples whenever their acquisition is expensive.

[2] In practice, $\alpha_i(s,a)$ is most usually kept constant across time and for all state action pairs (s,a).

If the agent executes every action in every state an infinite number of times, and for all s and a, $\sum_{i=1}^{\infty} \alpha_i(s,a) = \infty$ and $\sum_{i=1}^{\infty} \alpha_i(s,a)^2 < \infty$ then the Q-Learning algorithm converges with probability 1 to the optimal Q-Value [Watkins & Dayan 1992].

Similar guarantees can be obtained for SARSA [Singh *et al.* 1998]; the theoretical analysis is however more difficult as some care must be exercised to enforce the exploration of all states and actions. Formally, the online policy cannot be the greedy one w.r.t. the current Q-Value, it needs to incorporate some *exploration* to ensure all states and actions are visited. However, this exploration goes against the quest for optimality. The situation is simpler in the Q-learning context as its off-policy nature enables to consider two distinct policies, an exploration policy generating off-line the samples, and the policy learned using these samples.

Efficiently incorporating exploration such as to satisfy the requirement of visiting all the states and actions of an MDP, yet exploiting the acquired knowledge to focus on the most promising states of the MDP is a long known dilemma named exploration/exploitation trade-off. A simple exploration scheme known as ε-greedy, performs a random action with probability ε and otherwise greedily follows Q. Another way of favoring exploration is through the optimistic initialization of Q_0 to a high value, for instance upper bounding the reward function. Doing so ensures that at least in the early steps, exploitation will inevitably decreases the Q-Value of the visited state-action pairs thus ensuring that the greedy policy will explore any unexplored action due to their high Q-Value.

More principled approaches at exploration achieving near-optimal regret are based on upper confidence bounds in methods such as R-max [Brafman & Tennenholtz 2003] or UCRL2 [Jaksch *et al.* 2010]. However these methods use estimates of the probability transitions $\mathcal{P}(.|s,a)$ which may limit their applicability when dealing with very large state spaces.

2.3 Approximate algorithms for large state spaces

All RL algorithms described in section 2.2 represent the value function (resp. Q-Value) with a look-up table, associating to each state s (resp. state-action pair (s,a)) the value $V(s)$ (resp. $Q(s,a)$). Although the memory requirement only grows linearly in $|\mathcal{S}|$ (resp. $|\mathcal{S} \times \mathcal{A}|$), it can be prohibitive to store these estimates

for each state or state-action pair[3]. In addition to the memory requirements, the computational resources and data resources (number of required samples (s, a, r, s')) required can be the most limiting factor for the scalability of RL. Ideally, one would like the learned estimate for one state to **generalize** to neighbor states, in a metric sense to be defined.

2.3.1 Approximate RL with Supervised Learning

A first way to scale RL algorithms to large state spaces is by leveraging supervised learning algorithms, with strong generalization guarantees. Classification-based Approximate Policy Iteration algorithm (CAPI) [Lagoudakis & Parr 2003b, Fern *et al.* 2006] is a Policy Iteration algorithm (section 2.2.1) where the look-up table-based representation of the Q-Value is replaced by a multiclass classifier. Assuming that the action space $\mathcal{A}$ is of limited size, this classifier associates to each state s an action a. In this way, CAPI sidesteps learning the full Q-Value: the classification algorithm is trained from state-action pairs (s, a) where a maximizes the (estimated through simulation) Q-Value $Q^{\hat{\pi}_t}(s, .)$ of the policy $\hat{\pi}_t$ currently under evaluation in the Policy Iteration scheme. The generalization property of the classifier makes it possible to consider only a sample of the states in $\mathcal{S}$.

At iteration t, CAPI first computes an empirical estimation of $Q^{\hat{\pi}_t}(s, a)$ (evaluation step). $Q^{\hat{\pi}_t}(s, a)$ is the average over K rollouts of length T starting at state-action pair (s, a), of the cumulative reward gathered by executing a in s and following $\hat{\pi}_t$ for T time steps. Considering a subset $\hat{\mathcal{S}}_t \subset \mathcal{S}$ of the state space, a training set made of pairs (s, a^*_s) with $s \in \hat{\mathcal{S}}$ and a^*_s the action such that $Q^{\hat{\pi}_t}(s, a^*_s)$ is significantly greater than $Q^{\hat{\pi}_t}(s, a)$ for all $a \neq a^*_s$ is built (if no such action exists w.r.t. the considered statistical test, no training sample built on s is considered). A classifier h_t is learned from this training set, and policy $\hat{\pi}_{t+1}$ defined over the whole state space $\mathcal{S}$ is learned with $\hat{\pi}_{t+1}(s) = h_t(s)$. By construction this policy approximately selects the action maximizing $Q^{\hat{\pi}_t}$ (policy improvement step) thus completing the Policy Iteration algorithm.

[3] For instance the 9×9 Go board, a smaller scale version of the Go game used in the literature [Gelly & Silver 2007], has more than 10^{170} states. In robotics, states are often continuous physical quantities (e.g. angle or acceleration of an arm). Large state spaces can also derive from complex sensory inputs such as visual perception.

The distinction between the prediction and the control problems is at the core of the approximate RL literature. The prediction problem aims at approximating value function $\hat{V}^\pi$ of a policy π from a set of sample transitions, where the approximation is meant in terms of the L_2 norm. The control problem aims at the (approximate) Q-Value (or equivalently the approximate policy $\hat{\pi}^*$), where the approximation is meant in terms of the L_∞ norm. The pitfall is that a low L_2 error of $\hat{V}^\pi$ does not necessarily result in a low L_∞ error of $\hat{Q}^\pi$, possibly manifested as a degradation of the performance from one iteration to the next in the Policy Iteration scheme. Some care must thus be exercized at the theoretical and algorithmic levels. For instance, CAPI selects $\hat{\mathcal{S}}_t$ to match the distribution of states visited by $\hat{\pi}_{t+1}$, thus minimizing the approximation error in regions of the state space that matter [Lagoudakis & Parr 2003b].

Another approximate RL algorithm, Fitted-Q Iteration [Ernst *et al.* 2003, Antos *et al.* 2007] uses regression to learn the Q-Value associated to each state-action pair. The algorithm can be thought of an approximate Value Iteration: at each iteration t, $\hat{Q}_t$ is learned by regression from the samples associating to each pair (s, a) the value $r + \max_{a'} \hat{Q}_{t-1}(s', a')$ (and $\hat{Q}_0$ is set to the null function). Other approaches in approximate RL with Supervised Learning combined the temporal difference principle with back-propagation on an Artificial Neural Network [Tesauro 1995] or Deep Neural Nets [Mnih *et al.* 2013].

2.3.2 RL with function approximation

Another way to scale up RL algorithms is to replace the look-up table representation of the value functions with a functional approximation. The most studied approaches rely on linear approximations, assuming a mapping from the state-action space (or the state space) onto the k-dimensional real-valued space. Formally, let $\phi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^k$ denote the feature mapping, then the Q-Value (or the V-Value) function is written as the scalar product of $\phi(s, a)$ with a parameter vector θ, to be identified.

The pitfall of linear approximations of the Q-Value in the previous model-free RL algorithms (section 2.2.2) is that they might diverge in the off-policy setting [Baird 1995], while this setting is appropriate to handle the exploration/exploitation trade-off or to reuse expensive data samples. Regarding the prediction problem (es-

timating the V-Value of a fixed policy), gradient descent algorithms (Gradient Temporal Difference (GTD) and Temporal Difference with gradient Correction (TDC), among other variants) have been proposed to ensure convergence even in the off-policy case [Sutton *et al.* 2009], where the gradient is that of the convex *mean square projected Bellman error*, to be minimized:

$$MSPBE(\theta) = ||V_\theta - L^\Phi T^\pi V_\theta||^2$$

with $V_\theta = \Phi\ \theta$, and Φ the $|\mathcal{S}| \times k$ matrix bearing $\phi(s_i)^T$ on the i-th line. Note that applying the Bellman operator $T^\pi V_\theta = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi V_\theta$ (with $\mathcal{R}_\pi$ and $\mathcal{P}_\pi$ defined as in section 2.2.1) on value function V_θ might yield a non-linear function of the feature space. Therefore operator L^Φ is used to achieve the orthogonal mapping of $\mathbb{R}^{|\mathcal{S}|}$ onto the feature space (defined by the column vectors of Φ), formally:

$$\forall f \in \mathbb{R}^{|\mathcal{S}|}\ L^\Phi f = \arg\min_{\theta'} ||f - V_{\theta'}||^2$$

Assuming the independence of features ϕ_i, one has:

$$L^\Phi = \Phi(\Phi^T\Phi)^{-1}\Phi^T$$

GTD thus aims at parameter vector θ such that the associated V-Value is the fixed point of the projected Bellman operator. Greedy-GQ extends GTD to the control problem [Maei *et al.* 2010].

The Least Square Temporal Difference (LSTD) [Bradtke *et al.* 1996] and its extension to control Least Square Policy Iteration (LSPI) [Lagoudakis & Parr 2003a] directly compute θ as follows:

$$\begin{aligned}
V_\theta - L^\Phi T^\pi V_\theta &= 0 \\
\Phi\theta - L^\Phi\left(\mathcal{R}_\pi + \gamma\mathcal{P}_\pi\Phi\theta\right) &= 0 \\
(\Phi^T\Phi)^{-1}\Phi^T\left(\mathcal{R}_\pi + \gamma\mathcal{P}_\pi\Phi\theta\right) &= \theta \\
\left(\Phi^T\Phi - \gamma\Phi^T\mathcal{P}_\pi\Phi\right)\theta &= \Phi^T\mathcal{R}_\pi \\
\left(\Phi^T\Phi - \gamma\Phi^T\mathcal{P}_\pi\Phi\right)^{-1}\Phi^T\mathcal{R}_\pi &= \theta
\end{aligned}$$

As shown by [Koller & Parr 2000] (and still assuming that features ϕ_i are linearly independent), θ is well defined except perhaps for finitely many γ. Note that in both GTD and LSTD, matrix Φ is never computed explicitly (since the approach aims at

large state spaces $\mathcal{S}$) and matrix $A = \left(\Phi^T\Phi - \gamma\Phi^T\mathcal{P}_\pi\Phi\right)$ is estimated from samples before being inverted.

Since A is a $k \times k$ matrix, its inversion is in $\mathcal{O}(k^3)$ ([4]) whereas the GTD algorithm is in $\mathcal{O}(k)$. However, LSTD is more sample efficient and converges after seeing less samples [Dann 2012]. Another algorithm, iLSTD offers a good trade-off between computational complexity and sample efficiency (but still with quadratic memory requirement) [Geramifard *et al.* 2007], which only modifies the matrix A on a small number of entries at a time.

2.4 RL vs. Evolutionary Algorithms

An alternative to gradient-based Direct Policy Search algorithms to solve control problems is to use gradient-free methods, and specifically evolutionary algorithms (EAs) [Igel 2003, Whiteson & Stone 2006, Whiteson *et al.* 2010b, Heidrich-Meisner & Igel 2009, Stulp & Sigaud 2013]. Evolutionary algorithms operate directly on the parametric space of the policies, iterating the following three steps:

1. Evaluation of a set of (parametric) policies, *in-silico* (using Monte Carlo simulations) or *in-situ* (using direct interaction with the environment);

2. Selection of the best policies according to their evaluation referred to as fitness (which can be any user-defined function, including the cumulative reward gathered along the trajectory; the fitness is possibly averaged over several trajectories; additional criteria such as the behavioral diversity of the controllers may also be taken into account to avoid a premature convergence of the EA to local minima;)

3. Generation of new candidate policies using variation operators on the selected policies.

Evolutionary algorithms are wide-spectrum gradient-free optimization algorithms, not specifically tailored to RL problems (with the notable exception of the hybrid algorithm Neat+Q that mixes both neuro-evolution and temporal difference

[4] The computational complexity can be reduced to $\mathcal{O}(k^2)$ if the inverse is incrementally computed from the samples using the Sherman-Morrison formula.

function approximation [Whiteson & Stone 2006]). For these reason, they can be slower to solving MDP problems (especially when considering noisy transition models [Whiteson *et al.* 2010b]) than previously presented RL algorithms.

Still in some cases, evolutionary algorithms can be an appropriate alternative to the approximate RL algorithms presented in section 2.3:
When considering a **finite horizon** ($H < \infty$), the recursive structure of value functions does not hold any more. The guarantee of a stationary optimal policy then vanishes. In fact, it is trivial to build an MDP where the agent has to take different actions in the same state depending on whether it reaches it in the first or in the last time step of the trajectory. Quite the contrary, EAs deal with any evaluation function of a controller trajectory, irrespective of whether it is based on a reward function or satisfies the Markovian assumption. As such, they do not depend on the Bellman optimality; in counterpart, their sample complexity is higher especially in the presence of an important noise in the state transitions [Whiteson *et al.* 2010b]. Specific heuristics, e.g. based on Bernstein races to early discard unpromising policies, however render distribution-based EAs such as Covariance Matrix Adaptation (CMA-ES) [Hansen & Ostermeier 2001] competitive with RL algorithms on some usual RL benchmarks [Heidrich-Meisner & Igel 2009].

Another case is that of **Partially Observable MDP** (POMDP) [Sondik 1971], which generalize MDPs to contexts where the agent state is uncertain, for instance when the agent is unable to directly observe the current state s_t and only gets some observation o_t probabilistically related to s_t. For instance, the *perceptual aliasing* phenomenon occurs when two different states result in the same sensor values. In such a situation, the agent can only maintain a belief vector on its current state, expressed as a probability distribution on the state space and depending on its current and past observations; the action selection depends on the belief vector. While exactly solving POMDPs is intractable for large state and action spaces [Kaelbling *et al.* 1998], some well founded approximate alternatives are available [Ross *et al.* 2011b]. The EA robustness with respect to moderate or high perception aliasing has been empirically demonstrated [Whiteson *et al.* 2010b].

A third case is when the RL problem is **too complex** and cannot be captured using linear approximations of the Q-Value (section 2.3) and derived policies. In

practice, feature engineering is often required to enable linear models to solve the problem at hand. However when feature engineering fails, one usually resorts to artificial neural networks. EAs enable the parametric optimization of the neural net weight vector for a fixed architecture; they can also evolve the architecture to adapt the complexity of the network to the difficulty of the problem [Stanley *et al.* 2009]. Note that the use of neural networks to tackle RL problems is not new: it was successfully combined with the temporal difference principle to build the TD-gammon player [Tesauro 1995]. More recently, the features based on a deep neural architecture have been exploited to build an Atari player [Mnih *et al.* 2013].

A comprehensive presentation of EA applications to control problems will be found in [Wiering & van Otterlo 2012] (chapter 10). Let us only present the algorithm that will be used in the experimental part of chapter 5, 6 and some experiments of chapter 7.

For a parametric class of policies π_θ with $\theta \in \Theta \subset \mathbb{R}^D$ (for instance, θ can be the parameters of a neural network), new policies are generated by perturbing the best previous policy according to the $(1+\lambda)$-ES evolutionary strategy algorithm [Schwefel 1981]. Formally, a Gaussian distribution $\mathcal{N}(\theta_t, \Sigma_t)$ on Θ of center θ_t and covariance matrix Σ_t is maintained at each generation t. From this distribution, λ policies are drawn and evaluated, from which the best among the generated policies are retained and used to update θ_t, defining θ_{t+1}. We adapt the covariance matrix as in [Auger 2005], by letting $\Sigma_t = \sigma_t * I$, where I is the identity matrix. Parameter σ_t is augmented/reduced according to the $1/5^{th}$ rule: If the number of sampled policies performing better than θ_t is more (resp. less) than $1/5^{th}$, the radius is increased (resp. decreased) with constant tuned on the target unit sphere function [Auger 2005].

2.5 Summary and discussion

This chapter has presented some representative RL algorithms and briefly discussed their strengths – the requirements for their optimality guarantees – and their limitations, specifically regarding their scalability w.r.t. the size of the state and action spaces.

The focus of the manuscript (chapters 5 to 7) is to learn the RL objective function by interaction with the expert; this objective function will be exploited using some of the presented RL and EA algorithms to actually build the controllers.

Our primary assessment criterion however will be the number of interactions with the human user needed to reach a satisfactory performance. RL algorithms will only be considered based on their ability to deliver a solution on the basis of the learned objective function: Typically Policy Iteration algorithms will be used to deal with discrete and small-size state space; LSPI/TDC algorithms will be used to deal with continuous state spaces; finally, EAs will be used when considering a non-linear policy return estimate w.r.t. the trajectory representations, and/or when LSPI is found to fail.

Chapter 3

Learning from Demonstrations

Foreword

The general RL framework was presented in the previous chapter together with related algorithms, depending on whether the transition model $\mathcal{P}(.|s,a)$ is directly available, or only known through samples (s,a,r,s'). This chapter examines the case where the reward function is not naturally present in the environment, or does not comply with the MDP setting. The cases where rewards do not comply with the standard setting are first discussed. The chapter then presents a general overview of the learning from demonstration approaches.

3.1 Reward functions

In quite a few contexts, the reward function does not satisfy the MDP assumptions. A first case is when the reward is defined or interpreted from the user's feedback; this reward might be related to the whole trajectory or a sub-trajectory, rather than to the last state-action pair. The case where the agent interacts with a human being, whose feedback is exploited as an external reward, will be described in next chapter.

Beyond the direct communication with the user, it might be hard to define a sufficiently informative reward function for complex RL tasks. As mentioned, if the reward is too scarce (e.g. only the goal state is rewarded), the RL problem is too hard and only poor solutions are found. Additional prior knowledge in form of additional rewards, for instance highlighting subgoals, needs be provided, possibly iteratively, along some so-called reward shaping process [Ng *et al.* 1999].

For many tasks where defining an informative reward function is the primary RL difficulty, providing trajectories and demonstrating a solution behavior can overcome

the difficulty. Let us illustrate this claim on two representative problems.

The first problem is concerned with finding an optimal route between locations A and B. Assuming that many different routes exist (one being the fastest one, another one being the cheapest one, a third one being the one with less traveling time variance), the agent actually faces a multi-objective RL problem [Vamplew *et al.* 2009] where the best solution depends on the user's trade-off among the different criteria: slight modifications on the weights of these criteria will change the optimal route returned by the agent. The user might engage in a tedious trial and error process to adjust the weights until the agent returns a satisfactory solution; however beyond certain limits, related to the number of criteria and complexity of the tasks, trial and error processes are no longer applicable. An elegant alternative is to let the user demonstrate some routes to his liking, and let the system automatically infer a reward function accounting for these preferences. The learned reward function is then available to solve future instances of the task at hand within the classical RL setting.

The second problem is concerned with peeling a vegetable for a domestic robot. This task requires a large action space (the required level of dexterity implies a fine-grained discretization of the continuous search space); accordingly, any random exploration will unlikely lead to the end goal[1]. Additionally, letting the robot randomly explore its state-action space while it holds a peeler likely raises safety concerns. To guide the exploration of the robot, the expert might attach rewards to some intermediate states that she deems relevant for peeling the vegetable, thereby defining subgoals. Such subgoals might however be ill-suited to completing the task at hand, due for instance to the gap between the human and the robot motor degrees of freedom (DoF). In such cases, an alternative would be for the expert to take direct control of the robot arm and perform the desired task with the robot own DoF. Providing a demonstration of the task will benefit the robot in more than one way: It gives a continuum of subgoals linking the initial state to the end state, it guides the exploration process by visiting a path of interesting regions of the state-action space and it addresses to some extent the safety

[1] Since the action space is large and the number of action sequences increases exponentially with their length, the number of sequences reaching the end goal will likely be a negligible fraction of all sequences.

problem if the robot is constrained to stay in the vicinity of the demonstration path.

These example tasks are representative of two classes of problems.

The first class includes RL problems which are multi-objective in a high-level cognitive sense, where the known criteria refer to some abstract concepts in the state and action spaces[2] but the desired trade-off between these criteria is unknown (by definition of non-trivial multi-objective optimization problems, the relevant criteria cannot be all simultaneously optimized). Examples of this first class of problems include the driving task [Abbeel & Ng 2004] and the grocery checkout task [Jain *et al.* 2013].

The second class includes low-level tasks involving a complex dynamics system, where either i) the end goal can hardly be manually specified through a reward function (this case is illustrated by the helicopter aerobatics in [Coates *et al.* 2008], where the goal is exactly to display a target behavior); or ii) the end goal can be thought of as a needle in the haystack in a large state-action space, which can hardly be visited through random exploration if no intermediate milestones are defined (this case is illustrated by the swinging up a pole task with a robot arm in [Atkeson & Schaal 1997b]).

3.2 Learning from demonstrations

When a control problem can hardly be specified through defining a reward function, preventing the direct application of RL algorithms, a viable alternative is to proceed by demonstrating (quasi) optimal behaviors. Searching a policy amenable to reproduce the provided demonstration and generalize the taken actions under small fluctuations of the initial conditions is called Learning from Demonstrations (LfD), also referred to as Behavioral Cloning [Bain & Sammut 1995, Pomerleau 1989], apprenticeship learning or imitation learning [Calinon *et al.* 2007, Billard & Grollman 2013], or Inverse Reinforcement Learning (IRL) [Ng & Russell 2000, Abbeel & Ng 2004].

Formally, [Billard & Grollman 2013] distinguishes two trends in LfD, detailed in

[2] These concepts can be thought of as options in MDPs [Sutton *et al.* 1999]. At execution, the high level actions will indeed unfold through a lower level sensory-motor feedback loop.

the next two sections: learning high-level action composition (section 3.4) and low-level learning of individual motions (section 3.3). Albeit [Billard & Grollman 2013] do not mention the under-specification issue arising from the multi-objective nature of the target goal in the first trend, we believe that this issue is a key motivation for the LfD approach: the definition of unambiguous milestones would be less difficult if all ways of reaching the goal were equally appropriate.

After [Ratliff 2009, Ross 2013], the main difference between the early approaches of Behavioral Cloning (BC) [Bain & Sammut 1995, Pomerleau 1989] and the later approaches to Inverse Optimal Control (IOC) [Abbeel & Ng 2004, Ratliff *et al.* 2006, Ziebart *et al.* 2008] can be understood in term of generality.

The reward function, associated with an RL or planning algorithm, can be used to solve new instances of the task at hand – possibly dealing with new states provided that the new state space "sufficiently" overlaps the set of demonstrated states. In this sense, IOC thus yields some generality property, whereas BC does not apply in general if the agent faces new states. A possibility, at the core of Transfer Reinforcement Learning [Bou-Ammar *et al.* 2013] is to define representations and features mapping the source and target state spaces onto a single space over which the reward function is defined. Conditionally to these features, Behavioral Cloning would also be able to generalize to novel instances; however this extension requires the features to capture the long-term effects of the actions in order to model the desired policy in terms of reactive behavior.
It might be argued that IOC is specifically relevant for the first, multi-objective-like, type of RL problems, where the reward function is interpretable since we are dealing with high-level concepts, and the generalization property is important. Quite the contrary, reactive systems trained by BC can be sufficient for learning lower-level specialized tasks [LeCun *et al.* 2006].

Algorithmically, [Schaal 1999] distinguishes three approaches to LfD. The first one is Behavioral Cloning or Direct Policy Learning; the demonstrations are used to define a supervised learning problem, where the goal is to map the state space onto the action space according to the training samples provided by the demonstrated trajectories. BC thus captures reactive behaviors, mapping a state onto an action;

a representative application thereof is the use of camera images to train a neural network and drive a car [Pomerleau 1989, LeCun *et al.* 2006]. The main limitation of BC is due to the fact that errors are independent in supervised learning, whereas they have a cumulative effect in RL; formally, the state distribution of the agent (test distribution) might deviate from the demonstration ones (more in section 3.5.1). Along the same lines, BC does not uncover the expert's intent (through learning a reward or utility function), limiting the self-improvement abilities of the learner.

A second approach extends the former one by incorporating task knowledge, specifying for instance an end goal, and letting the robot improve upon an initial policy obtained by pure imitation.

The third approach, referred to as Model-based Imitation learning, learns a forward model (analogous to a transition model) using the demonstrations and then applies RL together with the definition of the goal state to find policies reaching the end goal. The last two approaches differ from IOC as they do not aim at learning a general reward function.

We shall see in chapter 4 that some approaches have been deployed to extend LfD using **expert's feedback**, with two main motivations in mind: overcoming the under-optimality of the expert's demonstrations[3], and preventing the brittleness of learned policies where they deviate from the demonstrated trajectories[4].

[3] [Coates *et al.* 2008] used generative models to learn the optimal trajectory, cast as a hidden sequence producing the observed sub-optimal demonstrations. However, to which extent the actual demonstrations can be far from the optimal one remains an open question. [Kim *et al.* 2013] proposes an alternative solution, by formulating a quadratic optimization problem akin regularized LSPI, where the regularization is based on both the demonstrations and the reward function (see section 2.3). The expert can add new demonstrations in each iteration of the algorithm, and he can also adjust a trade-off parameter between solving the initial RL problem and imitating the demonstrations, reflecting his confidence in the optimality of the demonstrations. [Cakmak & Thomaz 2012] studied more direct interactions between the expert and the learning agent, where the latter can ask questions such as "is this feature relevant for the task" or "how to continue from here".

[4] Even if the provided demonstrations are optimal, if the learner cannot learn a perfect mapping from states to actions, he is bound to choose at some point a different action than what the expert would have chosen. In this case, he may drift from the distribution of states seen during learning, resulting in unpredictable behaviors. [Ross & Bagnell 2010] propose an interactive algorithm to overcome this problem by querying the expert for demonstrations as the agent learns to mimic

3.3 Low-level learning of complex skills

This section focuses on tasks where the agent should reproduce a specific trajectory and/or when reaching the target state requires to discover a long and complex sequence of actions, beyond the reach of standard RL algorithms. As said, Behavioural Cloning casts LfD as a supervised learning problem [Bain & Sammut 1995, Pomerleau 1989]. However, when dealing with few or sub-optimal demonstrations, a critical issue is to build local task-specific models and to exploit the available domain knowledge to produce high quality controllers.

For instance, let us consider the pole balancing task, aimed at maintaining a pole at an angle $\theta = 0$ with a robot arm [Schaal 1996, Atkeson & Schaal 1997a]. [Atkeson & Schaal 1997b] elaborates on this problem, tackling the additional problem of swinging up the pole when starting from position $\theta_0 = -\pi$. Even with full knowledge of the state (including θ, the pole angle, $\dot{\theta}$ the angular velocity, x the horizontal position of the arm and $\dot{x}$ its velocity), standard RL fails to solve the problem.

The expert's demonstrations are used to i) learn a (local) transition model $s_{t+1} = f(s_t, a_t)$ according to the environment dynamics ii) define a reward function $\mathcal{R}(s_t, a_t, t) = -||s_t - s_t^*||^2 - ||a_t||$ that penalizes deviations from the target trajectory state and too large actions and iii) seed the planer with an initial plan.

LfD succeeds on this task of swinging up the pole (considering both parametric and non-parametric settings). However, the approach fails when the difficulty increases, and the task requires to 'pumping' the pole twice before swinging it up. The authors blame the failure on the complex dynamics (as the arm reaches the upward position with more energy), resulting in a more inaccurate transition model. The solution proposed, based on domain knowledge, considers two sub-tasks (swinging up and balancing the pole) and lets the algorithm select when to switch from a subtask to the other.

When considering long and sub-optimal trajectories, another issue must be addressed, the fact that the trajectories are not aligned. A generative model is learned to aggregate and generalize the actual trajectories, assuming that each

him, so as to have policies that learn to mimic the expert on their own induced distribution of states.

demonstration is a perturbed observation of the (unobserved) perfect trajectory. In [Calinon & Billard 2005], the generative model is sought as a Hidden Markov Model. In [Coates *et al.* 2008], the generative model is built by alternate optimization of i) the HMM generalizing the demonstrations, and ii) the transition model.

Prior knowledge can be incorporated in terms of constraints, by defining a function $\rho(.)$ and target values $\rho_t = \rho(s_t^*)$. In the helicopter flight experiment for instance, ρ represents the fact that the helicopter should stay in the same position along a manouver. The experimental validation of the approaches lead to behaviors that often improve on the expert's demonstrations; the price to pay regards the complexity of the generative model, and the extensive prior knowledge required to obtain these results.

3.4 High-level Multi-objective Problems

The previous approaches focus on inferring the ideal (aggregated, unobserved) demonstration and possibly the transition model. Let us now consider how to infer a reward function from the demonstrations.

3.4.1 Learning a reward from an optimal trajectory

Inverse Reinforcement Learning [Russell 1998] aims at characterizing the reward function from the expert's demonstration. As previously discussed, extracting the reward function (a.k.a. the expert's intent) yields to better generality properties than merely mimicking the agent's behavior. The critical issue of IRL however is the indeterminacy of the reward identification problem; for instance, a constant reward function makes every policy optimal.

[Ng & Russell 2000] characterizes the sought rewards through linear constraints, and further impose i) to maximize the difference between the agent's policy and any other policy (the so-called margin); ii) to maximize the reward sparsity (with respect to the L_0 or L_1 norm).

Let us first assume that the user's policy π^* is fully known. With same notations as in section 2, policy π^* is optimal under $\mathcal{R}$ if and only if for any policy π:

$$(\mathcal{P}_{\pi^*} - \mathcal{P}_\pi)(I - \gamma\mathcal{P}_{\pi^*})^{-1}\mathcal{R}_{\pi^*} \geq 0 \tag{3.1}$$

this immediately derives from the Bellman optimality property

$$(\forall s \in \mathcal{S})(\forall a \in \mathcal{A}) \;\; Q^{\pi^*}(s, \pi^*(s)) \geq Q^{\pi^*}(s, a)$$

Although equation (3.1) must hold for any policy π, it is sufficient to check that it holds for every state[5].

These constraints give rise to a linear optimization problem, aimed at maximizing the following expression in the state space [Ng & Russell 2000]:

$$\begin{aligned} &\underset{\mathcal{R}}{\text{maximize}} && \sum_{s \in \mathcal{S}} Q^{\pi^*}(s, \pi^*(s)) - \max_{a \in \mathcal{A} \setminus \{\pi^*(s)\}} Q^{\pi^*}(s, a) - \lambda ||\mathcal{R}||_1 \\ &\text{subject to} && (\mathcal{P}_{\pi^*} - \mathcal{P}_{\pi})(I - \gamma \mathcal{P}_{\pi^*})^{-1} \mathcal{R}_{\pi^*} \geq 0 \\ &\text{and} && |\mathcal{R}| \leq R_{\max} \end{aligned} \tag{3.2}$$

The reward function thus maximizes the margin of the optimal actions and its L_1 sparsity , with λ a hand-tuned parameter trading-off between the maximization of the margin of the optimal actions and the sparsity regularization (3.2).

Finally, when the policy is unknown and the user's behavior is only known from a set of trajectories, [Ng & Russell 2000] propose an iterative optimization scheme as follows. Starting with an initial pool made of a single random policy, the learning step alternates between: i) finding a reward that maximizes the difference between the cumulative gain of the user's average trajectory $\bar{\mu}$ and the trajectories of the pool; and ii) for this reward, finding a policy with better cumulative gains that $\bar{\mu}$.

The approach is very similar in spirit with the LfD algorithms. The main difference is that there is no additional goal of recovering the policy that matches the behaviour of the agent, encoded by the vector $\bar{\mu}$.

3.4.2 Matching the feature expectation of the demonstrations

[Abbeel & Ng 2004, Ziebart *et al.* 2008] aim at finding some policy π that behaves as well as the unknown policy π^* of the expert. The two proposed approaches rely on a mapping ϕ from the state space $\mathcal{S}$ onto $[0, 1]^k$, defining k descriptive features. To each policy π is associated its (discounted) feature expectation, defined as

$$\mu_\pi = \mathbb{E}_{a_t \sim \pi(s_t)} \left[\sum_{t=0}^{T} \gamma^t \phi(s_t) \right]$$

[5] More precisely, for all $(|A| - 1)$ sub-optimal actions yielding $|S| \times (|A| - 1)$ linear constraints in $\mathcal{R}$.

with $\mu_\pi \in \mathbb{R}^k$. [Abbeel & Ng 2004, Ziebart *et al.* 2008] proceed by searching for a policy matching the feature expectation μ^* of the demonstrations, i.e. searching π such that $||\mu^* - \mu_\pi||_2 \leq \varepsilon$. Let us assume that the reward function is linear w.r.t. the ϕ representation, i.e. $r(s) = \langle w, \phi(s) \rangle$, then by linearity it comes $V(\pi) = \langle w, \mu_\pi \rangle$. It follows that if μ^* and μ_π are distant by at most ε, their V-value also differ by at most ε.

The first algorithm presented by [Abbeel & Ng 2004] is a max-margin algorithm, as follows:

1. Let π_0 be an initial policy and μ_0 be its feature expectation.
2. Iterate:
   (a) Find the reward vector $\mathbf{w}_t$ (with $||\mathbf{w}_t||_2 = 1$) maximizing the minimum of $\langle \mathbf{w}, (\mu^* - \mu_i) \rangle$ for i ranging in $0 \ldots t-1$;
   (b) If $\langle \mathbf{w}_t, (\mu^* - \mu_i) \rangle$ is less than ε, then stop, with $n = t$;
   (c) Else build the optimal policy π_t from reward $\mathbf{w}_t$ (using standard RL algorithm);
   (d) Estimate μ_t from π_t.
3. Output the set of policies $\pi_0 \ldots \pi_n$.

By construction, this set of policies is such that for all reward $\mathbf{w}$, there exists a policy π_i such that $\langle w, (\mu^* - \mu) \rangle \leq \varepsilon$; in other words the set of policies $\pi_0, \ldots \pi_t$ samples all policies with feature expectation ε-close to μ^*. The expert, by browsing this set can find the closest policy to π^* for the true (hidden) reward w^*. An alternative is to consider a mixed policy $\tilde{\pi}$ with mixing parameters $\{\lambda_i, i = \{0, \ldots, n\}, \lambda_i \geq 0, \sum \lambda_i = 1\}$, which selects policy π_i with probability λ_i at the initial state and follows it for a whole trajectory. By the linearity of expectation it comes $\tilde{\mu} = \sum_{i=0}^{n} \lambda_i \mu_i$. Since μ^* is distant of at most ε from the convex hull of $\{\mu_i, i = \{0, \ldots, n\}\}$ then $||\mu^* - \tilde{\mu}||_2 \leq \varepsilon$, and hence policy $\tilde{\pi}$ performs as well as π^* under any reward w (up to ε).

The key steps are i) searching for the reward $\mathbf{w}_t$, which maximizes the margin between μ^* and the feature expectation of all previous policies; and ii) finding the

optimal policy π_t corresponding to reward $\mathbf{w}_t$. This alternate optimization of a disadvantageous reward, and the associated policy is formalized as an adversarial game by [Syed & Schapire 2007]. Algorithmically, the search for $\mathbf{w}_t$ is achieved using a RankingSVM approach [Joachims 2002]:

$$\begin{aligned} \mathbf{w}_t = \min \quad & ||\mathbf{w}||_2 \\ s.t. \quad & \langle \mathbf{w}, (\mu^* - \mu_i) \rangle \geq 1, \text{ for all } i \in \{0, \dots, t-1\} \end{aligned}$$

The second algorithm proposed by [Abbeel & Ng 2004] (which will be used for some of the experiments in chapter 6) is a projection algorithm, iteratively computing the residual of the expert feature expectation μ^* with respect to the previous $\mu_0, \dots, \mu_t$ and computing $\mathbf{w}_t$ accordingly. Formally, let $\bar{\mu}_0 = \mu_0$ and $\mathbf{w}_1 = \mu^* - \mu_0$; at each iteration $t \geq 2$, let $\bar{\mu}_{t-1}$ be the projection of μ^* onto the line $(\bar{\mu}_{t-2}, \mu_{t-1})$, and $\mathbf{w}_t$ is set to $\mu^* - \bar{\mu}_{t-1}$. As $\bar{\mu}_{t-1}$, the projection of μ^*, is a convex combination of the $\mu_i, i = 0 \dots t-2$, the convex hull[6] of the μ_i gets closer to μ^*, until satisfying $||\mu^* - \bar{\mu}_n||_2 \leq \varepsilon$.

3.5 LfD with a human in the loop

As mentioned, the expert might want to provide additional information to a LfD agent for at least two reasons: because the agent might be induced to explore different state distributions as the one visited by the demonstrator, referred to as *distribution drift*, and because demonstrations are sub-optimal.

3.5.1 Distribution drift

[Ross & Bagnell 2010, Ross *et al.* 2011a] show how imitation learning can be studied in terms of no-regret online learning, as follows.

Let $d_\pi(s)$ denote the average number of visits to state s along T-long trajectories controlled by π. As the cost or reward function is unknown in imitation learning, one usually aims at finding a policy minimizing some surrogate loss expectation

[6] [Ziebart *et al.* 2008] note that the projection algorithm is still ambiguous as many different mixtures of policies satisfy the matching, and propose instead to employ the principle of maximum entropy to resolve the ambiguity.

($\pi = \arg\min_\pi \mathbb{E}_{s\sim d_\pi}[\ell_\pi(s)]$), where loss ℓ can be the 0-1 loss between policy π and the expert policy π^*. By exploiting the demonstration data using supervised learning, one builds a policy minimizing the loss in expectation. The critical issue is that, due to the sequential nature of the RL tasks, errors will accumulate along time: the state distribution tends to drift away from the training (demonstration) distribution, quadratically w.r.t. the time horizon. Formally, [Ross & Bagnell 2010] proves the following result:

Theorem: Let π be such that $\mathbb{E}_{s\sim d_{\pi^*}}[\ell_\pi(s)] = \varepsilon$, where ℓ upper bounds the 0-1 loss. Then for a cost function bounded in $[0, 1]$, the expected cumulative cost $J(\pi)$ over T-long trajectories is upper bounded by $J(\pi^*) + T^2\varepsilon$.

Proof. Let distributions $d_1 \dots d_T$ denote the state distributions at time step $1, \dots T$ when following π, and let these distributions be further decomposed into d_t^+ and d_t^-, where d_t^+ denotes the distribution of states at time t such that π executed the same actions as π^* up to step t, and d_t^- the complementary distribution (π made at least one mistake w.r.t. π^* before step t). Let p_t be the probability that π makes no mistake in the t first steps; simple calculations show that $d_t = p_{t-1}d_t^+ + (1 - p_{t-1})d_t^-$. Distributions under policy π^* can be decomposed in the same way: $d_t^* = p_{t-1}d_t^{*+} + (1 - p_{t-1})d_t^{*-}$. Note that the decomposition of d_t^*, the state distribution when following π^* shares the same distribution d_t^+ when π and π^* agrees (and they do with the same probability p_{t-1}). And d_t^{*-} is the complementary distribution.

Let $C_{\pi,t}$, $C^*_{\pi,t}$, $C^+_{\pi,t}$ and $C^-_{\pi,t}$ be the cost of executing π under the distribution d_t, d_t^*, d_t^+ and d_t^- respectively. From the decomposition of these distributions it follows that:

$$C_{\pi,t} = p_{t-1}C^+_{\pi,t} + (1 - p_{t-1})C^-_{\pi,t}$$

Letting ε_t^+ be the probability that π chooses a different action than π^* under d_t^+, it comes[7]:

$$C^+_{\pi,t} \leq C^+_{\pi^*,t} + \varepsilon_t^+$$

Using the decomposition of d_t^*, it comes $C^*_{\pi^*,t} = p_{t-1}C^+_{\pi^*,t} + (1-p_{t-1})C^{*-}_{\pi^*,t}$ and since

[7] Since π incurs the same cost as π^* with probability $1 - \varepsilon_t^+$, and its cost is bounded by 1 otherwise.

costs are non negative, it follows that:

$$p_{t-1}C^{+}_{\pi^*,t} \leq C^{*}_{\pi^*,t}$$

Finally, the cost of policy π at time t is upper bounded by:

$$C_{\pi,t} \leq C^{*}_{\pi^*,t} + p_{t-1}\varepsilon^{+}_{t} + (1 - p_{t-1})$$

On the other hand, the probability of making at least one mistake under π after t time steps is $(1 - p_t) = \sum_{k=1}^{t} p_{k-1}\varepsilon^{+}_{k}$, by considering the union of the disjoint events *making the first mistake at step* k (which happens with probability p_{k-1} of not making a mistake in the first $k-1$ steps and then choosing a different action than π^* under d^{+}_{k}, with probability ε^{+}_{k}). Hence we have:

$$p_{t-1}\varepsilon^{+}_{t} + (1 - p_{t-1}) = (1 - p_t)$$

Denoting ε_t the probability of π and π^* disagreeing under training distribution d^{*}_{t}, it comes:

$$\mathbb{E}_{s \sim d_{\pi^*}}[\ell_\pi(s)] = \varepsilon \geq \frac{1}{T}\sum_{t=1}^{T}\varepsilon_t$$

(where the equality holds iff ℓ is the 0-1 loss). On the other side, using the decomposition of d^{*}_{t} it follows that $\varepsilon_t \geq p_{t-1}\varepsilon^{+}_{t}$. Hence:

$$(1 - p_t) \leq \sum_{k=1}^{t}\varepsilon_k \leq T\varepsilon$$

Finally $C_{\pi,t} \leq C^{*}_{\pi^*,t} + T\varepsilon$, which yields upon summation over T:

$$J(\pi) \leq J(\pi^*) + T^2\varepsilon \qquad \square$$

This upper bound is tight [Kaariainen 2006, Ross & Bagnell 2010].

New algorithms are proposed in [Ross & Bagnell 2010] to contain the propagation of errors. The first one, called forward training, iteratively trains a non stationary policy π_t on a training set generated after π_{t-1}, where the initial policy is set to the demonstrated one ($\pi_0 = \pi^*$). Formally, at time step $t = 1 \ldots T$, policy $\hat{\pi}_t$ is learned on training pairs $(s, \pi^*(s))$ with s sampled from state distribution $d^{t}_{\pi_{t-1}}$. Policy π_t is defined by executing π_{t-1} except at time t where it executes $\hat{\pi}_t$. The main result is that policy π_T built by forward training algorithms incurs a cost $J(\pi) \leq$

$J(\pi^*) + uT\varepsilon$, where $u = \sup_{a,s,t\in\{1\dots T\},d^t_\pi(s)>0}[Q^{\pi^*}_{T-t+1}(s,a) - Q^{\pi^*}_{T-t+1}(s,\pi^*(s))]$ is the extra cost when following an action $a \neq \pi^*(s)$ in an initial state s before branching back to π^*. In the worst case u is in $\mathcal{O}(T)$ (making some initial mistake yields a penalty at all further time steps). However if mistakes can be recovered sufficiently quickly, then the forward training algorithm incurs a linear extra cost. For instance, if the cost function is the 0-1 loss, then u cannot be bigger than 1, and the T-step cost difference between the learned policy and π^* is in $\mathcal{O}(T)$.

As a drawback, this first algorithm might not be practical for large time horizons as it needs to sequentially learn T different policies. An alternative is thereby presented in [Ross & Bagnell 2010], referred to as SMILe (*Stochastic Mixing Iterative Learning*) algorithm. Likewise, SMILe iteratively trains policies $\pi_0 \dots \pi_N$, with $\pi_0 = \pi^*$, generating sample states from $d^t_{\pi_i}$ that will be used to learn $\hat{\pi}_{i+1}$. Contrary to the forward training, π_{i+1} is stationary; it stochastically mixes π_i and $\hat{\pi}_{i+1}$ as follows:

$$\pi_{i+1} = (1-\alpha)^{i+1}\pi^* + \alpha\sum_{k=1}^{i+1}(1-\alpha)^{k-1}\hat{\pi}_k, \text{with } \alpha \in (0,1)$$

In other words, π_{i+1} executes $\hat{\pi}_{i+1}$ with probability $\alpha(1-\alpha)^i$ and π^* otherwise. This anytime algorithm can return policy $\dot{\pi}_n$ by eliminating the expert policy π^* from the mixture and re-normalizing accordingly at any point in time, with cost on T-long trajectories upper-bounded by $J(\dot{\pi}_n) \leq J(\pi_n) + (1-\alpha)^n T^2$. A further research perspective is to reuse previous data to reduce the sample complexity (queries to $pi^*(s)$ as policies slowly change at each iteration.

The DAgger algorithm (*Dataset Aggregation*, [Ross *et al.* 2011a]) aims at reducing the sample complexity through reusing all samples.

The algorithm is analyzed in terms of online learning: at the i-th iteration, policy π_i is produced and the environment picks up a loss $L_i(\pi) = \mathbb{E}_{s\sim d_{\pi_i}}[\ell_\pi(s)]$, assessing some policy π w.r.t. the target policy π^* under the state distribution induced by π_i. Choosing $\pi_{i+1} = \arg\min_\pi \mathbb{E}_{s\sim\mathcal{D}}[\ell_\pi(s)]$ is similar to selecting $\pi = \arg\min_\pi \sum_{k=0}^{i} L_k(\pi)$ (or at least an empirical estimate of it) since $\mathcal{D}$ is the union of all samples generated by the previous policies. By doing so, the algorithm implements a Follow-The-Leader no-regret strategy on convex losses [Kakade & Tewari 2008]. It presents same guarantees as for the previous algorithms, with linear or quasi-linear error in T and ε if the problem has recoverability prop-

erties from previously made mistakes.

The good behavior of DAgger compared to SMILe is experimentally demonstrated in [Ross *et al.* 2011a], on cases where supervised learning from samples generated from the expert policy fails to learn a satisfactory policy, even when increasing the learning set size.

3.5.2 Under-optimal Demonstrations

[Kim *et al.* 2013] investigates a hybrid setting where a reward function is available as well as (possibly bad) expert demonstrations. The proposed *Approximate Policy Iteration with Demonstration* (APID) algorithm achieves a trade-off between standard RL and LfD. Ideally, APID gets the best of both worlds: leveraging expert data (even though scarce or not optimal, where LfD alone would fail) to provide API with insights about the structure of a good policy.

APID is provided an expert sample $\mathcal{D}_E = \{(s_i, \pi_E(s_i))\}_{i=1}^m$, where $\pi_E(s_i)$ the action performed in s_i by the expert policy π_E, and a set of transitions $\mathcal{D}_{RL} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^n$ (where action a_i is selected along an off-policy setting, section 2.2.2). In the k-th iteration, the Q-Value of the policy π_k is computed using the fixed point Bellman operator, together with constraints reflecting the expert sample; the respective importance of the Bellman error reduction and the constraints is controlled by a user-specified trade-off parameter α. For low values of α the stress is put on the Bellman error (still with a faster convergence rate than LSPI, in the experiments) and for high values of α APID reaches the same performance as state-of-art LfD algorithms.

Formally, the expert sample induces large margin constraints (for non-expert action $a \neq a_i$, $Q(s_i, a_i)$ should be higher than $Q(s_i, a)$), the violation of which is added as penalty term to the optimization objective of minimizing the Bellman error. When the exact Bellman operator T^π is not known, replacing it with its empirical estimate $\hat{T}^\pi Q(s_i, a_i) \stackrel{\Delta}{=} r_i + \gamma Q(s'_i, \pi(s'_i))$ induces a biased estimate of the Bellman error [Antos *et al.* 2008]. One thus rather minimizes the Regularized Projected Bellman error [Farahmand *et al.* 2008] by solving the following nested

optimization problem:

$$\hat{Q} = \underset{Q \in \mathcal{F}^{|\mathcal{A}|}}{\arg\min} ||Q - \hat{h}_Q||_n^2 + \lambda J^2(Q) \tag{3.3}$$
$$\text{s.t. } \hat{h}_Q = \underset{h \in \mathcal{F}^{|\mathcal{A}|}}{\arg\min} ||h - \hat{T}^\pi Q||_n^2 + \lambda_h J^2(h)$$

Algorithmically, let $\phi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^p$ denote the mapping from the state-action space onto p features. Letting h and Q be linear forms on $\mathbb{R}^p$ ($h(s,a) = \phi^T(s,a)\ \mathbf{u}$ and $Q(s,a) = \phi^T(s,a)\ \mathbf{w}$), with regularization term $||u||^2$, then the optimum of the above is reached for:

$$\mathbf{u}^*(\mathbf{w}) = (\mathbf{\Phi}^T\mathbf{\Phi} + n\lambda_h\mathbf{I})^{-1}\mathbf{\Phi}^T(\mathbf{r} + \gamma\mathbf{\Phi}'\mathbf{w})$$

Where $\mathbf{r} = (r_1, \ldots, r_n)$ is the reward vector, $\mathbf{\Phi}$ and $\mathbf{\Phi}'$ the feature matrices of the state-action pairs corresponding to distributions $\mathcal{D}_{RL}$ and to the distribution associated to the policy under evaluation, respectively.

Combining equation (3.3) with constraints defines the constrained quadratic problem of APID:

$$\mathbf{w}^* = \underset{\mathbf{w} \in \mathbb{R}^p, \xi \in \mathbb{R}_+^m}{\arg\min} ||\mathbf{\Phi}\mathbf{w} - \mathbf{u}^*(\mathbf{w})||^2 + \lambda \mathbf{w}^T\mathbf{w} + \frac{\alpha}{m}\sum_{i=1}^{m}\xi_i \tag{3.4}$$
$$\text{s.t. } \phi^T(s_i, \pi_E(s_i)) \cdot \mathbf{w} - \max_{a \in \mathcal{A} \setminus \pi_E(s_i)} \phi^T(s_i, a) \cdot \mathbf{w} \geq 1 - \xi_i, \text{ for all } (s_i, \pi_E(s_i)) \in \mathcal{D}_E$$

The solution thereof yields the Q-Value $\hat{Q}^{\pi_k}$, defining policy π_{k+1}; the process is iterated and stops after a prescribed number of iterations.

The main difficulty is that reducing the Bellman error and satisfying the constraints can be antagonistic objectives; it might be difficult to adjust the trade-off. In practice the authors advocate increasing $\mathcal{D}_{RL}$ at each iteration k, e.g., by running π_k with some additional exploration and adapting accordingly the regularization parameters and α.

The experimental validation of APID, comparatively to a vanilla LfD and DAgger, considers two problems, the vehicle brake control simulation [Hester *et al.* 2012] and a real robot navigating in a simple environment. In both LfD and DAgger, the supervised learning algorithm is a random forest. On the first problem, APID significantly outperforms both LfD methods when considering sub-optimal or scarce expert demonstrations; in the ideal case (abundant and optimal expert demonstrations), APID performs as well as DAgger. In all cases, APID converges faster than

LSPI with less variance among the runs. On the second problem, when considering a single demonstration, vanilla LfD failure is blamed on the distribution drift (section 3.5.1); LSPI fails to reach the target position in the first iterations due to an inefficient exploration; APID uses the available expert data to prune far-off regions of the map and generate better ε-greedy policies.

3.6 Summary and discussion

This chapter investigates how to overcome the lack of reward functions complying with the standard RL MDP setting. The need for new approaches has been illustrated on two types of applications: achieving a fine-grained expert skill (where the random exploration of the state action space is unlikely to provide any good result soon), and finding appropriate behaviors in a high-level multi-objective optimization setting (where the expert might find it hard to adjust the reward by trials and errors, to make the learned policy produce the desired behavior).

In such contexts, a demonstration of the desired agent behavior can be used to define an optimization criterion and/or guide the exploration of the state-action space.

Expert demonstrations can be exploited in two main ways. The simplest one is to mimic the expert behavior, by learning a state-action mapping from the expert samples through standard supervised learning. However, supervised learning is meant to handle independent samples: in RL, quite the contrary any mistake done by the agent will make it explore a sample distribution which might deviate from the training one (the distribution drift); eventually, the discrepancy between the expert and the agent distributions quadratically increases with the length of the agent trajectory.

Another option is to exploit the demonstrations to uncover the expert's intent, and infer a reward function compatible with the expert demonstrations. This option offers more guarantees of generalization and robustness, conditionally to the optimality of the demonstrations.

Hybrid LfD approaches have thus been designed, letting the expert interact with the learning agent through additional demonstrations or explicit guidance, thereby mitigating the shortcomings of LfD: the interaction covers for under-optimal

demonstrations and provides complementary guidance in case of distribution drift.

[Cakmak & Thomaz 2012] further studies the interactions between a human expert and a learning agent by taking into account three possible types of interactions. The lessons learned are two-fold. On the one hand, human experts prefer providing binary (yes/no) answers as these are less cognitively demanding. On the other hand, it is suggested that negative answers are useless as the agent can hardly exploit trajectories demonstrating what *not* to do.

Other approaches, focused on exploiting the expert's feedback are presented and discussed in chapter 4.

Chapter 4

Interactive Learning for Optimization

Foreword

Experts shape machine learning problems in different ways. In unsupervised learning, they mostly describe and gather the examples. In supervised learning, they additionally attach labels to each example. In reinforcement learning, they attach rewards to state-action pairs in standard RL (chapter 2) or they demonstrate the desired behavior in Learning from Demonstrations (chapter 3).

In some cases however, experts do not provide any objective guidance, for the concepts to be learned can hardly be defined in a general and unambiguous way. This might be the case when the concepts to be learned are subjective, such as "a pleasant music" in computational art, or "an interesting behavior" in robotics. In such cases, an alternative is to have the expert in the loop, guiding the algorithm *in lieu* of a computable objective.

Early approaches involving the expert in the loop were pioneered in Interactive Evolutionary Computation (IEC) [Herdy 1996, Takagi 2001]: in comparison-based optimizers, the lack of objective function is palliated by asking the user to manually express preferences or rate the candidate solutions. Application domains include e.g., coffee recipe optimization, image synthesis, image rendering, computational music and autonomous robotics [Herdy 1996, Secretan *et al.* 2011, Lund *et al.* 1998, Suga *et al.* 2005].

In supervised learning, the expert in the loop intervenes in the so-called active learning setting: the cost of labelling the whole training set can be reduced by letting the learning system iteratively determine the most informative samples and ask their labels to the oracle (expert) [Freund *et al.* 1997, Dasgupta 2005],

thereby reducing the number of samples required to ensure a given model accuracy a.k.a. the sample complexity. Likewise, in active ranking, the learning agent can determine the most informative preference queries and ask the expert's feedback [Chu & Ghahramani 2005a].

This chapter is interested in Interactive Learning for Optimization (ILO). In ILO, the expert in the loop answers the agent preference queries in order to find (quasi) optimal solutions to an optimization problem. In short, ILO sidesteps the definition of the target objective, thanks to the user in the loop.
The essential difference between ILO and active learning or active ranking – although these can be handled as optimization problems as well – is the following. In active learning or ranking, the primary goal is to find an accurate classification or ranking model; the criterion to be optimized is the accuracy of the model. Quite the contrary, the primary goal in ILO is to find *the optima* of the learned model, also referred to as *surrogate objective*: the learned model is but a means toward the ILO goal, whereas it is the end for active learning and ranking approaches. In brief, in ILO the ranking errors made in the unpromising regions are harmless, whereas all ranking errors matter in active ranking.

This chapter first reviews some related work pioneering the ILO scheme for autonomous control or optimization, before discussing in more depth its application to RL. We distinguish whether the expert's feedback is related to a single state-action pair [Thomaz & Breazeal 2006, Knox & Stone 2009, Knox & Stone 2012] (section 4.2), to a sub-behavior [Wilson *et al.* 2012] (section 4.4), or a whole trajectory [Cheng *et al.* 2011, Akrour *et al.* 2011] (section 4.3). Hybrid approaches, combining preferences over full trajectories and manual revision of a trajectory, are finally presented [Jain *et al.* 2013].

4.1 Related ILO work

Let us first briefly mention some work based on interactive optimization, though mildly related to reinforcement learning.

In a pioneering work, [Lund *et al.* 1998] showed how IEC could be used to fast prototyping of interesting robotic behaviors by children. A limited search space was defined and, at each generation, the children iteratively selected their preferred

behaviors from a pool of candidates. The main limitation of IEC is the number of user's interventions (akin sample complexity) needed to reach a satisfactory result; the user's cognitive burden must clearly remain tolerable. In [Lund *et al.* 1998], the limitations on the search space were instrumental in enforcing a limited sample complexity. The user has access to a "bad" button, resulting in random modifications of the connections of the sensors which were active whenever the button was pressed.

[Suga *et al.* 2005] are interested in the visual exploration of the policy space; they use the expert's feedback about the policy diversity to build a clustering on the parametric policy space, along a self-organized map approach, thus enforcing the diversity in the displayed behaviors, mitigating the user's boredom and increasing the quality of the feedback.

Let us also present two ILO approaches [Chu & Ghahramani 2005b, Viappiani & Boutilier 2010]. Although they do not fall in the scope of control and robotics, they were influential in the design of our own approaches, presented in chapters 6 and 7. Specifically, they inspired the handling of the exploitation/exploration dilemma that any ILO algorithm is faced with (while extensive adaptations were required due to the high dimensionality of the control problems search spaces).

4.1.1 Preference Learning with Gaussian Processes

[Chu & Ghahramani 2005b] present a probabilistic preference learning approach based on Gaussian Processes (GPs). Letting prior $\mathcal{P}(f)$ be a Gaussian Process, and given a dataset $\mathcal{D} = \{v_k \succ u_k : k = 1, ..., m\}$ of pairwise preferences over the space $\mathcal{X}$, the posterior distribution $P(f|\mathcal{D})$ over the function space of functions $f : \mathcal{X} \mapsto \mathbb{R}$ is learned, where the likelihood $\mathcal{P}(\mathcal{D}|f)$ follows the Thurstone-Mosteller paired comparison model.

The Thurstone-Mosteller model returns the random variable "preferring v_k over u_k knowing $f(v_k)$ and $f(u_k)$" as: $f(v_k) + \varepsilon_v$ is greater than $f(u_k) + \varepsilon_u$ (where ε_v and ε_u are two i.i.d. random variables following $\mathcal{N}(0, \sigma^2)$, a zero mean normal distribution of variance σ^2).

More formally, if the noiseless feedback function $\mathcal{P}_{\text{ideal}}$ is given by:

$$\mathcal{P}_{\text{ideal}}(v_k \succ u_k | f(v_k), f(u_k)) = \begin{cases} 1 & \text{if } f(v_k) \geq f(u_k) \\ 0 & \text{otherwise,} \end{cases}$$

then the probability of $v_k \succ u_k$ given f according to the Thurstone-Mosteller model is equal to:

$$\begin{aligned} \mathcal{P}(v_k \succ u_k | f) &= \mathcal{P}_{\text{ideal}}(v_k \succ u_k | f(v_k) + \varepsilon_v, f(u_k) + \varepsilon_u) \\ &= \int_{-\infty}^{z_k} \mathcal{N}(t)\, \mathrm{d}t = \Phi(z_k), \text{with } z_k = \frac{f(v_k) - f(u_k)}{\sqrt{2}\sigma} \end{aligned}$$

where $\mathcal{N}(.)$ denotes the p.d.f. of the standard normal distribution.

The prior $\mathcal{P}(f)$ is such that for any set of points $\{x_1 \dots x_n\} \in \mathcal{X}^n$ their associated values $\mathbf{f} = [f(x_1) \dots f(x_n)]^T$ are the realization of a zero mean Gaussian Process. This process is fully defined by its second order statistics which itself can be specified by any Mercer kernel function. A common choice of kernel is the Gaussian one $\mathcal{K}(x_i, x_j) = \exp(-\frac{\kappa}{2}||x_i; -x_j||^2)$. Denoting Σ the covariance matrix s.t. $\Sigma_{i,j} = \mathcal{K}(x_i, x_j)$, it follows for the process property that $\mathbf{f}$ is distributed according to a multivariate Gaussian distribution:

$$\mathcal{P}(\mathbf{f}) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} \exp(-\frac{1}{2}\mathbf{f}^T \Sigma^{-1} \mathbf{f})$$

Imposing a Gaussian Process prior, and a Thurstone-Mosteller pairwise preference model for the likelihood function, completes the definition of the posterior:

$$\mathcal{P}(\mathbf{f}|\mathcal{D}) \propto \mathcal{P}(\mathbf{f}) \prod_{k=1}^{m} \mathcal{P}(v_k \succ u_k | \mathbf{f}) \tag{4.1}$$

Finding the *maximum a posteriori* estimate $\mathbf{f}_{\text{MAP}} = \arg\max_{\mathbf{f}} \mathcal{P}(\mathbf{f}|\mathcal{D})$, is equivalent to minimizing $\mathcal{S}(\mathbf{f}) = -\sum_{k=1}^{m} \ln \Phi(z_k) + \frac{1}{2}\mathbf{f}^T \Sigma^{-1} \mathbf{f}$, which is a positive constant shy of minus logarithm of expression (4.1). To find the minimum of $\mathcal{S}(.)$, the authors first show that the Hessian matrix of $\mathcal{S}$ is positive semidefinite and hence the function is convex. It follows that the global minimum $\mathbf{f}_{\text{MAP}}$ of $\mathcal{S}(.)$ verifies $\frac{\partial \mathcal{S}(\mathbf{f})}{\partial \mathbf{f}}|_{\mathbf{f}_{\text{MAP}}} = 0$ yielding $\mathbf{f}_{\text{MAP}} = \Sigma \frac{\partial \sum_{k=1} m ln \Phi(z_k)}{\partial \mathbf{f}}|_{\mathbf{f}_{\text{MAP}}}$.

In order to predict $\mathbf{f}_t = [f(x)\ f(y)]^T$, the random vector specifying the latent values $f(x)$ and $f(y)$ of an unseen pair $(x, y) \in \mathcal{X} \times \mathcal{X}$, the posterior distribution $\mathcal{P}(\mathbf{f}_t|\mathcal{D})$ needs to be computed. From the Gaussian Process hypothesis, it

follows that $[\mathbf{f}\ \mathbf{f}_t] \sim \mathcal{N}\left(0, \begin{pmatrix} \Sigma & k_t \\ k_t^T & \Sigma_t \end{pmatrix}\right)$, where $k_t = \begin{bmatrix} \mathcal{K}(x,x1) & \dots & \mathcal{K}(x,x_n) \\ \mathcal{K}(y,x1) & \dots & \mathcal{K}(y,x_n) \end{bmatrix}^T$ and $\Sigma_t = \begin{pmatrix} \mathcal{K}(x,x) & \mathcal{K}(x,y) \\ \mathcal{K}(y,x) & \mathcal{K}(y,y) \end{pmatrix}$. For any zero mean joint multivariate normal distribution $\mathcal{P}(\mathbf{f}, \mathbf{f}_t)$, the conditional $\mathcal{P}(\mathbf{f}_t|\mathbf{f})$ is also normally distributed, with law $\mathcal{N}(k_t^T\Sigma^{-1}\mathbf{f}, \Sigma_t - k_t^T\Sigma^{-1}k_t)$. Knowing the conditional distribution, integration over the function space yields $\mathcal{P}(\mathbf{f}_t|\mathcal{D}) = \int \mathcal{P}(\mathbf{f}_t|\mathbf{f})\mathcal{P}(\mathbf{f}|\mathcal{D})\ \mathrm{d}\mathbf{f}$. The authors then use a Laplace approximation [MacKay 1994] to reduce $\mathcal{P}(\mathbf{f}|\mathcal{D})$ to a Gaussian distribution centered at the previously computed *maximum a posteriori* function $\mathbf{f}_{\text{MAP}}$ with covariance matrix $(\Sigma^{-1} + \Delta_{\text{MAP}})^{-1}$; as a result $\mathcal{P}(\mathbf{f}_t|\mathcal{D})$ is also normally distributed with mean $\mu^* = k_t^T\Sigma^{-1}\mathbf{f}_{\text{MAP}}$ and covariance matrix $\Sigma^* = \Sigma_t - k_t^T(\Sigma + \Delta_{\text{MAP}}^{-1})^{-1}k_t$ which fully defines the distribution of the latent values of any test pair (x, y).

4.1.2 Active Preference Learning with Discrete Choice Data

[Brochu *et al.* 2007] advocate the use of preference learning in an interactive optimization framework, to solve problems where the objective function is not analytically known. The authors take as an example psycho-perceptual models where artists have to manually tune continuous parameters until a computer generated image "looks right". On the one hand, if presented with an image and asked to rate it, the person is likely to exhibit a *drift effect*, where the scale will vary over time [Siegel & Castellan 1988]. However, human beings are known to be good at comparing stimuli or items (see chapter 1 of [Brown & Peterson 2009]). Accordingly, the algorithm iteratively presents a set of alternatives and lets the user pick the best one; preference learning is thus iteratively used to estimate a valuation function on the parameter space. The valuation function acts as a surrogate model, driving the exploration to find the optimum. Note that the goal is here to find the optimum of the valuation function (as opposed to, accurately model the valuation function over the whole parameter space).

The main challenge is the trade-off between exploring new regions of the parameter space and exploiting the regions with known high evaluation. The authors chose a probabilistic modeling of the parameter landscape using Gaussian Processes as in [Chu & Ghahramani 2005b], giving necessary tools to handle the aforementioned

trade-off.

The mathematical details strictly follow those of [Chu & Ghahramani 2005b]. Given a dataset of preferences $\mathcal{D} = \{v_k \succ u_k : k = 1, ..., m\}$, and by imposing the hypothesis of a Gaussian Process and a Thurstone-Mosteller preference model[1], the posterior evaluation $f(x)$ of a candidate set of parameters x is approximated by a normal distribution of mean $\mu(x)$ and variance $\sigma^2(x)$. On the basis of the posterior distribution, the Expected Improvement [Jones *et al.* 1998] is used, defining the probability of improvement over a value $\mu_{\max}$ as

$$P(f(x^*) \leq \mu_{\max}) = \Phi\left(\frac{\mu_{\max} - \mu(x^*)}{\sigma(x^*)}\right)$$

Defining an improvement over the current best $\mu_{\max}$ as $I(x^*) = \max\{0, f(x^*) - \mu_{\max}\}$ [Jones *et al.* 1998], the Expected Improvement $EI(x^*)$ is then the expectation of I under the normal distribution hypothesis, yielding upon integration :

$$EI(x^*) = \begin{cases} (\mu_{\max} - \mu(x^*))\Phi(d) + \sigma(x^*)\mathcal{N}(d) & \text{if } \sigma(x^*) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

With $d = \frac{\mu_{\max} - \mu(x^*)}{\sigma(x^*)}$. Φ and $\mathcal{N}$ are respectively the cumulative and density function of the standard normal distribution. Taking the arg max of $EI(.)$ can be used to select the point with the highest expectation of improvement over the current best valuation. It is not clear however from the paper how this criterion is used to build a diverse query with more than one parameter set to be shown to the user.

The experimental validation shows that, on multi-modal functions of dimension 2 to 6, the approach improves on selecting the points with maximum variance, which is a common practice in active learning for regression [Seo *et al.* 2000, Chu & Ghahramani 2005a].

The authors note that for dimensions higher than 6, the benefit of following EI instead of choosing the query points randomly tend to be negligible. However, uniformly sampling the space of interest may be a non-trivial task[2]; criteria such as

[1] [Stern 1992] show that a family of Gamma distributions with a shape parameter r can encompass both the Thurstone-Mosteller model when $r \to \infty$, and the Bradley-Terry model (also known as the Luce model) for $r = 1$. Moreover, all models in this family seem to perform equivalently in practice (in terms of goodness of fit), suggesting that the Thurstone-Mosteller model is not a key ingredient of the approach, and that other preference models might be used as well.

[2] For instance, uniformly sampling the policy parameter space does not translate into a uniform sampling of the trajectory space.

EI are useful guides to achieve such sampling.

4.1.3 Preference elicitation for recommendation sets

[Viappiani & Boutilier 2010] tackles the interaction between a user and a recommendation system of options $x \in \mathcal{X}$. The user has an unknown utility function $\mathbf{w} \in W$ and the system current uncertainty over the true utility is captured by a belief θ defining a distribution $P(\mathbf{w};\theta)$ over W. The system interacts with the user by proposing a set of k alternative options $S = \{x_1, \ldots, x_k\}$ that could either be:

i) a *recommendation set* maximizing the user *expected utility of selection* (EUS) under the current belief. For $k = 1$ the best proposal is the option maximizing the *expected utility*

$$x^*(\theta) = \arg\max_{x \in \mathcal{X}} EU(x;\theta)$$

with

$$EU(x;\theta) = \mathbb{E}_{w \sim P(.;\theta)}\left[\mathbf{w}(x)\right]$$

For $k \geq 2$, the EUS criterion captures the intuitive need for selecting diverse options reflecting the uncertainty of the user's utility. Letting $P_R(S \rightsquigarrow x;\mathbf{w})$ be the user response model (giving the probability the user selects alternative x from S had the true utility $\mathbf{w}$ been known) and $P_R(S \rightsquigarrow x;\theta) = \mathbb{E}_{w \sim P(.;\theta)}\left[P_R(S \rightsquigarrow x;\mathbf{w})\right]$ its expectation under the current belief, EUS is defined by:

$$EUS_R(S;\theta) = \sum_{x \in S} P_R(S \rightsquigarrow x;\theta) EU(x;\theta \wedge S \rightsquigarrow x)$$

Where the belief $\theta \wedge S \rightsquigarrow x$ is an update of θ that takes into account the information that the user prefers the option x in S.

ii) a *query set* aimed at refining the system's understanding of the user preferences (often called preference elicitation). From this set the user picks its preferred option x. Upon receiving the user's feedback, the system updates its belief to $\theta \wedge S \rightsquigarrow x$. In the literature, a natural optimization criterion to select the most adapted query set is *expected value of information* (EVOI) [Chajewska *et al.* 2000, Boutilier 2002] that represents the improvement in the expected utility. Letting $EU^*(\theta) = EU\left(x^*(\theta);\theta\right)$ and the *expected posterior utility* (EPU) be:

$$EPU_R(S;\theta) = \sum_{x \in S} P_R(S \rightsquigarrow x;\theta) EU^*(\theta \wedge S \rightsquigarrow x)$$

EVOI is simply the difference $EPU_R(S;\theta) - EU^*(\theta)$ between the posterior and the current best utility. Since the $EU^*(\theta)$ is fixed for any set S, optimizing EVOI amounts to optimizing EPU.

However, [Viappiani & Boutilier 2010] claim that at least in the noiseless case[3] the optimal query set coincides with the optimal recommendation set. In other words, the best query to make in order to improve the recommendation system is to ask the user his preferred option in a diverse portfolio of his most preferred items. The immediate benefit of this result is to accelerate the research of the optimal query set since the computation of the EUS of a set does not require an additional optimization step as with EPU that looks one step ahead in the belief.

In the noisy case, the Luce-Sheppard model $P_R(S \rightsquigarrow x; w) = \frac{exp(\gamma w(x))}{\sum_{x' \in S} exp(\gamma w(x))}$ is studied and even if $EUS^*(\theta)$ and $EPU^*(\theta)$ do not coincide any more, they are not further apart from each other than a small constant that only depends on the temperature γ and the size of the set k.

Finally, a practical optimization scheme is proposed for the optimization of the noiseless EUS, noting that it is a sub-modular function on the set of alternatives. Hence its greedy optimization (i.e. iteratively adding the x that increases the most the EUS) will lead to a set S_g that is lower bounded by $\eta EUS^*(\theta)$ with $\eta = 1 - \left(\frac{k-1}{k}\right)^k$. For the pairwise preference case, $\eta = .75$ and goes to .63 as $k \to \infty$.

The rest of the chapter now focuses on ILO for control, distinguishing the cases where the user's feedback is related to single state-action pairs (section 4.2), on short behaviors (section 4.4) or on whole trajectories (section 4.3).

4.2 User's feedback on state-action pairs

The TAMER (Training an Agent Manually via Evaluative Reinforcement) framework exploits the expert in the loop of an RL algorithm [Knox & Stone 2009], with two specificities. Firstly, the inherent delay in human reactions implies that the feedback will arrive after the intended state-action pair. Hence a probabilistic credit assignment mechanism is designed, following a Gamma distribution of the reward on the recent state-action pairs. This distribution is parameterized after experi-

[3] In this case $P_R(S \rightsquigarrow x; \mathbf{w}) = 1$ if $x = \arg\max_{x' \in S} \mathbf{w}(x')$ and 0 otherwise.

mental studies on human response time in cognitive tasks [Hockley 1984]: after an initial interval of negligible mass, the function quickly peaks before decreasing exponentially. Secondly, as demonstrated by [Thomaz & Breazeal 2006], the positive human feedback often signals that the current states offer promising opportunities, more than the fact that the recent sub-behavior was relevant. The feedback should therefore be interpreted more in terms of a value function or a policy advice, than as an instant reward.

Experimental evidence confirming this remark is proposed in [Knox & Stone 2012]: when comparing different ways of combining the feedback with the RL reward, the best results are obtained when the feedback directly biases the action selection mechanism. Letting $H : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ be the function associating to each state-action pair a value reflecting the feedback of the user after the credit assignation is done, the best ways of using H in practice are:

- **Action biasing**: $\pi(s) = \arg\max_a Q(s,a) + \beta H(s,a)$ only during action selection.
- **Control sharing**: $\pi(s) = \arg\max_a H(s,a)$ with probability $min(\beta, 1)$, and $\arg\max_a Q(s,a)$ otherwise.

with β a temperature-like hyper-parameter, decreasing to 0 and enforcing the eventual use of the RL policy. In practice, on the mountain-car and the Tetris RL benchmarks, action biasing and control sharing outperform the use of H as an instant reward (e.g. replacing the instant reward $\mathcal{R}(s,a)$ with a weighted sum of $\mathcal{R}(s,a)$ and $H(s,a)$).

In the Advise algorithm [Griffith *et al.* 2013], the binary user's feedback attached to the current state action pair is interpreted as a policy advice: it is right (resp. wrong) to make this action in this state. All presented experiments are in simulation, where the feedback is assumed to depend on the optimal policy (with no delay), with a single optimal action in each state, according to hyper-parameters $\mathcal{C}$ (probability for the feedback to reflect the optimal policy) and $\mathcal{L}$ (probability to get a feedback). For every state-action pair, the difference $\Delta_{s,a}$ between the number of yes and no feedback related to this pair is stored, and the action selection is modified as:

$$\pi_F(a|s) \propto \mathcal{C}^{\Delta_{s,a}}(1-\mathcal{C})^{-\Delta_{s,a}}$$

yielding a proof of optimality under the assumption that there is a single optimal action in each state.

In addition to π_F the agent learns a second policy π_R from the reward signals using Bayesian Q-Learning. BQL maintains the parameters of a normal distribution representing the uncertainty on the Q-Value of each state-action pair. $\pi_R(a|s)$, the probability that a is optimal in s, is estimated by sampling the Q-Value at state s and counting the number of times it is maximized by the action a. The agent ultimately follows a policy π combining π_F and π_R by multiplying them: $\pi \propto \pi_R \times \pi_F$.

An empirical validation of Advise, comparatively to Action Biasing, Control Sharing and Reward Shaping [Knox & Stone 2010, Knox & Stone 2012] is conducted on a game problem with discrete 34,000 states and 5 actions, and a simulated expert. Experimentally, when both the consistency and the frequency of the expert feedback are high, Advise shows a faster convergence, suggesting that it uses the expert's feedback to reduce the exploration. Advise shows a comparatively better robustness, even for a feedback consistency up to 50%; this better robustness might however be explained by the best adjustment of hyper-parameter $\mathcal{C}$. A sensitivity analysis w.r.t. $\mathcal{C}$ suggests that i) underestimating $\mathcal{C}$ reduces the feedback benefits; ii) overestimating $\mathcal{C}$ speeds up the learning but might lead to local optima.

In summary, Advise offers a principled way to integrate the user's feedback as direct policy suggestions. Further perspectives might be to tackle the automatic estimation of the consistency $\hat{\mathcal{C}}$ parameter together with the credit assignment mechanism.

4.3 User's feedback on full trajectories

Intuitively, it requires less expertise to provide a feedback on a whole trajectory than on a single state-action pair. In the former case, the user's feedback is interpreted as whether the robot behavior is as expected; it judges a result. In the latter case, the user's feedback is interpreted as to whether the current state offers promising opportunities; it judges potentialities.

A main approach exploiting the user's feedback on full trajectories is the Preference-Based Policy Iteration (PBPI) algorithm [Cheng *et al.* 2011] (another one, our PPL algorithm, will be presented in chapter 5). PBPI builds upon the Roll-

out Classification Policy Iteration (RCPI) algorithm [Lagoudakis & Parr 2003b] (section 2.2.1). Letting π denote the current policy, in a state s, two actions a and a' are compared by considering the trajectories starting by selecting a (respectively a') in state s, and thereafter following π. Whereas RCPI compares the cumulative reward of these trajectories, PBPI asks the user's feedback[4]. The user's feedback is exploited using a label ranker [Hüllermeier *et al.* 2008]. Thereby, conditionally to s and π, all actions can be ordered (whereas only the optimal action is considered in RCPI); PBPI can thus exploit any significant preference of action a over action a' ($a \succ_{\pi,s} a'$), facilitating the further exploration of the search space.

Experimentally, the exploitation of the available pairwise preferences (including the comparison of suboptimal actions) yields better policies. The advantage of qualitative preferences is demonstrated on the cancer treatment problem expending the realm of reachable solutions outside of the convex hull defined from the two objectives, the tumor size and the toxicity (more in section 6.4.2.1). The main limitation of PBPI regards its scalability, since one user's feedback is required for each pair of actions conditionally to a state and a current policy.

This scalability limitation will be addressed by the Preference-based Policy Learning (PPL) (chapter 5).

4.4 User's feedback on short trajectories

[Wilson *et al.* 2012] ask the user's preference related to pairs of trajectories of short fixed length with same initial state, referred to as *trajectory preference queries* (TPQs); their motivation is to reduce the user's fatigue. The formal setting is that of MDP, deprived from the reward function $\mathcal{R}$; the TPQs are used to build a posterior distribution over the policy parameters space.

The user is assumed to have in mind a target policy π_{θ^*} and favor trajectories matching π_{θ^*} behavior. It is further assumed that i) the transition model of the MDP is known; ii) a distribution P_0 used to sample initial states for the short trajectories is provided; iii) the agent can select and replay verbatim the trajectories.

[4] The motivating application is the cancer treatment in clinical trials [Zhao *et al.* 2009]; instead of associating a numerical penalty to the death issue, one might safely assume that a treatment resulting in the patient death is always worse than one where the patient is healed.

Note that distribution P_0 conveys much more information than the standard initial distribution used in MDP: since short trajectories are demonstrated, their initial state must be sufficiently informative (close to the goal states) for the user to emit relevant feedback.

Let $\mu = \{s_1, a_1, \ldots, a_{K-1}, s_K\}$ be a trajectory of length K. The user's preference among trajectories μ and μ' compares their distance to the target policy π_{θ^*}:

$$f(\mu, \mu', \theta^*) = d(\mu', \theta^*) - d(\mu, \theta^*)$$

with

$$d(\mu, \theta^*) = \mathbb{E}_{\mu^* \sim \pi_{\theta^*}} \left[\sum_{t=0}^{K} ||s_t - s_t^*|| + ||a_t - a_t^*|| \right]$$

The dissimilarity measure $d(\mu, \theta^*)$ encodes how much the trajectory μ disagrees in expectation with trajectories sampled from π_{θ^*}, summing the e.g. Euclidean distance among the states and actions. Given the (real) value $f(\mu, \mu', \theta^*)$, the user's binary feedback is set to the sign thereof up to a Gaussian noise $\varepsilon \sim \mathcal{N}(0, \sigma)$ (the Thurstone-Mosteller preference model, section 4.1.1). Finally:

$$P\left(y = 1 \middle| \mu, \mu', \theta^*\right) = \Phi\left(\frac{f(\mu, \mu', \theta^*)}{\sigma}\right) = \Phi(z)$$

The posterior distribution is defined from a learning set of TPQs $D = \{(\mu, \mu', y)_i\}$ as:

$$P(\theta|D) \propto P(\theta) \prod_i \Phi\left(z_i\right)^{y_i} \left(1 - \Phi(z_i)\right)^{1-y_i}$$

with prior $P(\theta)$ a zero mean Gaussian, favoring policy parameters with small values.

Two criteria are considered to select the next TPQ to be demonstrated, the query by disagreement and the expected belief change. The former criterion proceeds by sampling an initial state from P_0 and two policy parameters θ and θ' from the posterior distribution $P(.|D)$; their disagreement $\mathbb{E}_{\mu \sim \theta}\left[d(\mu, \theta')\right]$ is computed and the TPQ is retained if the disagreement exceeds a predefined threshold. The latter criterion builds a pool of TPQ (by sampling states from P_0 and policy parameters from the posterior) and selects the one which maximizes the change in the posterior after seeing the additional TPQ.

Experimental validation on four standard RL benchmarks with continuous states and discrete actions show the merits of the approach, requiring 25 to 50 TPQs to build optimal policies and showing that the query by disagreement criterion is more effective when P_0 does not incorporate enough domain knowledge.

4.5 Hybrid approaches

A hybrid approach, the Trajectory Preference Perceptron (TPP) is presented by [Jain *et al.* 2013] within the online co-active learning framework [Shivaswamy & Joachims 2012]. In TPP the user can rank the demonstrated trajectories (the ranks are interpreted as pairwise preferences) or modify the top-rank demonstration. The latter option offers a way to escape from local optima (when trajectories are correctly ranked although none of them is satisfactory).

The motivating applications deal with a robotic arm with a high number of degrees of freedom (DoF). An informative set of features is assumed to be available, describing both the trajectory and the object-to-object and object-to-environment relations along the trajectories. Expliciting the background knowledge (e.g. heavy objects should not be moved on the top of fragile objects; sharp objects should be kept at a distance from humans) is not easy. Quite the contrary, refining a trajectory which violates such above constraints is straightforward.

The user's feedback can be one of the following: i) after viewing an ordered sequence of (simulated) trajectories, the user can swap a trajectory with the top one; ii) the user can physically modify one of the way-points[5] of the top trajectory by direct manipulation of the robot arm. In both cases, the user's feedback is formalized as $\bar{y}_t \succ y_t$, where trajectory y_t is the optimal trajectory after the user preference model at time t and trajectory $\bar{y}_t$ is either the trajectory preferred to y_t (first type of feedback) or the result of direct modifications of y_t (second type of feedback).

Let ϕ be a feature function[6] mapping both a trajectory y and its context on $\mathbb{R}^D$. For simplicity, the preference model is assumed to be linear, parameterized as $\mathbf{w}_t$ in the feature space. Upon receiving feedback $\bar{y}_t \succ y_t$, the model is updated using a perceptron like formula:

$$w_{t+1} = w_t + \phi(x_t, \bar{y}_t) - \phi(x_t, y_t)$$

TPP first samples a set of trajectories $\mathcal{S}_t$ using rapidly-exploring random tree

[5] A trajectory y is discretized as a sequence of way-points $y^1 \dots y^N$, seen as key moments of the trajectory.

[6] The features are further split into ϕ_O, capturing the interactions between the objects along the trajectory way-points, and ϕ_E which captures the way the object is moved.

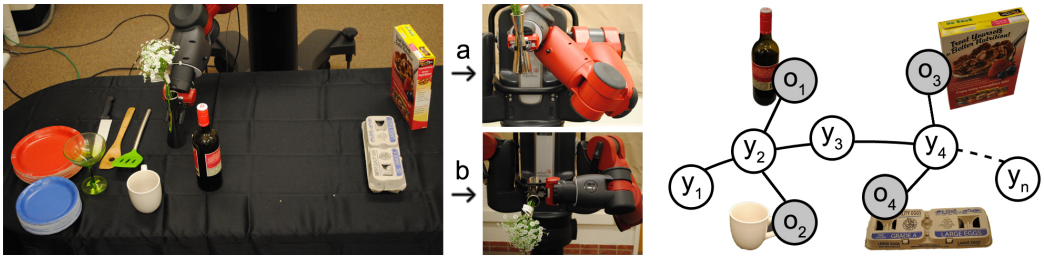


Figure 4.1: [Jain *et al.* 2013]. **Left**: General grocery checkout scene consisting of a table and objects with different properties such as weight, fragility and sharpness. The robot task is to move the flower vase from right to left. **Center**: Two ways of moving the vase are shown, both are suboptimal. In a) the arm is contorted, which is visible from the ϕ_E feature description of the trajectory (robot arm configuration). In b) the vase, which may contain water, is tilted. Keeping liquid containers straight can be learned using ϕ_O. **Right**: A graph representation of the trajectory encodes all the object-object interactions. $\{y_1 \dots y_n\}$ are way-points; and the edges to the shaded nodes indicate that the object is in the vicinity of the way-point.

(RRT) [LaValle & Jr. 2001] (algorithm 3), selects the best trajectory based on the current preference model $\mathbf{w}_t$. The user is asked for a feedback and returns an improved trajectory $\bar{y}_t$. Finally, the parameters are updated according to $\bar{y}_t \succ y_t$ and the process is iterated.

Algorithm 3 Trajectory Preference Perceptron. (TPP)

1: Initialize $\mathbf{w}_1 \leftarrow 0$
2: **for** $t = 1$ to T **do**
3: Sample trajectories $\mathcal{S}_t = \{y_t^{(1)}, ..., y_t^{(n)}\}$
4: $y_t = \arg\max_{y \in \mathcal{S}_t} \; s(x_t, y; \mathbf{w}_t)$
5: Obtain user feedback $\bar{y}_t$
6: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \phi(x_t, \bar{y}_t) - \phi(x_t, y_t)$
7: **end for**

The user's feedback $\bar{y}_t$ is said to be α-informative w.r.t. y_t^* in expectation if it satisfies:

$$\mathbb{E}\left[s^*(x_t, \bar{y}_t)\right] - s^*(x_t, y_t) \geq \alpha(s^*(x_t, y_t^*) - s^*(x_t, y_t)) - \xi_t, \;\; \alpha \in (0, 1]$$

where s^* is the true (unknown) preference function and ξ_t a slack variable accounting for the user's inconsistency (α-informativeness is strict if $\xi_t = 0$). Let us further define the regret after T steps as:

$$\mathrm{REG}_T = \frac{1}{T}\sum_{t=1}^{T} s^*(x_t, y_t^*) - s^*(x_t, y_t)$$

The authors show that if the user's feedback is strictly α-informative, then the regret is in $\mathcal{O}(\frac{1}{\sqrt{T}})$, sharing the same asymptotic behavior when the user knows the optimal trajectory for each context.

The validation of TPP considers a grocery checkout; the baseline involves 1300 manually labelled trajectories. The experimentations involve 5 humans and the Baxter robot (high DoF robot arm, see figure 4.1-Left). Overall, TPP outperforms SVM-rank trained on the labelled trajectories after circa 5 feedbacks. Swapping trajectories is the most used feedback option for simple tasks. Direct manipulation of the arm configuration is increasingly used as the task difficulty increases. Overall it takes 5.5 minutes on average until the user is satisfied with the top-ranked trajectory.

4.6 Summary and Discussion

The expert in the loop is not required to be as knowledgeable as the expert in the Learning from Demonstration framework, presented in the previous chapter: the only assumption done is that i) the current solution(s) can be displayed to the user; ii) she can provide a feedback.

This feedback can be thought of in terms of guidance in an optimization landscape. The most informative feedback (when the expert is allowed to revise a solution [Jain *et al.* 2013]) corresponds to the case where the current solution is moved in the search space, akin a steepest ascent gradient procedure. Unsurprisingly, such abilities on the behalf of the expert yield some guarantees on the overall convergence of the ILO process.

Several trade-offs must be achieved to tackle ILO, at the crossroad of algorithmic and cognitive concerns.

On the algorithmic side, we first face the exploration vs exploitation dilemma. On the one hand, the agent should visit the most promising regions of the search space, according to the current model of the expert's preferences; on the other hand,

a sufficient exploration of the search space is needed to support the consistency of the algorithm. The exploration should however comply with the limited availability and attention of the expert: a few dozens, perhaps hundreds of preference queries can be asked; not thousands. Active learning/ranking heuristics are thus needed to keep the number of feedback queries within reasonable limits.

On the cognitive side, a tradeoff between the precision and the reliability of the expert's feedback is observed: the more precise the expert's feedback, the more useful it is in an algorithmic perspective, *a priori*. But the value of precise feedback might be subject to noise and drift, according to the so-called rating drift phenomenon [Siegel & Castellan 1988]. Furthermore, the impact of feedback noise is bound to be more severe if the number of feedback queries is restricted.

Again on the cognitive side, a trade-off between focus and expertise is expected: providing a feedback on a whole behavior likely requires less expertise than providing an instant feedback on a state-action pair, or a feedback on a sub-behavior. In the former case, the expert appreciates whether the robot agent has reached the goal(s) with full information. In the latter cases, the expert appreciates whether the robot agent *can* (on the basis of this sub-behavior or instant behavior) reach some relevant goal in the future: this requires more finesse. In counterpart, observing long behaviors is more demanding (and thus induces more noise) than observing short agent demonstrations.

In the next chapters, three algorithms facing these challenges will be presented:

- Preference-based Policy Learning (PPL, chapter 5) pioneered the use of preference learning in reinforcement learning, exploiting the expert's binary feedback on pairs of whole trajectories to learn a policy return estimate on the trajectory space;
- Active Preference-learning for Reinforcement Learning (APRIL, chapter 6) focuses on reducing the expert's burden and the number of preference queries needed to yield satisfactory results;
- Finally, Programming by Feedback (PF, chapter 7) aims at enforcing a robust agent behavior, resilient w.r.t. expert's mistakes.

Chapter 5

Preference-based Policy Learning

Foreword

This chapter introduces the first contribution of the PhD, the *Preference-based Policy Learning* (PPL) algorithm first presented in [Akrour *et al.* 2011].

The primary motivation for designing PPL was that the use of robotic simulators was not an authorized option in the first years of the FP7 SYMBRION IP that funded this work, for several reasons. First and overall, the SYMBRION roboticist partners thought poorly of robotic controllers developed *in silico*, due to former disappointing experiences. Moreover, at that time, there did not exist any appropriate robotic simulator for the target SYMBRION robotic hardware, the robot unit that was still under design. Finally, simulator-based studies are ill-suited to robotic swarm design, as the computational cost increases quadratically with the size of the swarm and the simulator precision is adversely affected by the unavoidable manufacturing variability of the robotic units.

In the PPL framework, the human in the loop basically replaces the simulation-based assessment and optimization of a robotic controller, according to a four-step process:

i) the robot demonstrates a candidate policy;
ii) the expert ranks this policy comparatively to previously seen ones according to her preferences;
iii) these preferences are used to learn a policy return estimate;
iv) the robot uses the policy return estimate to build new candidate policies.

This process is iterated until the desired behavior is obtained.

The critical issue is the representation of the policy search space, meant to enable learning accurate policy return estimates and limiting the human ranking effort needed to yield a good policy. The difficulty is that this representation cannot use informed features (e.g., how far the robot is from any target position) due to the simulator-free, ground truth-less setting (as opposed to e.g. [Abbeel & Ng 2004]). The representation that is proposed in this chapter is based on the agnostic exploitation of the robotic log, using unsupervised learning to incrementally build sensori-motor states.

This chapter also includes an analytical study of PPL on the artificial RiverSwim problem [Strehl *et al.* 2006], with a convergence proof in that context. Finally, PPL is experimentally validated, on a 2D version of the RiverSwim problem, as well as on two problems relevant to swarm robotics (in simulation). The first one is concerned with reaching of a target location in a maze. The difficulty is due to perceptual aliasing (non-Markovian environment): different locations are perceived to be similar due to the limited infra-red sensors of the robot. The second problem involves two interacting robots. The task is to yield a synchronized behavior of the two robots, controlled with the same policy. The difficulty is again due to the fact that sensors hardly enable a robot to discriminate between its relative and an obstacle. Interestingly, it is shown that the PPL framework enables the robot to achieve the synchronized behavior *without directly acquiring the perceptual primitives* discriminating the other robot from the wall.

5.1 Motivations

As mentioned in chapters 2 and 3, many machine learning approaches in robotics, ranging from direct policy learning [Bain & Sammut 1995] and learning by imitation [Calinon *et al.* 2007] to reinforcement learning [Peters & Schaal 2008] and inverse optimal control [Abbeel & Ng 2004, Kolter *et al.* 2007], rely on simulators, considered as complete motion and world model of the robot. Obviously, the actual performance of the learned policies depends on the accuracy and calibration of the simulator, a phenomenon referred to as *Reality Gap* [Lipson *et al.* 2006] (see also [Saxena *et al.* 2008]). As shown in [Lipson *et al.* 2006], one can alleviate the shortcomings of the simulator using an active learning approach, gradually *closing the*

gap between the simulated and physical worlds; incidentally, the closure of the gap also handles the non-stationarity of the world, e.g. due to the robot sensor fatigue and hazards. The combined use of simulated and physical experiments however is outside of the scope of the presented work, and it defines a perspective for further research.

The first research question investigated in this PhD work is whether robotic policy learning can be achieved in a simulator-free setting, while keeping the human labor and computational costs within reasonable limits. This question is motivated by Machine Learning application to Swarm Robotics [Stirling *et al.* 2010, Trianni *et al.* 2006, Groß *et al.* 2006, Liu & Winfield 2010, O'Dowd *et al.* 2011]. Swarm Robotics aims at designing robust and reliable robotic systems through the physical and virtual interaction of a large number of small-scale, possibly unreliable, robot entities (see chapter 1 Fig. 1.1). Within this framework, the use of simulators suffers from two limitations. Firstly, the simulation cost increases quadratically in the worst case with the size of the swarm (and it does increase more than linearly in practice). Secondly, robot entities might differ among themselves due to manufacturing tolerance, entailing a large variance among the results of physical experiments and severely limiting the accuracy of simulated experiments.

In the Swarm Robotic settings, the Learning from Demonstrations approach (chapter 3) is likewise inappropriate as the expert can hardly impersonate the swarm robot on the one hand, and because the desired behavior is usually unknown on the other hand. More precisely, swarm policy learning defines an inverse problem in a Complex System context: One aims at e.g. designing the ant behavior in such a way that the whole ant colony achieves a given task.

5.2 Overview and analysis

Constrasting with LfD, the proposed approach is based on a light interaction between the policy learning process and the expert, only assuming that the former can *demonstrate* policies and that the latter can emit *preferences*, telling a more appropriate behavior from a lesser one.

Formally, the presented *Preference-based policy learning* (PPL) approach is inspired from both energy-based learning [Ranzato *et al.* 2006] and learning-to-rank

[Joachims 2005, Bakir *et al.* 2006]. PPL proceeds by iterating a 4-step process:

- In the *demonstration* phase, the robot demonstrates a candidate policy.
- In the *teaching* or feedback phase, the expert ranks this policy compared to archived policies, based on her agenda and prior knowledge.
- In the *learning* phase, a policy return estimate (energy criterion) is learned based on the expert ranking, using an embedded learning-to-rank algorithm. A key issue concerns the choice of the policy representation space (see below).
- In the *self-training* phase, new policies are generated, using an adaptive trade-off between the policy return estimate and the policy diversity w.r.t. the archive. A candidate policy is selected to be demonstrated and the process is iterated.

PPL initialization proceeds by demonstrating two policies, which are ordered by the expert.

A main advantage of the PPL approach compared to LfD is that it does not require the expert to know how to solve the task and to demonstrate a perfect behaviour; the expert is only required to know whether some behaviour is more likely to reach the goal than some other one. The demonstrated trajectories are also feasible by construction since they are done by the robot (whereas trajectories demonstrated by the human expert are not necessarily directly feasible whenever the expert and the robot have different degrees of freedom).

Compared to Policy Gradient approaches [Peters & Schaal 2008], the continued interaction with the expert offers some means to detect and avoid the convergence to local optima, through the expert's feedback.

In counterpart, PPL faces one main challenge. The human ranking effort needed to yield a competent policy must be limited; the sample complexity must be of the order of a few dozens to at most a couple hundred. Therefore the policy return estimate must enable fast progress in the policy space, which requires the policy representation space to be carefully designed (a general concern, as noted by [Abbeel & Ng 2004]). On the other hand, the simulator-free setting precludes the use of any informed features such as the robot distance to obstacles or other robots.

In contrast, IRL used to consider fairly informative features, e.g. the driving speed or the bumping into pedestrians [Abbeel & Ng 2004]

5.3 Notations

PPL is concerned with deterministic parametric policies with finite time horizon H, where H is the number of time steps each candidate policy is demonstrated.

Policy π_θ, defined from its parameter $\theta \in \Theta \subseteq \mathbb{R}^D$, stands for a deterministic mapping from the state space $\mathcal{S}$ onto the action space $\mathcal{A}$. To set the ideas, π_θ will most often be implemented as a neural net with fixed topology, where θ stands for the weight vector of this neural net.

A *behavioural representation* is defined from the demonstrations (section 5.4) and used to learn the policy return estimate. PPL proceeds by maintaining a policy and constraint archive, respectively denoted Π_t and $\mathcal{C}_t$, along a four-step process. At step t,

- A new policy π_t is demonstrated by the robot and added to the archive

$$\Pi_t = \Pi_{t-1} \cup \{\pi_t\}$$

- π_t is ranked by the expert w.r.t. the other policies in Π_t, and the set $\mathcal{C}_t$ is accordingly updated from $\mathcal{C}_{t-1}$:

$$\mathcal{C}_{t+1} = \mathcal{C}_t \cup \{\pi_t \prec / \succ \pi_i, i = 1, \ldots, t-1\}$$

- The policy return estimate J_t is built from all constraints in $\mathcal{C}_t$ (section 5.5);
- New policies are generated; candidate policy π_{t+1} is selected using an adaptive trade-off between J_t and an empirical diversity term (section 5.6), and the process is iterated.

PPL is initialized from two policies π_0 and π_1 with specific properties (section 5.7), which are ordered by the expert; the policy and constraint archives are initialized accordingly. After detailing all PPL components, an analytical convergence study of PPL is presented in section 5.8.

5.4 The Behavioural Representation

In many parametric settings, e.g. when policy π_θ is characterized as the weight vector $\theta \in \Theta$ of a neural net with fixed architecture, the Euclidean metric on Θ is poorly informative of the policy behaviour. Typically the upper bound on the difference between the positions reached by two policies π_θ and $\pi_{\theta'}$, starting from the same initial position, after h time steps exponentially increases with h [Fonteneau *et al.* 2010].

As the sought policy return estimate is meant to assess policy behaviour, it thus appears inappropriate to learn the policy estimate J_t as a mapping defined on Θ.

The proposed alternative is inspired from sensori-motor states [O'Regan & Noë 2001]. It takes advantage of the fact that the agent is given for free the datastream made of its sensor and actuator values, generated along its trajectories in the environment (possibly after unsupervised dimensionality reduction) [Lange *et al.* 2012].

A frugal online clustering algorithm approach (e.g., ε-means [Duda & Hart 1973]) is used to define sensori-motor clusters. To each such cluster, referred to as sensori-motor state (sms), is associated a feature. It thus comes naturally to describe a trajectory by the fraction of overall time it spends in every sensori-motor state[1].

Letting D denote the number of sms, each trajectory generated from π_θ thus is represented as a unit vector in $[0,1]^D$. In all generality, the behavioural representation associated to parametric policy π_θ, noted μ_{π_θ} is the distribution over $[0,1]^D$ of all trajectories generated from policy π_θ, reflecting the actuator and sensor noise, and possibly the presence and actions of other robots in the swarm. In this chapter however, the indeterminacy of the policy-to-log mapping (due to the influence of the starting position or the actuator and sensor noise) is neglected for the sake of simplicity and deferred to chapter 7. In practice, the indeterminacy depending on the starting point of the trajectory is discarded, either by requiring all demonstrations to have same starting point, or by considering sufficiently long trajectories and discarding the beginning thereof, the so-called burn-in period.

Formally, the proposed *behavioural representation* (BvR) maps each policy onto

[1] The use of the time fraction is chosen for simplicity; one might use instead the *discounted* cumulative time spent in every sms.

a real valued vector ($\mu : \pi \mapsto \mu_\pi \in \mathbb{R}^d$), as follows. Let us consider the datastream generated from the robot equipped with a policy π, reporting the sensor and actuator values observed at each time step. This datastream, a.k.a. robotic log, is most often processed offline; in a resource-bounded context, it makes sense to process it online. The policy demonstration is represented by a histogram, associating to each sensori-motor state the fraction of time spent in this sms. The BvR representation complies with a resource-constrained framework and it is agnostic w.r.t. the policy return and the environment. The number of sensori-motor states exponentially increases with the clustering precision ε and the intrinsic dimension of the sensori-motor space; it can however be controlled to some extent as ε is set by the user.

Let the sensori-motor data stream SMS(π) consist of the robot sensor and actuator values recorded at each time step, with SMS(π) be given as $\{y_1, \dots y_H\}$, with $y_h \in \mathbb{R}^b$, where b is the number of sensors and actuators of the robot and y_h denotes the concatenation of the sensory and motor data of the robot at time step [2] h.

As mentioned, an ε-clustering algorithm ($\varepsilon > 0$) [Duda & Hart 1973] is used to incrementally define a set $\mathcal{S}$ of 'exemplar' sensori-motor states s_i from all SMS(π) data streams generated from policies $\pi_1, \dots \pi_t$. For each considered policy π, in each time step h, y_h is compared to all elements of $\mathcal{S}$, and it is added to $\mathcal{S}$ if $min\{||y_h - s_i||_\infty,\ s_i \in \mathcal{S}\} > \varepsilon$. Let n_t denote the number of states at time t. BvR is defined as follows:

$$\begin{aligned} &\text{Given SMS}(\pi) = \{y_1, \dots y_H\} \text{ and } \mathcal{S} = \{s_1, \dots s_{n_t}\} \\ &\quad \mu : \pi \mapsto \mu_\pi \in [0,1]^{n_t} \\ &\quad \mu_\pi[i] = \frac{1}{H} |\{y_h \ s.t.\ ||y_h - s_i||_\infty < \varepsilon\}| \end{aligned}$$

BvR differs from the feature counts used in [Abbeel & Ng 2004, Kolter *et al.* 2007] as features are learned here from policy trajectories as opposed to being defined from prior knowledge. Compared to the discriminant policy representation built from the set of policy demonstrations [Levine *et al.* 2010], the main two differences are that BvR is independent of the policy return estimate, and that it is built online as new policies explore new regions of the sensori-motor space. As the number of states n_t, and thus BvR dimension, increase along time, BvR consistency follows from the fact that $\mu_\pi \in [0,1]^{n_t}$ is naturally mapped onto

[2] A temporal relaxation is used to filter out sensor noise: $y_h = (1-\eta)\, y_{h-1} + \eta\, z_h$ where z_h denotes the instant sensor and actuator values at time step h.

$(\mu_\pi, 0)$ in $[0,1]^{n_t+1}$ (the i-th coordinate of μ_π is 0 if s_i was not discovered at the time π was demonstrated).

The price to pay for the representation consistency is that the number of states exponentially increases with precision parameter ε; on the other hand, it increases like $\varepsilon^{b'}$ where b' is the intrinsic dimension of the sensori-motor data.

5.5 Policy Return Estimate Learning

For the sake of notational simplicity, μ_i will be used instead of μ_{π_i} when no confusion is to be feared. Let the policy archive Π_t be given as $\{\mu_1, \ldots \mu_t\}$ at step t [3], assuming with no loss of generality that the μ_i's are ordered after the expert preferences. Constraint archive $\mathcal{C}_t$ thus includes the set of $\frac{t(t-1)}{2}$ ordering constraints $\mu_i \succ \mu_j$ for $i > j$.

The policy return estimate J_t is sought as a linear mapping $J_t(\mu) = \langle \mathbf{w}_t, \mu \rangle$ with $\mathbf{w}_t \in \mathbb{R}^{n_t}$. Using a standard constrained convex optimization formulation [Bakir *et al.* 2006, Joachims 2006], w_t is the solution of the following problem:

$$\begin{cases} \text{Minimize} & \frac{1}{2}\ ||\mathbf{w}||^2 + C \sum_{i,j=1, i>j}^{t} \xi_{i,j} \\ \text{subject to} & (\langle \mathbf{w}, \mu_i \rangle - \langle \mathbf{w}, \mu_j \rangle \geq 1 - \xi_{i,j}) \text{ and } (\xi_{i,j} \geq 0) \text{ for all } i > j \end{cases} \tag{5.1}$$

This policy return estimate J_t features some good properties. Firstly, it is consistent, despite the fact that the dimension n_t of the state space might increase with t. After the same arguments as above, the $\mathbf{w}_t$ coordinate related to a state which has not yet been discovered is set to 0. Secondly, by construction, it is independent on the policy parameterization since it only depends on the sensor and actuator space; it can thus be transferred among different policy spaces. Likewise, it does not involve any external information that cannot be made available to the robot itself. Therefore it can be computed on-board and provides the robot with a self-assessment.

Finally, although J_t is defined at the global policy level, $\mathbf{w}_t$ provides some valuation of the sensori-motor states. If a high positive weight $\mathbf{w}_t[i]$ is associated to the i-th sensori-motor state s_i, this state is considered to significantly and positively contribute to the quality of a policy, *comparatively to the policies considered so far*.

[3] In the absence of noise, the trajectories, rather than the policies themselves, will be stored in the archive, and in the constraint archive C_t.

Learning J_t thus significantly differs from learning the optimal RL value function V^*. By definition, $V^*(s_i)$ estimates the maximum expected cumulative reward ahead of state s_i, which can only increase as better policies are discovered. In contrast, $\mathbf{w}_t[i]$ reflects the fact that visiting state s_i is conducive to discovering better policies, comparatively to policies viewed so far by the expert. In particular, $\mathbf{w}_t[i]$ can increase *or decrease* along t; some states might be considered as highly beneficial in the early stages of the robot training, and become useless and be discarded later on (more on this in section 7.6.5, around Fig. 7.15).

5.6 New Policy Generation

The policy generation step can be thought of in terms of self-training. The generation of new policies relies on black-box optimization; expected-global improvement methods [Brochu *et al.* 2008] and gradient methods [Peters & Schaal 2008] are not applicable since the policy return estimate is not defined on the parametric representation space Θ. New policies are thus generated using a stochastic black-box optimization method, more precisely a $(1 + \lambda)$-ES algorithm [Auger 2005]. Formally, a Gaussian distribution $\mathcal{N}(\theta_c, \Sigma)$ on Θ is maintained, the center θ_c and the covariance matrix Σ of this distribution are updated after the parameter θ of the current best policy. For n_g iterations, λ policies π are drawn after $\mathcal{N}(\theta_c, \Sigma)$, they are executed in order to compute their behavioural description μ_π, and the best one after an optimisation criterion $\mathcal{F}$ (see below) is used to update $\mathcal{N}(\theta_c, \Sigma)$. After n_g iterations of the policy generation step, the best policy after $\mathcal{F}$ denoted π_{t+1} is selected and demonstrated to the expert, and the whole PPL process is iterated.

The criterion $\mathcal{F}$ used to select the best policy out of the current λ policies is meant to enforce some trade-off between the exploration of the policy space and the exploitation of the current policy return estimate J_t (note that the discovery of new sensori-motor states is worthless after J_t since their weight after $\mathbf{w}_t$ is 0). Taking inspiration from [Auger 2005], $\mathcal{F}$ is defined as the sum of the policy return estimate J_t and a weighted exploration term E_t, measuring the diversity Δ of the policy w.r.t the policy archive Π_t:

$$\begin{aligned} \text{Maximize } \mathcal{F}(\mu) = &\quad J_t(\mu) + \alpha_t E_t(\mu) \quad \alpha_t > 0 \\ \text{with } E_t(\mu) = &\quad \min \{\Delta(\mu, \mu_u),\ \mu_u \in \Pi_t\} \\ \alpha_t = &\quad \begin{cases} c.\alpha_{t-1} & \text{if } \pi_t \succ \pi_{t-1} \quad c > 1 \\ \frac{1}{c^{1/p}} \alpha_{t-1} & \text{otherwise} \end{cases} \end{aligned}$$

Parameter α_t controls the **exploration vs exploitation tradeoff**, accounting for the fact that both the policy distribution and the objective function J_t are non stationary. Accordingly, α_t is increased or decreased by comparing the empirical success rate (whether π_t improves on all policies in the archive) with the expected success rate of a reference function (usually the sphere function, on which the proved optimal success rate is close to 0.2 [Auger 2005]). When the empirical success rate is above (respectively below) the reference one, α_t is increased (resp. decreased). This is de facto achieved here through parameter p, empirically adjusted so that p failures cancel out one success and bring α_t back to its original value (leading for instance to $p = \frac{1}{4}$ to reach the $\frac{1}{5}$ value for the sphere function).

Finally, the **diversity function** $\Delta(\mu, \mu')$ is defined as follows. Let a, b, c respectively denote the number of states visited by μ only, by μ' only, and by both μ and μ'. Then

$$\Delta(\mu, \mu') = \frac{a + b - c}{\sqrt{a + c}\sqrt{b + c}}$$

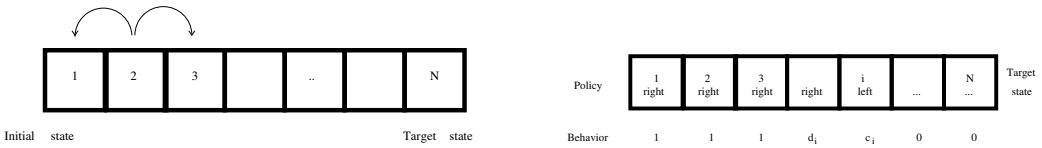
i.e., $\Delta(\mu, \mu')$ computes and normalizes the symmetric difference minus the intersection of the two sets of states visited by μ and μ'.

5.7 Initialization

The PPL initialization is another challenging step, as policy behaviours corresponding to uniformly drawn parameters θ are usually quite uninteresting, and therefore difficult to rank. The first two policies are thus generated as follows.

First, let us define the information quantity J_0 of a policy μ as

$$J_0(\mu) = -\sum_{i=1}^{d} \mu[i] \log \mu[i]$$



(a) State and Action Space (b) Parametric and behavioural representation

Figure 5.1: The RiverSwim problem. The parametric representation of a policy π specifies the action (go left or right) selected in each state. The behavioural representation μ_π is $(1, ...1, 0, ..0)$, with the rightmost 1 at position $\ell(\pi)$, defined as the leftmost state associated to the *left* move.

Now given a set P_0 of randomly generated policies, π_1 is selected as the policy in P_0 with maximal information quantity J_0. Then policy π_2 is the one in P_0 with maximum diversity w.r.t. π_1, i.e.

$$\pi_2 = argmax\,\{\Delta(\mu, \pi_1), \mu \in P_0\}$$

The rationale for this initialization is that π_1 should experiment as many distinct sensorimotor states as possible; and π_2 should experiment as many sensorimotor states different from that of π_1 as possible, to facilitate the expert ranking, and yield an informative policy return estimate J_2.

Admittedly, the use of the information quantity and more generally BvR, only make sense if the robot faces a sufficiently rich environment. In an empty environment, i.e. when there is nothing to see, the robot can only visit a single sensori-motor state, and nothing can happen.

5.8 Convergence study

PPL convergence is analytically studied and proved in the context of the artificial RiverSwim problem [Strehl *et al.* 2006]. This problem involves N states and two actions, going *left* or *right*; starting from the leftmost state, the goal is to reach the rightmost state (Fig. 5.1). While the policy parametric representation space is $\{\text{left, right}\}^N$, the behaviour of the policy π is entirely characterized from the leftmost state i where it goes left, denoted $\ell(\pi)$. In the following, we only consider the boolean BvR, with $\mu_\pi = \{1, 1, .., 1, 0, ..0\}$ and $\ell(\pi)$ the index of the rightmost 1. For notational simplicity, let μ_π and $\mu_{\pi'}$ be noted μ and μ' in the following. By

definition, the expert prefers the one out of μ and μ' which goes farther on the right ($\mu \succ \mu'$ iff $\ell(\mu) > \ell(\mu')$).

In the RiverSwim setting, problem (5.1) (section 5.5) can be solved analytically. Let archive Π_t be given as $\{\mu_1 \dots \mu_t\}$ and assume w.l.o.g. that the μ_i's are ordered by increasing value of $\ell(\mu_i)$. It comes immediately that constraints related to adjacent policies ($\langle \mathbf{w}, \mu_{\mathbf{i+1}} \rangle - \langle \mathbf{w}, \mu_i \rangle \geq 1 - \xi_{i,i+1}$ for $1 \leq i < t-1$) entail all other constraints, and can be all satisfied by setting $\sum_{k=\ell(\mu_i)+1}^{\ell(\mu_{i+1})} \mathbf{w}[k] = 1$; slack variables $\xi_{i,j}$ thus are 0 at the optimum.

Problem (5.1) can thus be decomposed into $t-1$ independent problems involving disjoint subsets of indices $]\ell(\mu_i), \ell(\mu_{i+1})]$; its solution $\mathbf{w}$ is given as:

$$\mathbf{w}[k] = \begin{cases} 0 & \text{if } \; k < \ell(\mu_1) \text{ or } k > \ell(\mu_t) \\ \frac{1}{\ell(\mu_{i+1}) - \ell(\mu_i)} & \text{if } \ell(\mu_i) < k \leq \ell(\mu_{i+1}) \end{cases}$$

Proposition 1: In the RiverSwim setting, $J_t(\mu) = \langle \mathbf{w}_t, \mu \rangle$ where $\mathbf{w}_t$ satisfies:

$$\begin{cases} \mathbf{w}_t[k] = 0 \text{ if } k \leq \ell(\mu_1) \text{ or } k > \ell(\mu_t) \\ \mathbf{w}_t[k] > \frac{1}{N} \text{ if } \ell(\mu_1) < k \leq \ell(\mu_t \\ \sum_{k=1}^{i=N} \mathbf{w}_t[k] = t \end{cases}$$

Policy generation (section 5.6) considers both the current J_t and the diversity E_t respective to the policy archive Π_t, given as $E_t(\mu) = min\{\Delta(\mu, \mu_u), \mu_u \in \Pi_t\}$. In the RiverSwim setting, it comes:

$$\Delta(\mu, \mu_u) = \frac{|\ell(\mu) - \ell(\mu_u)| - \min(\ell(\mu), \ell(\mu_u))}{\sqrt{\ell(\mu)\ell(\mu_u)}}$$

For $i < j$, $\frac{|i-j| - \min(i,j)}{\sqrt{ij}} = \sqrt{\frac{j}{i}} - 2\sqrt{\frac{i}{j}}$. It follows that the function $i \mapsto \frac{\sqrt{|\ell(\mu)-i|} - \min(\ell(\mu), i)}{\sqrt{\ell(\mu)\, i}}$ is strictly decreasing on $[1, \ell(\mu)]$ and strictly increasing on $[\ell(\mu), +\infty)$. It reaches its minimum value of -1 for $i = \ell(\mu)$. As a consequence, for any policy μ in the archive, $E_t(\mu) = -1$.

Let μ_t and μ^* respectively denote the best policy in the archive Π_t and the policy that goes exactly one step further right. We will now prove the following result:

Proposition 2. The probability that μ^* is generated at step t is bounded from below by $\frac{1}{eN}$. Furthermore, after μ^* has been generated, it will be selected according to the selection criterion $\mathcal{F}_t$ (section 5.6), and demonstrated to the expert.

Proof

Generation step: Considering the boolean parametric representation space $\{0,1\}^N$, the generation of new policies proceeds using the standard bitflip mutation, flipping each bit of the current μ_t with probability $\frac{1}{N}$. The probability to generate μ^* from μ_t is lower-bounded by $\frac{1}{N}(1-\frac{1}{N})^{\ell(\mu_t)}$ (flip bit $\ell(\mu_t)+1$ and do not flip any bit before that). Furthermore, $(1-\frac{1}{N})^{\ell(\mu_t)} > (1-\frac{1}{N})^{N-1} > \frac{1}{e}$ and hence the probability to generate μ^* from μ_t is lower-bounded by $\frac{1}{eN}$.

Selection step. As shown above, $E_t(\mu^*) > -1$ and $E_t(\mu) = -1$ for all $\mu \in \Pi_t$. As the candidate policy selection is based on the maximization of $\mathcal{F}_t(\mu) = \langle \mathbf{w}_t, \mu \rangle + \alpha_t E_t(\mu)$, and using $\langle \mathbf{w}_t, \mu_t \rangle = \langle \mathbf{w}_t, \mu_{t+1} \rangle = t$, it follows that $\mathcal{F}_t(\mu_t) < \mathcal{F}_t(\mu^*)$, and more generally, that $\mathcal{F}_t(\mu) < \mathcal{F}_t(\mu^*)$ for all $\mu \in \Pi_t$. Consider now a policy μ that is not in the archive though $\ell(\mu) < t$ (the archive does not need to contain all possible policies). From Proposition 1, it follows that $\langle \mathbf{w}, \mu \rangle < t - \frac{1}{N}$. Because $E_t(\mu)$ is bounded, there exists a sufficiently small α_t such that $\mathcal{F}_t(\mu) = \langle \mathbf{w}_t, \mu \rangle + \alpha_t E_t(\mu) < \mathcal{F}_t(\mu^*)$. □

Furthermore, thanks to the monotonicity of $E_t(\mu)$ w.r.t. $\ell(\mu)$, we know that $\mathcal{F}_t(\mu_{t+i}) < \mathcal{F}_t(\mu_{t+j})$ for all $i < j$ where $\ell(\mu_{t+i}) = \ell(\mu_t) + i$. Better RiverSwim policies will thus be selected along the policy generation and selection step.

5.9 Experimental Validation

This section reports on the experimental validation of the PPL approach. For the sake of reproducibility, all reported results have been obtained using the publicly available simulator Roborobo [Bredeche 2006].

5.9.1 Experiment Goals and Experimental Setting

The experiments are meant to answer three main questions. The first one concerns the feasibility of PPL compared to baseline approaches. The use as surrogate optimization objective of the policy return estimate, learned from the expert preferences, is compared to the direct use of the expert preferences. The performance indicator

is the *speed-up* in terms of sample complexity (number of calls to the expert), needed to reach a reasonable level of performance.

The second question regards the merits of the behavioural representation comparatively to the parametric one. More specifically, the point is to know whether BvR can enforce an effective trade-off between the number of sensori-motor states and the perceptual aliasing, enabling an accurate policy return estimate to be built from few orderings.

A third question regards the merits of the exploration term used in the policy generation (self-training phase, section 5.6), and the entropy-based initialization process (section 5.7), and how they contribute to the overall performance of PPL.

Three experiments are considered to answer these questions. The first one is an artificial problem which can be viewed as a 2D version of the RiverSwim [Strehl *et al.* 2006]. The second experiment, inspired from the novelty search experiment in [Lehman & Stanley 2008], involves a robot in a maze. The goal is to reach a given location. The third experiment involves two robots, and the goal is to enforce a coordinated exploration of the arena by both robots. In all experiments the robotic architecture is a Cortex-M3 with 8 infra-red (IR) sensors and two motors respectively controlling the rotational and translation speed. The IR range is 6 cm; the surface of the arena is 6.4 square meters.

All policies are implemented as a multi-layer perpectron (a 1-hidden-layer feed-forward neural net), using the 8 IR sensors (and a bias) as inputs, with 10 neurons on the hidden layer, and giving the two motor commands as output, resulting in 121 weights ($\Theta = \mathbb{R}^{121}$). A random policy is built by uniformly selecting each weight in $[-1, 1]$.

BvR is built using the L_∞ norm and $\varepsilon = 0.45$; the number of sensori-motor states, thus the BvR dimension, is below 1,000. The policy return estimate is learned using SVM^{rank} library [Joachims 2006] with linear kernel and default parameter $C = 1$ (section 5.5).

The optimization algorithm used in the policy generation (section 5.6) and initialization (section 5.7) is a $(1+\lambda)$-ES [Schwefel 1981] with $\lambda = 7$. The variance of the Gaussian distribution is initialized to .3, and increased by a factor of 1.5 upon any improvement and decreased by a factor of $1.5^{-1/4}$ otherwise. The $(1 + \lambda)$-ES is used for n_g iterations in each policy generation step, with $n_g = 10$ in the maze

experiment, and $n_g = 5$ in the two-robot experiment. It is used for 20 iterations in the initialization step.

All presented results are averaged over 41 independent runs. Five algorithms are compared: PPL_{BvR} with entropy-based initialization and exploration term, where the policy return estimate relies on the behavioural representation; PPL_{param}, which only differs from PPL_{BvR} in that the policy return estimate is learned using the parametric representation; $\text{PPL}_{w/o\ i}$, which only differs from PPL_{BvR} in that it uses a random uniform initialization; *Expert-Only*, where the policy return estimate is replaced by the true expert preferences; and *Novelty-Search* [Lehman & Stanley 2008] which can be viewed as an exploration-only approach: at each time step, the candidate policy with highest average diversity compared to its k-nearest neighbors in the policy archive is selected (we used $k = 1$ for the 2D RiverSwim, and $k = 5$ for both other experiments).

5.9.2 2D RiverSwim

In this 4×4 grid world, the robot starts in the lower left corner of the grid; the sought behaviour is to reach the upper right corner and to stay there. The expert preference goes toward the policy going closer to the goal state, or reaching earlier the goal state. The action space includes the four move directions with deterministic transitions (the robot stays in the same position upon marching to the wall). For a time horizon $H = 16$, the BvR includes only 6334 distinguishable policies (to be compared with the actual 4^{16} distinct policies). The small size of this grid world allows one to enumeratively optimize the policy return estimate, with and without the exploration term. Fig. 5.2 (left) displays the true policy return *vs* the number of calls to the expert. In this artificial problem, *Novelty-Search* outperforms PPL and discovers an optimal policy in the 20-th step. PPL discovers an optimal policy at circa the 30-th step. The exploration term plays an important role in PPL performance; Fig. 5.2 (right) displays the average weight α_t of the exploration term. In this small-size problem, a mere exploration strategy is sufficient and typically *Novelty-Search* discovers two optimal policies in the first 100 steps on average; meanwhile, PPL discovers circa 25 optimal policies in the first 100 steps on average.

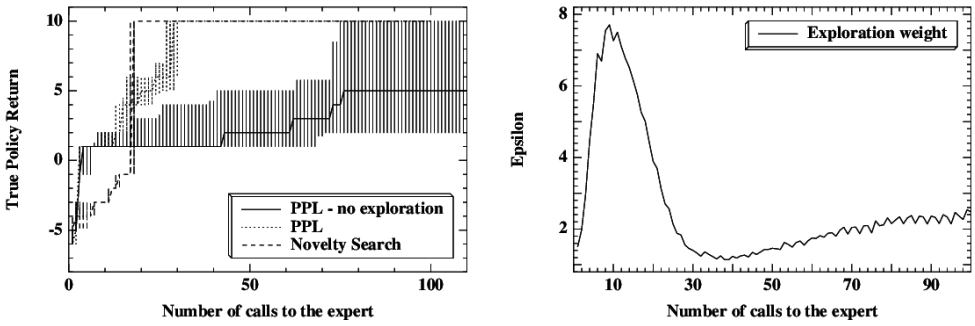


Figure 5.2: 2D RiverSwim: Performance of PPL and *Novelty-Search*, averaged over 41 runs with std. dev (left); average weight of the exploration term α_t in PPL (right).

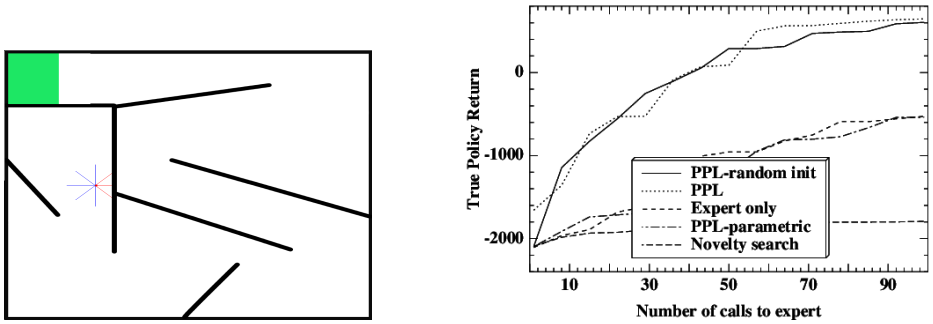


Figure 5.3: The maze experiment: the arena (left) and best performances averaged out of 41 runs.

5.9.3 Reaching the End of the Maze

In this experiment inspired from [Lehman & Stanley 2008], the goal is to traverse the maze and reach the green zone when starting in the opposite initial position (Fig. 5.3-left). The time horizon H is 2,000; an oracle robot needs circa 1,000 moves to reach the target location. As in the RiverSwim problem, the expert preference goes toward the policy going closer to the goal state, or reaching earlier the goal state; the comparison leads to a tie if the difference is below 50 distance units or 50 time steps. The "true" policy return is the remaining time when it first reaches the green zone, if it reaches it, minus the min distance of the trajectory to the green zone, otherwise. The main difficulty in this maze problem is the severe perceptual aliasing; all locations situated far from the walls look alike due to the IR sensor limitations.

This experiment supports the feasibility of the PPL scheme. A speed-up factor

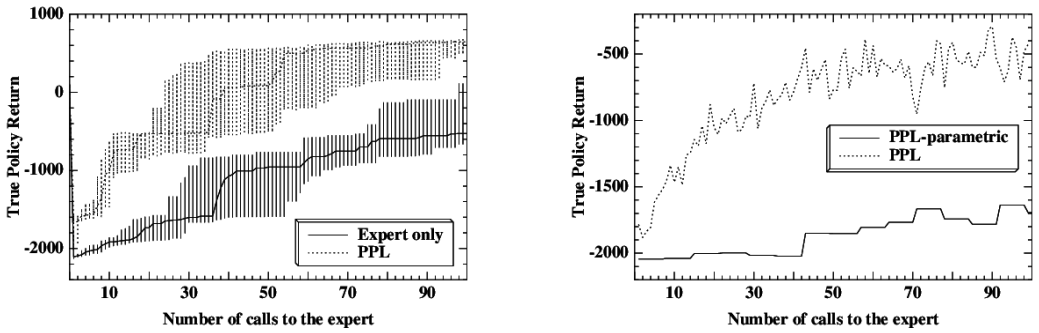


Figure 5.4: The maze experiment: PPL vs *Expert-Only* (left); Accuracy of parametric and behavioural policy return estimate(right)

of circa 3 is observed compared to *Expert-Only*; on average PPL reaches the green zone after demonstrating 39 policies to the expert, whereas *Expert-Only* needs 140 demonstrations (Fig. 5.4, left). This speed-up is explained as PPL filters out unpromising policies. *Novelty-Search* performs poorly on this problem comparatively to PPL and even comparatively to *Expert-Only*, which suggests that *Novelty-Search* might not scale up well with the size of the policy space.

This experiment also establishes the merits of the behavioural representation. Fig. 5.4 (right) displays the average performance of PPL_{BvR} and PPL_{param} when no exploration term is used, to only assess the accuracy of the policy return estimate. As can be seen, learning is very limited when done in the parametric representation by PPL_{param}, resulting in no speedup compared to *Expert-Only* (Fig. 5.3, right). A third lesson is that the entropy-based initialization does not seem to bring any significant improvement, as PPL_{BvR} and $\text{PPL}_{w/o\ i}$ get same results after the very first steps (Fig. 5.3).

This conclusion should naturally be tempered by the relative simplicity of the task.

5.9.4 Synchronized Exploration

This experiment investigates the feasibility of two robots exploring an arena (Fig. 5.5, left) in a synchronous way, using the same policy. The expert preferences are emulated by associating to a policy the number of distinct 25x25 tiles explored by any of the two robots conditionally to the fact that their distance is below 100

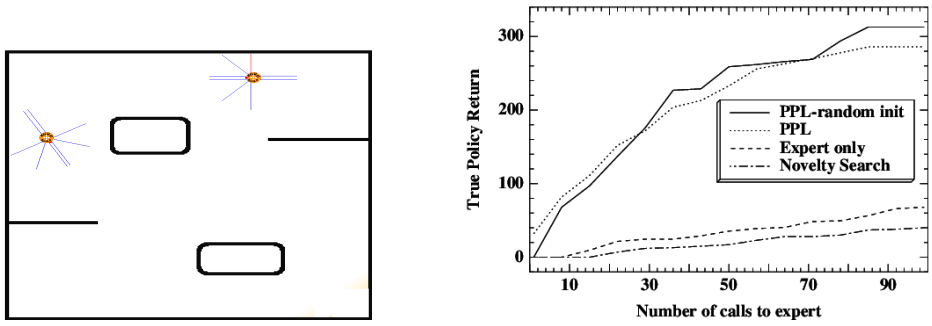


Figure 5.5: Synchronous exploration: the arena (left) and the best average performance out of 41 runs (right).

unit distances at the moment the tile is explored. Both robots start in (distinct) random positions. The difficulty of the task is that most policies wander in the arena, making it unlikely for the two robots to be anywhere close to one another at any point in time.

The experimental results (Fig. 5.5, right) mainly confirm the lessons drawn from the previous experiment. PPL improves on *Expert-Only*, with a speed-up circa 10, while *Novelty-Search* is slightly but significantly outperformed by *Expert-Only*. In the meanwhile, the entropy-based initialization does not make much of a difference after the first steps, and might even become counter-productive in the end.

5.10 PPL: Summary and Discussion

The PPL approach successfully demonstrates the feasibility of learning i) in a simulator-free setting; ii) with the only support of a "weak" expert unable to demonstrate the targeted behaviours (as opposed to the Inverse Reinforcement Learning setting).

The interaction with the expert yields to estimate a policy return estimate, supporting a direct policy search approach. On the one hand the convergence of the approach can be established in the artificial RiverSwim framework [Strehl *et al.* 2006]. On the other hand the experimental validation demonstrates that PPL can be used effectively to learn elementary behaviours involving one or two robots. Most interestingly, PPL yields behaviours which require perceptual primitives (discriminating another robot from the wall), although directly learning these primitives along a

standard discriminant learning setting eluded our efforts.

PPL was made possible through the original unsupervised *behavioural representation* BvR, based on the compression and ε-clustering of the robot trajectories. The experimental results confirm that using the controller parametric representation to learn the policy return estimate is ineffective, due to the size of the parametric input space compared to the number of preference constraints. The PPL limitations are related to the trade-off between the size of the input space, and the number of preference queries. On the one hand, the acquisition of precision skills would require a finer-grained BvR, by e.g. decreasing the granularity ε of the sensori-motor states; unfortunately, the number of sms exponentially increases with ε. On the other hand, for the PPL framework to reach out, the number of preference queries to the expert needed to acquire basic skills needs to be decreases. How to minimize this number and the expert's burden, will be the focus of chapter 6.

Regarding specifically the PPL framework, a perspective opened for further research is to gradually refine BvR, specifically using quad-trees to describe and retrieve the sensori-motor states while adaptively adjusting their granularity, or taking inspiration from the Robust Intrinsic Motivation [Oudeyer *et al.* 2010].

A second perspective is to explicitly exploit the policy return estimate, which provides an embedded *system of values* computable on-board and possibly supporting repair heuristics (e.g. to adjust the policy and compensate for the fatigue of the actuators) along a lifelong learning approach.

A third, minor, perspective is to save the experiment time, by adjusting the length of the self-training phases, taking inspiration from online learning on a budget [Dekel *et al.* 2008].

Chapter 6

Active Preference based ReInforcement Learning

Foreword

This chapter introduces the second contribution of the PhD, the *Active Preference-learning based Reinforcement Learning* (APRIL) algorithm first presented in [Akrour *et al.* 2012].

APRIL focuses on alleviating the expert's burden, and specifically on decreasing the number of ranking queries to the expert needed to yield a satisfactory policy. Note that this goal – decreasing the number of ranking queries – does not coincide with that of Active Ranking, for the following reason: the criterion to be optimized by active ranking is the precision and accuracy of the ranking model. Quite the contrary, APRIL is not interested in the accuracy of the ranking model (here the policy return estimate), which is but a means for the actual end, finding an optimal controller.

Along this line, APRIL takes inspiration from sequential model-based optimization and more specifically from *Optimal Bayesian Recommendation Sets* [Viappiani & Boutilier 2010]. The main idea is to consider the greedy *Expected Utility of Selection* (what is the best expected performance among two controllers, depending on the expert's preference about their behaviors), and the one-step look-ahead *Expected Posterior Utility* (what is the best performance one can get, based on the expert's preference among those two behaviors).

The key issue addresed in this chapter is the extension of the EPU-EUS criterion optimization to a high-dimension continuous search space in a tractable and efficient way.

The robustness of the proposed approximate active ranking criterion is first

assessed on an artificial problem. Its integration within APRIL is thereafter studied and a proof of concept of APRIL is given on the classical mountain car problem, and the cancer treatment testbed first introduced by [Zhao *et al.* 2009], showing that a couple dozen of preference queries is sufficient to yield competitive results compared to Inverse Reinforcement Learning [Abbeel & Ng 2004].

6.1 Motivations

Resuming the Preference-based Policy Learning (PPL) approach presented in chapter 5 [Akrour *et al.* 2011], our goal is to extend PPL along the lines of active learning, in order to minimize the number of expert's ranking feedbacks needed to learn a satisfactory policy. However our primary goal is to learn a competent policy; learning an accurate policy return is simply a way to learn an accurate policy. More than active learning *per se*, our goal thus relates to interactive optimization [Brochu *et al.* 2008] and online recommendation [Viappiani & Boutilier 2010]. The Bayesian settings used in these related works (section 6.2) will inspire the proposed *Active Preference-based ReInforcement Learning* (APRIL) algorithm.

The difficulty is twofold. Firstly, the above Bayesian approaches [Brochu *et al.* 2008, Viappiani & Boutilier 2010] hardly scale up to large-dimensional continuous spaces. Secondly, the PPL setting requires one to consider two different search spaces. Basically, RL is a search problem on the policy space $\mathcal{X}$, mappings of the state space on the action space. However, the literature underlines that complex policies can hardly be expressed in the state $\times$ action space for tractability reasons [Littman *et al.* 2002]. A thoroughly investigated alternative is to use parametric representations (see e.g. [Peters & Schaal 2008] among many others), for instance using the weight vectors of a neural net as policy search space $\mathcal{X}$ ($\mathcal{X} \subset \mathbb{R}^d$, with d in the order of thousands). Unfortunately, the experiments presented in chapter 5 have confirmed that parametric policy representations might be ill-suited to learn a preference-based policy return. The failure to learn an accurate preference-based policy return on the parametric space is explained as the expert's preferences essentially relate to the policy behavior, on the one hand, and the policy behavior depends in a highly non-linear or even non-smooth way on its parametric description, on the other hand. Indeed, small modifications of a neural

weight vector $\mathbf{x}$ can entail arbitrarily large differences in the behavior of policy π_x, depending on the robot environment (the tendency to turn right or left in front of an obstacle might have far fetched impact on the overall robot behavior).

The PPL framework thus requires one to simultaneously consider the parametric representation of policies (defining the policy search space) and the behavioral representation of policies (where the policy return, i.e., objective to be optimized, can be learned accurately). The distinction between the parametric and the behavioral spaces is reminiscent of the distinction between the input and the feature spaces, at the core of the celebrated kernel trick [Cortes & Vapnik 1995]. Contrasting with the kernel framework, however, the mapping Φ (mapping the parametric representation $\mathbf{x}$ of policy π_x onto the behavioral description $\Phi(\mathbf{x})$ of policy π_x) is non-smooth[1].

In order for PPL to apply the abovementioned Bayesian approaches used in interactive optimization [Brochu *et al.* 2008] or online recommendation [Viappiani & Boutilier 2010], where the objective function is defined on the search space, one should thus solve the inverse parametric-to-behavioral mapping problem and compute Φ^{-1}. However, computing such inverse mappings is notoriously difficult in general [Tsochantaridis *et al.* 2005]; it is even more so in the RL setting as it boils down to inverting the generative model.

The goal of the APRIL algorithm eventually is to propose a tractable approximation of the Bayesian setting used in [Brochu *et al.* 2008, Viappiani & Boutilier 2010], consistent with the parametric-to-behavioral mapping.

6.2 Overview and analysis

As detailed in chapter 5, PPL iteratively determines the next candidate policy π_{t+1} by heuristically optimizing a weighted sum of the current policy return J_t, and the diversity w.r.t. archive $\mathcal{U}_t$.

While PPL and Inverse Reinforcement Learning (IRL) [Abbeel & Ng 2004] both iteratively learn a policy return and a candidate policy in each phase, the difference is threefold. Firstly, IRL starts with an optimal trajectory u^* provided by the human expert (which dominates all policies built by the agent by construction) whereas PPL

[1] Interestingly, policy gradient methods face the same difficulties, and the guarantees they provide rely on the assumption of a smooth Φ mapping [Peters & Schaal 2008].

is iteratively provided with bits of information (this demonstration is/is not better than the previous best demonstration) by the expert. Secondly, in each iteration IRL solves an RL problem using a generative model, whereas PPL achieves direct policy learning. Thirdly, IRL is provided with an informed representation of the state space. With respect to preference-based value learning [Cheng *et al.* 2011] (chapter 5), the difference is that [Cheng *et al.* 2011] defines an order relation on the action space depending on the current state and the current policy, whereas PPL defines an order relation on the policy space.

Our goal is to extend PPL in the sense of the maximum Expected Value of Information (EVOI) proposed by [Viappiani 2012], where a linear utility function J is learned on a low dimensional search space $\mathcal{X} = \mathbb{R}^D$, with $J(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$ and $\mathbf{w}$ a vector in $\mathbb{R}^D$. Within the Bayesian setting, the uncertainty about the utility function is expressed through a belief θ defining a distribution over the space of utility functions ($\equiv \mathbb{R}^D$).

As detailed in chapter 4, the problem of (iterated) optimal choice queries is to simultaneously learn the user's utility function, and present the user with a set of good recommendations, such that she can select one with maximal expected utility. Viewed as a single-step (greedy) optimization problem, the goal thus boils down to finding a recommendation $\mathbf{x}$ with maximal expected utility $\mathbb{E}_\theta[\langle \mathbf{w}, \mathbf{x} \rangle]$. In a global optimization perspective however [Viappiani & Boutilier 2010], the goal is to find a set of recommendations $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ with maximum expected *posterior utility*, defined as the expected gain in utility of the next decision. The expected utility of selection (EUS) is studied in [Viappiani & Boutilier 2010] under several noise models, and the authors show that the greedy optimization of EUS provides good approximation guarantees of the optimal query.

In [Viappiani 2012], the issue of the maximum expected value of information (EVOI) is tackled, and the authors consider the following criterion, where $\mathbf{x}^*$ is the current best solution: select $\mathbf{x}$ maximizing

$$EUS(\mathbf{x}) = \mathbb{E}_{\theta, x > x^*}[\langle \mathbf{w}, \mathbf{x} \rangle] + \mathbb{E}_{\theta, x < x^*}[\langle \mathbf{w}, \mathbf{x}^* \rangle] \tag{6.1}$$

Eq. (6.1) thus measures the expected utility of $\mathbf{x}$, distinguishing the case where $\mathbf{x}$ actually improves on $\mathbf{x}^*$ (l.h.s.) and the case where $\mathbf{x}^*$ remains the best solution (r.h.s.). This criterion can be understood by reference to active learning and

the so-called splitting index criterion [Dasgupta 2005]: Within the realizable setting (the solution lies in the version space of all hypotheses consistent with all examples so far), an unlabeled instance $\mathbf{x}$ splits the version space into two subspaces: that of hypotheses labelling $\mathbf{x}$ as positive, and that of hypotheses labelling $\mathbf{x}$ as negative. The ideal case is when instance $\mathbf{x}$ splits the version space into two equal size subspaces; querying $\mathbf{x}$ label thus optimally prunes the version space. In the general case, the splitting index associated to $\mathbf{x}$ is the relative size of the smallest subspace: the larger, the better. In active ranking, any instance $\mathbf{x}$ likewise splits the version space into two subspaces: the challenger subspace of hypotheses ranking $\mathbf{x}$ higher than the current best instance $\mathbf{x}^*$, and its complementary subspace. In the Bayesian setting, considering an interactive optimization goal, the stress is put on the expected utility of $\mathbf{x}$ on the challenger subspace, plus the expected utility of $\mathbf{x}^*$ on the complementary subspace.

The main challenge is to express this criterion in a sound and computationally efficient way.

6.3 APRIL Overview

Like PPL (chapter 5), Active Preference-based Reinforcement Learning (APRIL) is an iterative algorithm alternating a demonstration and a self-training phase. The only difference between PPL and APRIL lies in the self-training phase.

This section first discusses the parametric and behavioral policy representations. It thereafter presents an Approximation of the Expected Utility of Selection criterion (AEUS) used to select the next candidate policy to be demonstrated to the expert, which overcomes the intractability of the EUS criterion (section 6.2) with regard to these two representations.

6.3.1 Parametric and Behavioral Policy Spaces

Like PPL, APRIL considers two search spaces. The first one noted $\mathcal{X}$, referred to as input space or parametric space, is suitable to generate and run the policies. In the following $\mathcal{X} = \mathbb{R}^d$; policy π_x is represented by e.g. the weight vector $\mathbf{x}$ of a neural net or the parameters of a control pattern generator (CPG) [Liu *et al.* 2011], mapping the current sensor values onto the actuator values. As already discussed,

the parametric space is ill-suited to learn a preference-based policy return, as the expert's preferences only depend on the agent behavior and the agent behavior depends in an arbitrarily non-smooth way on the parametric policy representation. Another space, noted $\Phi(\mathcal{X})$ and referred to as feature space or behavioral space, thus needs be considered. Significant efforts have been made in RL to design a feature space suitable to capture the state-reward dependency (see e.g. [Hachiya & Sugiyama 2010]); in IRL in particular, the feature space encapsulates an extensive prior knowledge [Abbeel & Ng 2004]. In the considered swarm robotics framework however, comprehensive prior knowledge is not available, and the lack of generative model implies that massive data are not available either to construct an informed representation.

The proposed approach, inspired from [O'Regan & Noë 2001], uses the sensori-motor states defined in chapter 5: the data stream made of the robot sensor and actuator values, generated under the policy in the environment (possibly after unsupervised dimensionality reduction), is processed using ε-means [Duda & Hart 1973] to define sensori-motor clusters. To each such cluster, referred to as sensori-motor state (sms), is associated a feature. A trajectory is described as a vector, associating to each feature the fraction of the trajectory spent in this sensori-motor state. Letting D denote the number of sms, each trajectory $\mathbf{u}_x$ generated from π_x thus is represented as a unit vector in $[0,1]^D$ ($||\mathbf{u}_x||_1 = 1$). As noted, behavioral representation $\Phi(\mathcal{X})$ does not require any domain knowledge. Moreover, it is consistent despite the fact that the agent gradually discovers its sensori-motor space; new sms are added along the learning process as new policies are considered, but the value of new sms is consistently set to 0 for earlier trajectories.

6.3.2 Approximate Expected Utility of Selection

Let $\mathcal{U}_t = \{\mathbf{u}_0, \dots \mathbf{u}_{t-1}; (\mathbf{u}_{i_1} \prec \mathbf{u}_{i_2}), i = 1 \dots t\}$ denote the archive of all demonstrations seen by the expert up the t-th iteration, and the ranking constraints defined from the expert's preferences. With no loss of generality, the best demonstration in $\mathcal{U}_t$ is noted $\mathbf{u}_{t-1}$.

In PPL, the selection of the next policy to be demonstrated is based on the policy return $J_t(\pi_x) = \mathbb{E}_{u\sim\pi_x}[\langle \mathbf{w}_t, \mathbf{u}\rangle]$, with $\mathbf{w}_t$ solution of problem (5.1) (section 5.5). By construction however, $\mathbf{w}_t$ is learned from the trajectories in the archive, and hence does not reward the discovery of new sensori-motor states (as they are associated

a 0 weight by $\mathbf{w}_t$). Instead of considering the only max margin solution $\mathbf{w}_t$, the rationale here is to consider the version space $\mathbf{W}_t$ of all $\mathbf{w}$ that are consistent with the ranking constraints in the archive $\mathcal{U}_t$, along the same line than the expected utility of selection (EUS) criterion [Viappiani & Boutilier 2010] (section 6.2). Indeed, as noted in this latter work, the expert's preferences are likely to be noisy . Hence we cannot assume a realizable setting, i.e. the expert's preferences are likely to be noisy as noted by [Viappiani & Boutilier 2010]. However, the number of ranking constraints is always small relatively to the number D of sensori-motor states. One can therefore assume that the version space defined from $\mathcal{U}_t$ is not empty.

The EUS criterion cannot however be applied as such, since both policy return J_t and the version space refer to the behavioral space (space of trajectories) whereas the goal is to select an element on the parametric space; furthermore, both the behavioral and the parametric spaces are continuous and high-dimensional. An approximate expected utility of selection is thus defined on the behavioral and the parametric spaces, as follows. Let $\mathbf{u}_x$ denote a trajectory generated from policy π_x. The expected utility of selection of $\mathbf{u}_x$ can be defined as in [Viappiani & Boutilier 2010], as the expectation over the version space of the max between the utility of $\mathbf{u}_x$ and the utility of the previous best trajectory $\mathbf{u}_t$:

$$EUS(\mathbf{u}_x) = \mathbb{E}_{w \ in \ \mathbf{W}_t}[max(\langle \mathbf{w}, \mathbf{u}_x \rangle, \langle \mathbf{w}, \mathbf{u}_t \rangle)]$$

Specifically, trajectory $\mathbf{u}_x$ splits version space $\mathbf{W}_t$ into a challenger version space noted $\mathbf{W}_t^+$ (including all $\mathbf{w}$ with $\langle \mathbf{w}, \mathbf{u}_x \rangle > \langle \mathbf{w}, \mathbf{u}_t \rangle$)), and its complementary subspace $\mathbf{W}_t^-$. The expected utility of selection of $\mathbf{u}_x$ thus becomes:

$$EUS(\mathbf{u}_x) = \mathbb{E}_{w \ in \ \mathbf{W}_t^+}[\langle \mathbf{w}, \mathbf{u}_x \rangle] + \mathbb{E}_{w \ in \ \mathbf{W}_t^-}[\langle \mathbf{w}, \mathbf{u}_t \rangle]$$

The expected utility of selection of policy π_x is naturally defined as the expectation of EUS($\mathbf{u}_x$) over all trajectories $\mathbf{u}_x$ generated from policy π_x:

$$\begin{aligned} EUS(\pi_x) &= \mathbb{E}_{u_x \sim \pi_x}[EUS(\mathbf{u}_x)] \\ &= \mathbb{E}_{u_x \sim \pi_x}\left[\mathbb{E}_{w \ in \ \mathbf{W}_t^+}[\langle \mathbf{w}, \mathbf{u}_x \rangle] + \mathbb{E}_{w \ in \ \mathbf{W}_t^-}[\langle \mathbf{w}, \mathbf{u}_t \rangle]\right] \end{aligned} \tag{6.2}$$

Taking the expectation over all weight vectors $\mathbf{w}$ in W^+ or W^- is clearly intractable as $\mathbf{w}$ ranges in a high or medium-dimensional continuous space. Two approximations are therefore considered, defining the Approximate Expected Utility

of Selection criterion (AEUS). The first one consists in approximating the center of mass of a version space by the center of the largest ball in this version space, taking inspiration from the Bayes point machine [Herbrich *et al.* 2001]. The center of mass of $\mathbf{W}_t^+$ (respectively $\mathbf{W}_t^-$) is replaced by $\mathbf{w}^+$ (resp. $\mathbf{w}^-$) the solution of problem (5.1) (section 5.5) where constraint $\mathbf{u}_x > \mathbf{u}_t$ (resp. $\mathbf{u}_x < \mathbf{u}_t$) is added to the set of constraints in archive $\mathcal{U}_t$. As extensively discussed by [Herbrich *et al.* 2001], the SVM solution provides a good approximation of the Bayes point machine solution provided the dimensionality of the space is "not too high" (more about this in section 6.4.1).

The second approximation takes care of the fact that the two version spaces $\mathbf{W}_t^+$ and $\mathbf{W}_t^-$ are unlikely of equal probability (said otherwise, the splitting index might be arbitrarily low). In order to approximate $EUS(\mathbf{u}_x)$, one should thus further estimate the probability of $\mathbf{W}_t^+$ and $\mathbf{W}_t^-$. Along the same line, the inverse of the objective value $F(\mathbf{w}^+)$ maximized by $\mathbf{w}^+$ (problem (5.1), section 5.5) is used to estimate the probability of $\mathbf{W}_t^+$: the higher the objective value, the smaller the margin and the probability of $\mathbf{W}_t^+$. Likewise, the inverse of the objective value $F(\mathbf{w}^-)$ maximized by $\mathbf{w}^-$ is used to estimate the probability of $\mathbf{W}_t^-$.

Finally, the approximate expected utility of selection of a policy π_x is defined as:

$$\mathrm{AEUS}_t(\pi_x) = \mathbb{E}_{u\sim\pi_x}\left[\frac{1}{F(\mathbf{w}^+)}\langle \mathbf{w}^+, \mathbf{u}_x\rangle + \frac{1}{F(\mathbf{w}^-)}\langle \mathbf{w}^-, \mathbf{u}_t\rangle\right] \tag{6.3}$$

6.3.3 Discussion

The fact that April considers two policy representations, the parametric and the behavioral or feature space, aims at addressing the expressiveness/tractability dilemma. On the one hand, a high dimensional continuous search space is required in order to express competent policies for complex tasks. But such high-dimensional search space makes it difficult to learn a preference-based policy return from only a moderate number of rankings, i.e., keeping the expert's burden within reasonable limits. On the other hand, the behavioral space does enable to learn a preference-based policy return from the little available evidence in the archive (note that the dimension of the behavioral space is controlled by April) although the behavioral description, because of its simplicity, might be insufficient to describe a flexible

policy.

The price to pay for dealing with both search spaces lies in the two approximations needed to transform the expected utility of selection (Eq. (6.2)) into a tractable criterion (Eq. (6.3)), replacing the two centers of mass of version spaces $\mathbf{W}_t^+$ and $\mathbf{W}_t^-$ (i.e. the solutions of the Bayes point machine [Herbrich *et al.* 2001]) with the solutions of the associated support vector machine problems, and estimating the probability of these version spaces from the objective values of the associated SVM problems.

6.4 Experimental Results

This section presents the experimental setting that has been used to validate April. Firstly, the performance of the approximate expected utility of selection (AEUS) criterion is assessed in an artificial setting. Secondly, the performance of April is assessed comparatively to that of Inverse Reinforcement Learning [Abbeel & Ng 2004] on two Reinforcement Learning benchmarks, the well-known mountain car problem, and the cancer treatment problem [Zhao *et al.* 2009]. Lastly, April is investigated in-situ to achieve the docking task involved in the Symbrion project.

6.4.1 Validation of the Approximate Expected Utility of Selection

The artificial active ranking problem used to investigate AEUS robustness is inspired from the active learning frame studied by [Dasgupta 2005], varying the dimension d of the search space in $\{10, 20, 50, 100\}$ (Fig. 6.1).

In each run, a target utility function is set as a vector $\mathbf{w}^*$ uniformly selected in the d-dimensional L_2 unit sphere. A fixed sample $S = \{\mathbf{u}_1, \ldots \mathbf{u}_{1000}\}$ of 1,000 points is uniformly generated in the d-dimensional L_1 unit sphere, so as to take into account for the fact that the behavioral representation of a trajectory has L_1 norm 1 by construction (section 6.3.1).

At iteration t, the sample $\mathbf{u}$ with best AEUS is selected in S; the expert ranks it comparatively to the previous best solution $\mathbf{u}_t$, yielding a new ranking constraint (e.g. $\mathbf{u} < \mathbf{u}_t$), and the process is iterated. The AEUS performance at iteration t is computed as the scalar product of $\mathbf{u}_t$ and $\mathbf{w}^*$.

AEUS is compared to an empirical estimate of the expected utility of selection

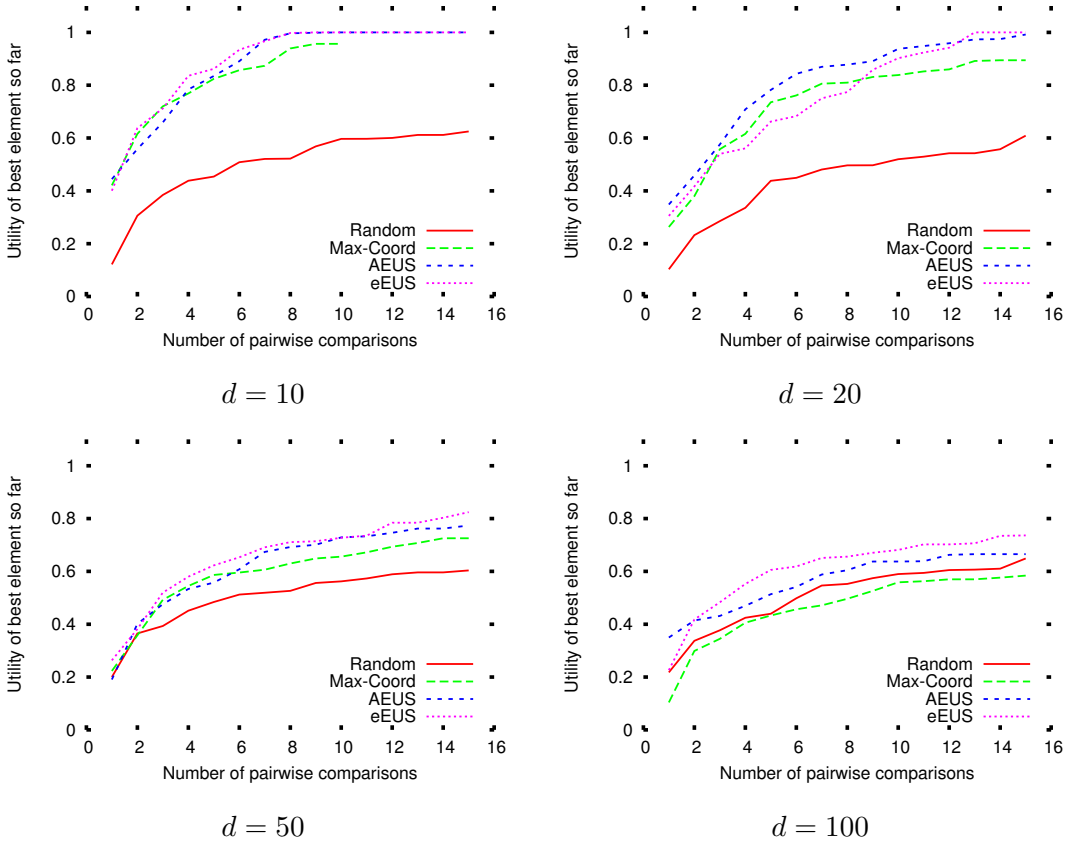


Figure 6.1: Performance of AEUS comparatively to baselines (see text) *vs* number of pairwise comparisons, depending on the dimension d of the search space (results averaged over 101 runs).

(baseline *greedy EUS*). In this *greedy EUS*, the current best sample $\mathbf{u}$ in S is selected from Eq. (6.2), computed from 10,000 points $\mathbf{w}$ selected in the version spaces $\mathbf{W}_t^+$ and $\mathbf{W}_t^-$ and $\mathbf{u}_t$ is built as above. Two other baselines are considered: Random, where sample $\mathbf{u}$ is uniformly selected in S, and Max-Coord, selecting with no replacement $\mathbf{u}$ in S with maximal L_∞ norm. The Max-Coord baseline was found to perform surprisingly well in the early active ranking stages, especially for small dimensions d.

The empirical results reported in Fig. 6.1 show that AEUS is a good active ranking criterion, yielding a good approximation of EUS. The approximation degrades gracefully as dimension d increases: As noted by [Herbrich *et al.* 2001], the center of the largest ball in a convex set yields a lesser good approximation of the center of

mass thereof as dimension d increases. In the meanwhile, the approximation of the center of mass degrades too as a fixed number of 10,000 points are used to estimate EUS regardless of dimension d.

As noted, and unexpectedly, the Max-Coord baseline performs well in the early stages of the active ranking process, all the more so as d is small. It is thereafter dominated by all other approaches (except the random baseline for low-dimensional spaces). The comparison with the Max-Coord baseline was meant to show that APRIL does not simply explore the extreme points in the simplex.

6.4.2 Validation of APRIL

The main goal of the next experiments is to comparatively assess APRIL with respect to Inverse Reinforcement Learning (IRL), more precisely, the projection IRL algorithm proposed in [Abbeel & Ng 2004] and described in section 3.4.2 (and denoted projection-AL on the graphs). Both IRL and APRIL extract the sought policy through an iterative two-step processes. The difference is that IRL is initially provided with an expert trajectory, whereas APRIL receives one bit of information from the expert on each trajectory it demonstrates (its ranking w.r.t. the previous best trajectory). APRIL performance is thus measured in terms of "expert sample complexity", i.e. the number of bits of information needed to catch up compared to IRL. APRIL is also assessed and compared to the direct policy search method using the black-box CMA-ES stochastic optimization algorithm, with its robust default parameters [Hansen & Ostermeier 2001].

All three IRL, APRIL and CMA-ES algorithms are empirically evaluated on two Reinforcement Learning benchmark problems, the well-known mountain car problem and the cancer treatment problem first introduced by [Zhao *et al.* 2009]. None of these problems has a reward function.

Policies are implemented as a multiplayer Perceptron with 1-hidden layer, 2 input nodes, and 1 output node, the acceleration for the mountain car problem and the dosage for the cancer treatment problem. The hidden layer contains 9 neurons for the mountain car (respectively 99 nodes for the cancer problem), thus the dimension of the parametric search space is 37 (resp. 397).

RankSVM is used as learning-to-rank algorithm [Joachims 2006], with linear

kernel and $C = 100$[2]. All reported results are averaged out of 101 independent runs.

6.4.2.1 The Cancer Treatment Problem

In the cancer treatment problem, a stochastic transition function is provided, yielding the next patient state from its current state (tumor size s_t and toxicity level t_t) and the selected action (drug dosage level a_t):

$$\begin{aligned} s_{t+1} &= s_t + 0.15\max(t_t, t_0) - 1.2(a_t - 0.5) \times 1(s_t > 0) + \varepsilon \\ t_{t+1} &= t_t + 0.1\max(s_t, s_0) + 1.2(a_t - 0.5) + \varepsilon \end{aligned}$$

Further, the transition model involves a stochastic death mechanism (modelling the possible patient death by means of a hazard rate model). The same setting as in [Cheng *et al.* 2011] is considered with three differences. Firstly, we considered a continuous action space (the dosage level is a real value in $[0, 1]$), whereas the action space contains 4 discrete actions in [Cheng *et al.* 2011]. Secondly the time horizon is set to 12 instead of 6. Thirdly, a Gaussian noise ε with mean 0 and standard deviation σ (ranging in 0, 0.05, 0.1, 0.2) is introduced in the transition model. The AEUS of the candidate policies is computed as the empirical AEUS average over 11 trajectories (Eq. 6.3).

The initial state is set to 1.3 tumor size and 0 toxicity. For the sake of reproducibility the expert preferences are emulated by favoring the trajectory with minimal sum of the tumor size and toxicity level at the end of the 12-months treatment.

The average performance (sum of tumor size and toxicity level) of the best policy found in each iteration is reported in Fig. 6.2 for the noiseless case, and in Fig. 6.3 for the 3 different levels of noise. It turns out that the cancer treatment problem is an easy problem for IRL, that finds the optimal policy in the second iteration. A tentative interpretation for this fact is that the target behavior extensively visits the state with zero toxicity and zero tumor size; the learned $\mathbf{w}$ thus associates a maximal weight to this state. In subsequent iterations, IRL thus favors policies reaching this state as soon as possible. APRIL catches up after 15 iterations, whereas CMA-ES

[2]The value of C typically depends on the dimension of the behavioral representation space, and is determined by off-line preliminary experiments.

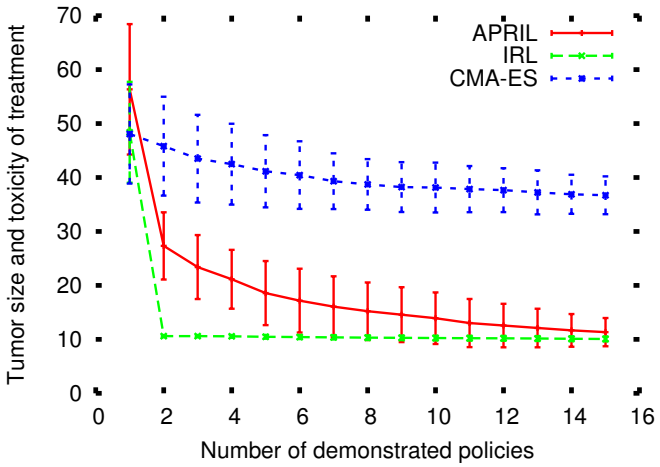


Figure 6.2: Comparative performances of APRIL IRL and CMA-ES on the cancer treatment problem, plotting the cumulative toxicity level and tumor size after 12-months treatment, versus the number of trajectories demonstrated policies to the expert. Results are averaged over 101 runs

remains consistently far from reaching the target policy in the considered number of iterations when there is no noise, and yields even worse results (not visible on the plot) for higher noise levels.

6.4.2.2 The Mountain Car

The same setting than in [Whiteson *et al.* 2010a] is considered. The car state is described from its position and speed, initially set to position -0.5 with speed 0. The action space is set to $\{-1, 0, 1\}$, setting the car acceleration. For the sake of reproducibility the expert preferences are emulated by favoring the trajectory which soonest reaches the top of the mountain, or is closest to the top of the mountain at some point during the 1000 time-step trajectory.

Interestingly, the mountain car problem appears to be more difficult for IRL than the cancer treatment problem (Fig.6.4), which is blamed on the lack of expert features. As the trajectory is stopped when reaching the top of the mountain and this state does not appear in the trajectory description, the target reward would have negative weights on every (other) sms feature. IRL thus finds an optimal policy after 7 iterations on average. As for the cancer treatment problem, APRIL

catches up after 15 iterations, while the stochastic optimization never catches up in the considered number of iterations.

6.5 APRIL: Summary and Discussion

The goal of April, decreasing the number of comparison requests to the expert needed to yield a satisfactory policy, was demonstrated feasible on the considered benchmark problems. More precisely, the lesson learned from the experimental validation of April is that a very limited external information might be sufficient to enable reinforcement learning: while mainstream Reinforcement Learning requires a numerical reward to be associated to each state, while Inverse Reinforcement Learning [Abbeel & Ng 2004, Kolter *et al.* 2007] requires the expert to demonstrate a sufficiently good policy, April requires a couple dozen bits of information (this trajectory improves/does not improve on the former best one) to reach state of the art results.

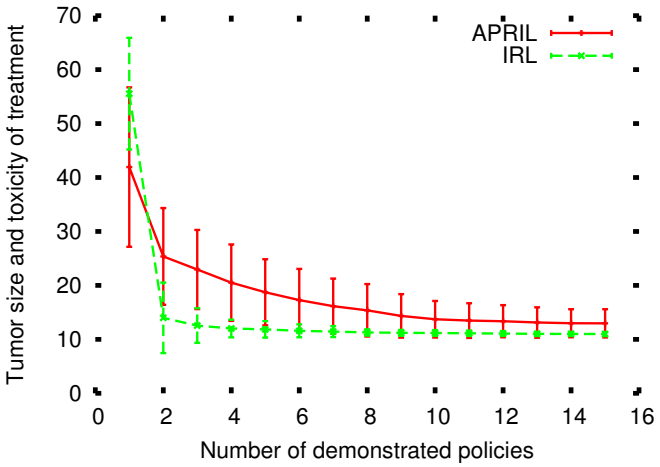
The proposed active ranking mechanism, inspired from preference elicitation [Viappiani 2012], is an approximation of the Bayesian expected utility of selection criterion; on the positive side, AEUS is tractable in high-dimensional continuous search spaces; on the negative side, it lacks the approximate optimality guarantees of EUS.

A first research perspective concerns the theoretical analysis of the April algorithm, specifically its convergence and robustness w.r.t. the ranking noise, and the approximation quality of the AEUS criterion. In particular, the computational effort of AEUS could be reduced with no performance loss by using Bernstein races to decrease the number of empirical estimates (considered trajectories in Eq. 6.3) and confidently discard unpromising solutions as done un [Heidrich-Meisner & Igel 2009].

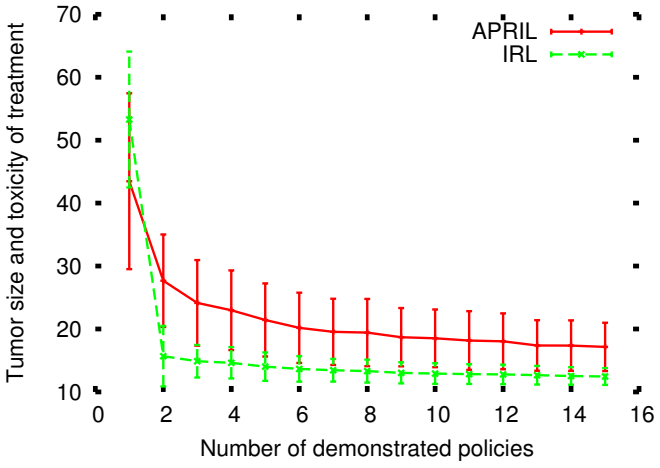
Another perspective, also mentioned in chapter 5 is concerned with hybrid policies, combining the (NN-based) parametric policy and the model of the expert's preferences. The idea behind such hybrid policies is to provide the agent with both reactive and deliberative skills: while the action selection is achieved by the parametric policy by default, the expert's preferences might be exploited to reconsider these actions in some (discretized) sensori-motor states.

A main perspective in line with our motivating application within the Symbrion Integrated Project, is to apply APRIL to the control of individual robots, for which designing good reward functions is notoriously difficult, and to swarm robotics, for which the design of reward functions raises even more difficulties.

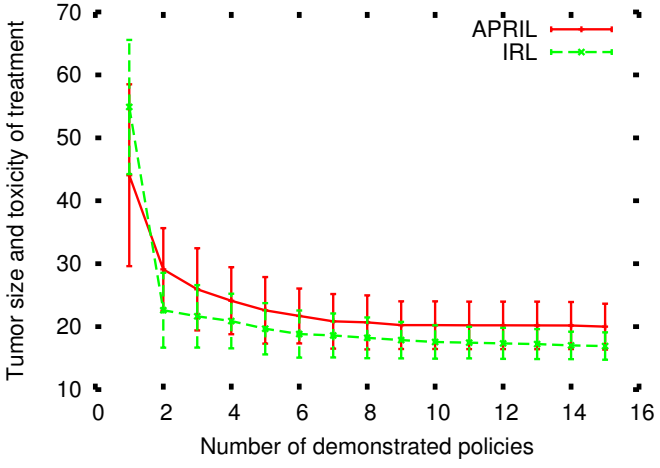
However our priority, addressed in next chapter, is to remedy the main APRIL limitation: its sensitivity with respect to the expert's preference mistakes.



$\sigma_{noise} = 0.05$



$\sigma_{noise} = 0.1$



$\sigma_{noise} = 0.2$

Figure 6.3: The cancer treatment problem: Average performance (cumulative toxicity level and tumor size after 12-months treatment) of APRIL, IRL and CMA-ES versus the number of trajectories demonstrated to the expert, for noise levels .05, .1 and .2. Results are averaged over 101 runs.

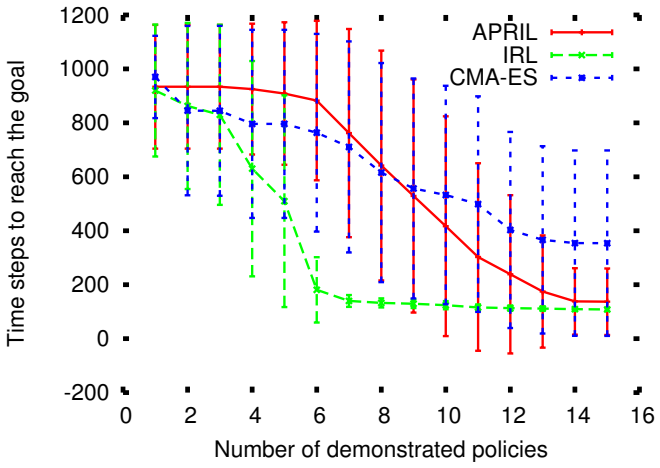


Figure 6.4: The mountain car problem: Average performance (number of time steps needed to reach the top of the mountain) of APRIL, IRL and CMA-ES versus the number of trajectories demonstrated to the expert. Results are averaged over 101 runs.

Chapter 7

Programming by Feedback

Foreword

This chapter introduces the third contribution of the PhD, the *Programming by Feedback* (PF) scheme first presented in [Akrour *et al.* 2014].

As detailed in previous chapter 6, April efficiently reduces the number of preference queries to the user that are needed to approximate her preference model a.k.a. rank-based policy estimate. In counterpart for this reduced number of queries, the impact of user's mistakes is much more detrimental to the overall process.

The problem is that (as could have been expected) every *in-situ* experiment confirms that the expert user (ourselves) is bound to make preference mistakes.

In the beginning – the bootstrapping phase – the input provided by the robot is hardly relevant and the user is easily driven into considering that all active computer suggestions are equally irrelevant. In the ending phase inversely, very relevant suggestions are provided and the user might easily make judgment errors; according to e.g. the Plackett-Luce model [Luce 1959], the probability of misranking two solutions exponentially increases as their difference in quality decreases. In both phases, the user might be deterred by getting insufficient returns, inducing him to behave in an inconsistent way, and getting even more inconsistent returns as a result.

Mistakes in the ending phase are easier to stand as the current solution already is satisfactory. Quite the contrary, mistakes in the beginning phase result in frustrating interactions between the user and the robot, where no behavior progress is seen for long periods of time – for ever.

The robot must thus be prepared to cope with user's mistakes; more generally, it should be endowed with a model of the user's competence. The interaction between the user and the robot is thus increasingly viewed as a cooperation between intelligent partners. The increased autonomy of the robot, thereafter referred to as active

computer, is materialized by its exploiting the user competence model in order to revise the user preference model.

Most generally, it has long been suggested that the cooperation between intelligent partners requires each partner to have a model of the other one [Lörincz *et al.* 2007].

As a first step toward such an interaction between intelligent partners, the *Programming by Feedback* scheme is presented and validated theoretically and empirically in this chapter.

7.1 Motivations

The general claim behind the PF scheme is the following: there is emerging evidence that the art of programming could be revisited in the light of the current state of the art in machine learning and optimization.

The royal road for software engineering did and still does rely on formal specifications, verification and theorem proving, at the core of software sciences for over three decades[1]. As of the end 80s however, particularly for applications concerned with pattern recognition, image processing or natural language processing, the formal specification-based approach was shown to be ineffective. Learning from examples and ML-based approaches were soon recognized to be efficient and relevant alternatives [Y. LeCun et al. 1989, Poggio & Girosi 1990]; cheque reading softwares appeared and their generalized use by some banks in US and Canada was worth a certification.

In the last decade, in domains concerned with combinatorial problems such as Constraint Programming, Stochastic Optimization and Machine Learning itself, a new trend dubbed *Programming by Optimization* (PbO) appeared [Hoos 2012]. PbO advocates algorithm portfolios endowed with a control layer such that *determining what works best in a given use context [could be] performed automatically, substituting human labor by computation* . In the domain of e.g. image classification, it is suggested that the *state of the art can be improved by configuring existing techniques better rather than inventing new learning paradigms* [Snoek *et al.* 2012]. All

[1] As witnessed by the 2014 *Turing Award* awarded to Leslie Lamport, or the 2013 *ACM Software System Award* given to the formal proof system COQ `http://coq.inria.fr/`

the above approaches rely on learning surrogate models, predicting the performance yielded by a portfolio algorithm or an algorithm configuration depending on the context, and using sequential model-based optimization [Lizotte 2008] to determine the best configuration for the particular problem instance.

Independently, another trend specifically aimed at information retrieval [Viappiani & Boutilier 2010, Yue & Joachims 2009, Shivaswamy & Joachims 2012] and reinforcement learning [Wilson *et al.* 2012, Akrour *et al.* 2012, Knox *et al.* 2013, Jain *et al.* 2013], is investigated in the ML field. This trend, generally referred to as *Interactive Learning for Optimization* (ILO, chapter 4), implements an iterative 2-step process, with the active computer providing some input to the user, and the user providing a response (preference judgment, modification or suggestion to the active computer) until getting a satisfactory solution. In all of the above, the user and the active computer cooperate and achieve some division of labor between the exploration and the exploitation parts of the search process.

The PF scheme aims to generalize the ILO interaction between the active computer and the user in a principled and robust way, where the active computer searches for the user's utility function and accounts for her inconsistencies. Note that the active computer cannot make any difference between the user's competence (her preferences are consistent with her goal) and the user's consistency (her preferences are consistent); at any rate, accounting for the user's possibly limited competence enables the active computer to better exploit the user's instant feedback.

The robustness w.r.t. user's inconsistencies also is a requisite to scale up to complex tasks, along two lines. Firstly, deploying the PF scheme on the grand scale requires that the active computer faces several users, collectively shaping the active computer behavior; in such a collective setting, individual preference models are likely to be inconsistent with each other. Further research will be devoted to considering ensemble of preference models in PF; but as of now, a single aggregated preference model is considered.

Secondly, the evolution of the user's preference model can be thought of Continuation Method [Wasserstrom 1973], presenting the robot/active computer with a sequence of goals with increasing difficulty. As an example, the user might want to teach first the robot to walk, before running: after some skills have been acquired

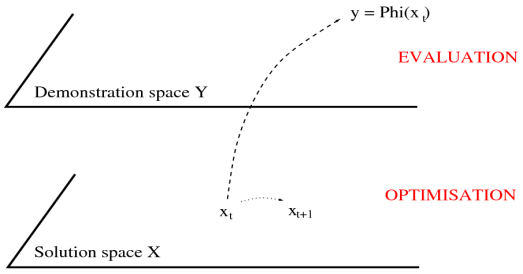


Figure 7.1: PF: Solution and demonstration search spaces

(walking), the acquisition of some other skills (running) will be facilitated. In such a context, the observed inconsistencies of the user are actually motivated by pedagogical concerns: the current user's preferences depend on the skills already acquired by the active computer, and new preferences will emerge along the active computer educational process (analogously, parents would praise and encourage some behaviors for a young child, while the same behaviors are taken for granted for older children).

7.2 Overview of the Programming by Feedback scheme

As in PPL (Chapter 5) and APRIL (Chapter 6), two search spaces are distinguished (Fig. 7.1): the search space $\mathcal{X}$ (analogous to the policy space in the Reinforcement Learning context) is referred to, in this chapter, as *solution space*[2]. Similarly, the evaluation space $\mathcal{Y}$ (analogous to the trajectory space in Reinforcement Learning) is referred to as *demonstration space*. The user only sees a realization $\mathbf{y} = \Phi(\mathbf{x})$ of a program $\mathbf{x}$. As already discussed earlier, in contrast with the direct policy learning setting [Whiteson *et al.* 2010a, Deisenroth *et al.* 2013], no continuity assumption regarding the stochastic mapping Φ from the solution to the demonstration space can be assumed (clearly, small modifications of a program can result in significantly different behaviors).

The true (unknown) user's utility function U is a linear function on demonstration space $\mathcal{Y}$, parameterized as $\mathbf{w}^*$:

$$U(\mathbf{y}) = \langle \mathbf{w}^*, \mathbf{y} \rangle$$

Like ILO, PF is an iterative two-step process. At time step t, the active computer

[2] so as to make it clear that the context is here more general than that of RL.

Algorithm 4 Programming by Feedback

Input: prior θ_0 on the utility function space $\mathbf{W}$

Init:

Generate $\mathbf{x}_0$ in $\mathcal{X}$

Demonstrate $\mathbf{y}_0 = \Phi(x_0)$; $\mathbf{y}^*_0 = \mathbf{y}_0$

Archive $\mathcal{U}_t = \{\mathbf{y}_0\}$

repeat

Optimize: Generate $\mathbf{x}_{t+1}$ from θ_t (Section 7.5)

Demonstrate $\mathbf{y}_{t+1} = \Phi(x_{t+1})$

Receive the user's feedback $1_{\mathbf{y}_{t+1} \succ \mathbf{y}^*_t}$

Update $\mathcal{U}_t = \mathcal{U}_t \cup \{\mathbf{y}_{t+1}, 1_{\mathbf{y}_{t+1} \succ \mathbf{y}^*_t}\}$

Learn posterior θ_{t+1} from $\mathcal{U}_t$ (Section 7.3)

until User stops

Return: Generate $\mathbf{x}$ from θ

i) selects a solution $\mathbf{x}_t$ and demonstrates $\mathbf{y}_t = \Phi(\mathbf{x}_t)$;

ii) receives the user's feedback, ranking $\mathbf{y}_t$ comparatively to the previous best demonstration[3] noted $\mathbf{y}^*_t$;

iii) updates its model of the user's utility function.

7.3 Learning the Utility Function

Let $\mathbf{W}$ denote the space of normalized linear functions on $\mathcal{Y}$. Given a uniform prior on $\mathbf{W}$, the active computer learns by computing its posterior distribution θ_t from evidence $\mathcal{U}_t = \{\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_t; (\mathbf{y}_{i_1} \succ \mathbf{y}_{i_2}), i = 1 \ldots t\}$ including all demonstrations and user's preferences up to the t-th time step. Given θ_t, the estimated utility $U^{(ph)}_t(\theta_t, \mathbf{y})$ of a demonstration $\mathbf{y}$ is defined as:

$$U^{(ph)}_t(\theta_t, \mathbf{y}) = \mathbb{E}_{\theta_t}[\langle \mathbf{w}, \mathbf{y} \rangle] \tag{7.1}$$

[3] Two settings are considered, referred to as oblivious and non-oblivious. In the oblivious setting, the user is presented with two demonstrations in each time step, and ranks them comparatively to each other. In the non-oblivious setting, the user is presented with a single demonstration in each iteration, which he ranks comparatively to (his memory of) the best former demonstration. The non-oblivious setting has been considered in the experiments as it is less demanding for the user.

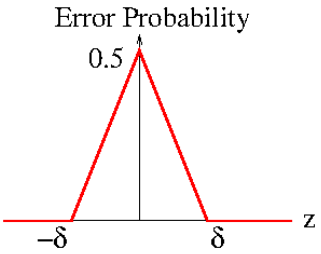


Figure 7.2: The ridge noise model: probability of ranking error depending on the true preference margin.

By construction, the utility function on the demonstration space induces a utility function on the solution space noted $U_t^{(g)}(\theta_t, \mathbf{x})$, defined as:

$$U_t^{(g)}(\theta_t, \mathbf{x}) = \mathbb{E}_{\mathbf{y} \sim \Phi(\mathbf{x})}[U_t^{(ph)}(\theta_t, \mathbf{y})] \tag{7.2}$$

In the following, by simplicity, and when no confusion is to fear from the context, $U_t^{(ph)}$ and $U_t^{(g)}$ will be noted U_t.

As already said, the utility function must account for the preference noise. Given two demonstrations $\mathbf{y}$ and $\mathbf{y}'$, and $\mathbf{w}^*$ denoting the true (unknown) utility of the user, the observed user's preference is usually modelled as a perturbation of his true preference $\langle \mathbf{w}^*, (\mathbf{y} - \mathbf{y}') \rangle$, considering a Gaussian perturbation [Chu & Ghahramani 2005b, Wilson *et al.* 2012] or following the Luce-Shepard model [Luce 1959, Shepard 1957]. In both cases, the noise magnitude is calibrated using one hyper-parameter, respectively the standard deviation of the Gaussian perturbation or the temperature of the Luce-Shepard rule.

A ridge noise model is considered in PF, calibrated from a hyper-parameter δ ($\delta \in \mathbb{R}^+$). Given the preference margin $\mathrm{z} = \langle \mathbf{w}^*, (\mathbf{y} - \mathbf{y}') \rangle$, the observed preference is bound to coincide with the true preference if $|\mathrm{z}| > \delta$; otherwise the observed preference $\mathbf{y} \succ \mathbf{y}'$ is set to 1 with probability $\frac{\delta + \mathrm{z}}{2\delta}$.

$$P(\mathbf{y} \succ \mathbf{y}' \mid \mathbf{w}^*, \delta) = \begin{cases} 0 & \text{if } \mathrm{z} < -\delta \\ 1 & \text{if } \mathrm{z} > \delta \\ \frac{\delta + \mathrm{z}}{2\delta} & \text{otherwise} \end{cases} \tag{7.3}$$

The user's competence is thus modelled through parameter δ. A first option

is to consider that the user's competence is characterized by a hidden but fixed δ^*. A second option is to consider that δ might vary along time. The drawback of the former option is that one abnormally large mistake can prevent the active computer from identifying an otherwise consistent utility function. Inversely, the latter option being more cautious could slow down the identification of the user's utility function. This latter option is however more appropriate to accommodate the cases where the task (or the user's understanding thereof, or his preferences) might change over time. Not only can the user change his mind; in the general case, one PF run might involve several users, responsible for the successive feedbacks.

Hypothesis 1. For the sake of robustness, the noise parameter δ is assumed to be uniform on $[0, M]$.

Under these assumptions, the posterior distribution θ_t on the utility function space can be expressed in closed form:

Lemma 1 Given evidence $\mathcal{U}_t$, the posterior distribution on the utility function space reads:

$$\begin{array}{rcl} \theta_t(\mathbf{w}) & \propto & \prod_{i=1,t} P(\mathbf{y}_{i_1} \succ \mathbf{y}_{i_2} \mid \mathbf{w}) \\ & = & \prod_{i=1,t} \left(\frac{1}{2} + \frac{\mathrm{z}_i(\mathbf{w})}{2M} \left(1 + \log \frac{M}{|\mathrm{z}_i(\mathbf{w})|} \right) \right) \end{array} \tag{7.4}$$

where $\mathrm{z}_i(\mathbf{w})$ is set to $\langle \mathbf{w}, (\mathbf{y}_{i_1} - \mathbf{y}_{i_2}) \rangle$ if it is not greater (resp. lower) than M (resp. $-M$) in which case it takes the value M (resp. $-M$)[4].

***Proof**: For a given $\mathcal{U}_t$ and $\mathbf{w}$, for any y, y' with $z = \langle \mathbf{w}, (\mathbf{y} - \mathbf{y}') \rangle$, and using the ridge model (7.3) with δ uniformly distributed, it comes*

$$\begin{array}{rcl} \int_0^M P(\mathbf{y} \succ \mathbf{y}' \mid \mathbf{w}, \delta) d\delta & = & \int_0^{|z|} P(\mathbf{y} \succ \mathbf{y}' \mid \mathbf{w}, \delta) d\delta + \int_{|z|}^M P(\mathbf{y} \succ \mathbf{y}' \mid \mathbf{w}, \delta) d\delta \\ & = & \int_0^{|z|} 1_{z>0} d\delta + \int_{|z|}^M (\frac{1}{2} + \frac{z}{2\delta}) d\delta \\ & = & |z| \cdot 1_{z>0} + \frac{1}{2}(M - |z|) + \frac{z}{2}(\log(M) - \log(|z|)) \\ & = & \frac{1}{2}(M + z) + \frac{z}{2} \log(\frac{M}{|z|}) \end{array}$$

[4] We use the convention that $x \log(x) = 0$ for $x = 0$.

hence the result, after normalizing the total mass from M to 1. ■

Comparatively to the Gaussian or the Luce-Shepard noise models, the ridge noise model enables a straightforward interpretation and calibration: the upper bound M is homogeneous to a utility, as there is no noise when the preference margin is higher than[5] M.

7.4 Optimization Criterion in the Demonstration Space

Conditionally to $\mathcal{U}_t$, the expected utility of selection of a pair of demonstrations $\mathbf{y}, \mathbf{y}'$ to be demonstrated to the user is defined as follows, where θ_t^+ (respectively θ_t^-) denotes the posterior distribution on $\mathcal{Y}$ based on evidence $\mathcal{U}_t \cup \{(\mathbf{y} \succ \mathbf{y}')\}$ (resp $\mathcal{U}_t \cup \{(\mathbf{y} \prec \mathbf{y}')\}$):

$$\begin{aligned} EUS(\mathbf{y},\mathbf{y}') &= \mathbb{E}_{\theta_t}[\langle \mathbf{w}, \mathbf{y}-\mathbf{y}'\rangle > 0] \,.\, U(\theta_t^+, \mathbf{y}) \\ &+\mathbb{E}_{\theta_t}[\langle \mathbf{w}, \mathbf{y}-\mathbf{y}'\rangle < 0] \,.\, U(\theta_t^-, \mathbf{y}') \end{aligned} \tag{7.5}$$

[Viappiani & Boutilier 2010] advocate the use of the expected posterior utility, defined as the expected utility of the best behavior $\mathbf{y}^*$ (respectively $\mathbf{y}'^*$) conditionally to the user preferring $\mathbf{y}$ to $\mathbf{y}'$ (resp. $\mathbf{y}'$ to $\mathbf{y}$):

$$\begin{aligned} EPU(\mathbf{y},\mathbf{y}') &= \mathbb{E}_{\theta_t}[\langle \mathbf{w}, \mathbf{y}-\mathbf{y}'\rangle > 0] \,.\, max_{\mathbf{y}} U(\theta^+, \mathbf{y}) \\ &+\mathbb{E}_{\theta_t}[\langle \mathbf{w}, \mathbf{y}-\mathbf{y}'\rangle < 0] \,.\, max_{\mathbf{y}} U(\theta^-, \mathbf{y}) \\ &= \mathbb{E}_{\theta_t}[\langle \mathbf{w}, \mathbf{y}-\mathbf{y}'\rangle > 0] \,.\, U(\theta^+, \mathbf{y}^*) \\ &+\mathbb{E}_{\theta_t}[\langle \mathbf{w}, \mathbf{y}-\mathbf{y}'\rangle < 0] \,.\, U(\theta^-, \mathbf{y}'^*) \end{aligned} \tag{7.6}$$

By construction $EUS(\mathbf{y},\mathbf{y}') \leq EPU(\mathbf{y},\mathbf{y}')$; and $EPU(\mathbf{y},\mathbf{y}') \leq EUS(\mathbf{y}^*,\mathbf{y}'^*)$ [Viappiani & Boutilier 2010]. The optimization of the highly computationally demanding EPU criterion (since computing EPU(**y**,**y**') requires solving two optimization problems) can therefore be replaced with the optimization of the EUS criterion:

$$\max\{EUS(\mathbf{y},\mathbf{y}')\} = \max\{EPU(\mathbf{y},\mathbf{y}')\}$$

The proposed noise model has an impact on both the EUS and the EPU criteria. Let us first bound its impact on the EUS criterion. In the following, the *noiseless,*

[5] Setting M to $\alpha\sqrt{2}U_{max}$ where U_{max} stands for the maximum utility implies that two demonstrations $\mathbf{y}$ and $\mathbf{y}'$ cannot be mistakingly ordered if their margin is more than α percent of U_{max}.

NL (respectively, *noisy, N*) subscript indicates that $\mathbf{y}$ is preferred to $\mathbf{y}'$ for every utility $\mathbf{w}$ such that $\langle \mathbf{w}, \mathbf{y} \rangle > \langle \mathbf{w}, \mathbf{y}' \rangle$ (resp. with probability defined by Eq. 7.3). Note that in both cases, the expectation is taken w.r.t. the posterior distribution on the $\mathbf{W}$ utility space given by Eq. 7.4, thus accounting for the user's noise. Then,

Lemma 2 For any pair of demonstrations $\mathbf{y}, \mathbf{y}'$, and for any archive $\mathcal{U}_t$, the difference between their expected utility of selection under the noisy and noiseless models is bounded as follows:

$$EUS^{NL}(\mathbf{y}, \mathbf{y}') - L \leq EUS^{N}(\mathbf{y}, \mathbf{y}') \leq EUS^{NL}(\mathbf{y}, \mathbf{y}')$$

With $L = \frac{M}{2\lambda}(1 - \frac{1+ln\lambda}{\lambda})$, $\lambda = e^{-\frac{1}{2} - W_{-1}(-\frac{1}{2}e^{-\frac{1}{2}})}$ and W_{-1} is the lower branch of the Lambert W function. Approximatively, $L \approx \dfrac{M}{19.6433}$.

***Proof**: from Lemma 1 and [Viappiani & Boutilier 2010].*

Note that the lower bound is tight; it corresponds to the adverse case where the preference margin is intermediate: sufficiently low to entail a high chance of error and sufficiently high to entail a significant loss of utility. The impact of the noise model on the EPU criterion is finally bounded as follows. Let $EPU_t^{*,N}$ (respectively $EUS_t^{*,N}$) denote the optimum of the noisy EPU (resp. EUS) criterion conditionally to $\mathcal{U}_t$; let likewise $EUS_t^{*,NL}$ denote the optimum of the noiseless EUS. Then:

Proposition 1

$$EUS_t^{*,NL} - L \leq EPU_t^{*,N} \leq EUS_t^{*,NL} + L$$

***Proof**: One first shows using Lemma 1 and adapting the proof sketch in [Viappiani & Boutilier 2010], that*

$$EUS_t^{*,N} \leq EPU_t^{*,N} \leq EUS_t^{*,N} + L$$

The result then follows from Lemma 2. ■

PF thus proceeds by optimizing the noiseless expected utility of selection, with a bounded loss of performance compared to the noisy expected posterior utility.

This result will allow us to tackle the optimization of the noiseless expected utility of selection, with bounded loss of performance compared to the noisy expected posterior utility.

7.5 Optimization in the Solution Space

At time step t, PF tackles the optimization problem defined on $\mathcal{X}$ as:

$$\text{Find argmax } \mathcal{F}_t(\mathbf{x}) = \mathbb{E}_\Phi[EUS_t^{NL}(\Phi(\mathbf{x}), \mathbf{y}_t^*)] \tag{7.7}$$

with EUS_t^{NL} the expected utility of selection under θ_t (where the user's noise is taken into account when updating the posterior θ_t) and $\mathbf{y}_t^*$ the best-so-far demonstration. As discussed in Section 7.2, this non-oblivious setting is meant to decrease the computational cost and the user's cognitive fatigue. While the performance loss compared to the oblivious setting is limited[6], its bounding is left for further work (chapter8).

A main issue is that Eq. 7.7 defines a noisy black-box optimization problem[7]: On the one hand, $\mathcal{F}_t(\mathbf{x})$ is defined as an expectation; on the other hand, as Φ is not expressed in closed form in the general case, the expectation needs be approximated by an empirical average over demonstrations drawn from $\Phi(\mathbf{x})$.

Two steps are taken to handle this optimization problem. Firstly, $\mathcal{F}_t$ is replaced by a lower bound thereof. The expected utility of selection of the average demonstration $\mathbb{E}[\Phi(\mathbf{x})]$, noted $\bar{\mathbf{y}}$ in the following, is a lower bound of the utility of selection expectation on $\Phi(\mathbf{x})$, due to the convexity of the max operator on $\mathbf{W}$ and the Jensen inequality:

$$\mathbb{E}_\Phi[EUS^{NL}(\Phi(\mathbf{x}), \mathbf{y}_t^*)] \geq EUS^{NL}(\bar{\mathbf{y}}, \mathbf{y}_t^*)$$

Secondly, a sequence of solutions with increasing $EUS_t^{NL}(\bar{\mathbf{y}}, \mathbf{y}_t^*)$ is built as follows. Let $\mathbf{x}_1$ be the solution with optimal expected utility according to some $\mathbf{w}_0$ drawn according to θ_t:

$$\mathbf{x}_1 = \text{ argmax } \{\mathcal{G}_1(\mathbf{x}) = \langle \mathbf{w}_0, \bar{\mathbf{y}} \rangle\}$$

[6] Due to the sub-modularity of EUS^{NL}, selecting $\mathbf{y}_0$ with maximal expected utility and $\mathbf{y}_1 = \text{argmax} EUS^{NL}(\mathbf{y}_0, \mathbf{y})$ yields a bounded loss compared to the maximization of $EUS^{NL}(\mathbf{y}, \mathbf{y}')$; and by construction $\mathbf{y}_t^*$ has a high expected utility. On the other hand, the difference between EUS^N and EUS^{NL} is limited after Lemma 2.

[7] A minor issue is that uniformly drawing a demonstration $\mathbf{y}$ of the candidate $\mathbf{x}$ might leave the user with a poorly informative demonstration. This drawback can be handled in practice by recording the sampled demonstrations and showing the most informative one (or the most representative of $\mathbf{x}_t^*$) to the expert [Wilson *et al.* 2012, Jain *et al.* 2013].

For $i \geq 1$, let $\bar{\mathbf{y}}_i$ denote the average demonstration of $\mathbf{x}_i$, and θ_i denote the posterior on $\mathbf{W}$ from $\mathcal{U}_t \cup \{(\bar{\mathbf{y}}_i \succ \mathbf{y}_t^*)\}$; the $i+1$-th optimization problem is defined as:

$$\mathbf{x}_{i+1} = \text{ argmax } \{\mathcal{G}_{i+1}(\mathbf{x}) = \langle \mathbb{E}_{\theta_i}[\mathbf{w}], \bar{\mathbf{y}} \rangle\}$$

Proposition 2

The expected utility of selection of $(\bar{\mathbf{y}}_i, \mathbf{y}_t^*)$ monotonically increases with i:

$$EUS_t^{NL}(\bar{\mathbf{y}}_i, \mathbf{y}_t^*) \leq EUS_t^{NL}(\bar{\mathbf{y}}_{i+1}, \mathbf{y}_t^*)$$

***Proof**: By construction of $\mathbf{x}_{i+1}$,*

$$\int_{\mathbf{W}, \bar{\mathbf{y}}_i \succ \mathbf{y}_t^*} \langle \mathbf{w}, \bar{\mathbf{y}}_i \rangle \mathrm{d}\theta_t(\mathbf{w}) \leq \int_{\mathbf{W}, \bar{\mathbf{y}}_i \succ \mathbf{y}_t^*} \langle \mathbf{w}, \bar{\mathbf{y}}_{i+1} \rangle \mathrm{d}\theta_t(\mathbf{w})$$

Hence (omitting domain $\mathbf{W}$ for clarity)

$$\begin{aligned}
EUS_t^{NL}(\bar{\mathbf{y}}_i; \mathbf{y}_t^*) &= \int \max(\langle \mathbf{w}, \bar{\mathbf{y}}_i \rangle, \langle \mathbf{w}, \mathbf{y}_t^* \rangle) \mathrm{d}\theta_t(\mathbf{w}) \\
&= \int_{\bar{\mathbf{y}}_i \succ \mathbf{y}_t^*} \langle \mathbf{w}, \bar{\mathbf{y}}_i \rangle \mathrm{d}\theta_t(\mathbf{w}) + \int_{\bar{\mathbf{y}}_i \prec \mathbf{y}_t^*} \langle \mathbf{w}, \mathbf{y}_t^* \rangle \mathrm{d}\theta_t(\mathbf{w}) \\
&\leq \int_{\bar{\mathbf{y}}_i \succ \mathbf{y}_t^*} \langle \mathbf{w}, \bar{\mathbf{y}}_{i+1} \rangle \mathrm{d}\theta_t(\mathbf{w}) + \int_{\bar{\mathbf{y}}_i \prec \mathbf{y}_t^*} \langle \mathbf{w}, \mathbf{y}_t^* \rangle \mathrm{d}\theta_t(\mathbf{w}) \\
&\leq \int \max(\langle \mathbf{w}, \bar{\mathbf{y}}_{i+1} \rangle, \langle \mathbf{w}, \mathbf{y}_t^* \rangle) \mathrm{d}\theta_t(\mathbf{w}) \\
&= EUS_t^{NL}(\mathbf{y}_{i+1}, \mathbf{y}_t^*) \qquad \blacksquare
\end{aligned}$$

As this sequence of optimization problems leads to a local optimum of EUS_t^{NL}, multiple restarts are used with different samples $\mathbf{w}_0$ and the best solution is retained.

Overall, the elementary optimization component in PF is concerned with optimizing $\langle \bar{\mathbf{w}}, \bar{\mathbf{y}} \rangle$ on $\mathcal{X}$. Different implementations of this optimization component are considered depending on $\mathcal{X}$ (next Section). As $\bar{\mathbf{w}}$ is the average utility function under distribution θ_t, this requires to integrate over $\mathbf{W}$ once, thus with tractable cost. In the experiments, less than 10 iterations are needed to find a local optimum, and 10 multiple restarts were considered.

Algorithmically, the search for π_k given $\mathbf{w}_k$ proceeds as follows. When considering a discrete state space of restricted size, the canonical feature function measures the visiting frequency of each state[8] ($\phi_j(s^i) = \delta_{i,j}$ with $\delta_{i,j} = 1$ iff $i = j$ and 0

[8] In the general case of a feature function on a discrete state space, a change of representation on the feature space can be used to get back to the canonical feature function case.

otherwise). In this case, the i-th coordinate w_i of the utility can be interpreted as the reward associated to s_i and standard RL algorithms such as Policy Iteration can be used to find π_k from $\mathbf{w}_k$.
In the case of a large or continuous state space, state of the art approximate RL algorithms such as LSTD or GTD [Lagoudakis & Parr 2003a, Sutton *et al.* 2009] can be used. Note however that while the used RL algorithms assume infinite length trajectories, only finite-length trajectories will be demonstrated to the expert. The trajectory length H must thus be adjusted depending on the discount factor γ, to ensure that the cumulative reward for $t > H$ can be safely discarded.

Eventually, three optimization components are considered, respectively addressing the discrete case, the white-box continuous case (known transition model), and the black-box continuous case (unknown transition model).

7.6 Experimental validation

The PF framework is experimentally validated on three well studied benchmark RL problems involving discrete and continuous solution spaces $\mathcal{X}$, with (the bicycle problem) or without (the cartpole, the grid-world)a known transition model. Two additional experiments have been performed in the robotic context, teaching a Nao robot to raise its arm, and learning to dock with precision for an e-puck robot to a motionless docking destination. Each experiment is targeted toward a dedicated issue regarding PF, from its sensitivity to the hyper-parameters of the noise model to how it scales up with the sizes of the state space, the action space or the solution space.

In all experiments, the true utility function is known, though of course it is not available to the algorithm and is only used to analyze the results. The user preferences are simulated with a realistic criterion, on top of which noise is added following the ridge noise model described in Section 7.3. Different core optimization procedures are used, however, depending on the characteristics of the test case, as discussed above (see also Section 2.5).

The computational time is less than 1 minute per run on a 2.4Ghz Intel processor for all problems except the Nao problem (10 mns).

The results are assessed comparatively to [Wilson *et al.* 2012] (presented in chap-

ter 4), which is the most closely related approach.

As mentioned, both [Wilson *et al.* 2012] and APRIL are based on an active preference-based policy learning scheme, exploiting pairwise preferences among trajectories demonstrated by the agent. While PPL and APRIL relax the strong expertise requirement of RL and IRL, they still assume dedicated experts, taking the time to look at the agent demonstrations, and reliably ranking them.

The expert burden is limited in [Wilson *et al.* 2012] by only considering short demonstrations, assuming that i) the initial states can be sampled after some prior distribution in order to enforce inspecting interesting regions of the behavioral space; ii) the user can compare sub-behaviors. [Wilson *et al.* 2012] thus assume that quite some expertise about the problem domain is available, expressed through an informative prior about interesting initial states; additionally, it assumes more knowledgeable users since they need to assess the quality of the end state of the demonstration.

Comparatively, APRIL requires the expert to look at long demonstrations (thus providing complete information about the robot behavior); in counterpart, these demonstrations might be boring especially in the early training phases, thereby increasing the chances of ranking noise, and thus motivating the PF approach, that takes possible ranking noise into account.

7.6.1 Discrete Case with Unknown Transition Model

A stochastic *Grid World* problem is considered, on a 5×5 grid (Fig. 7.3). The goal of the experiments is to analyze the sensitivity of the results w.r.t. the two parameters of the noise model, M_A and M_E. The experimental conditions are the following:

- The state space has 25 possible states (number of positions on the grid);
- The action space is made of 5 actions (up, down, left, right or stay motionless);
- The transition model involves a 50% probability of staying motionless (100% if the selected action would send the agent in the wall). It is estimated from 1,000 random triplets.

- The solution space is the controller space, functions from the state space (of size 25) onto the action set (of size 5);
- The demonstration space is $\mathbb{R}^{25}$, associating to each state the number of times it is visited during the trajectory.
- The true utility of demonstration ($y = s_{i_1}, \ldots s_{i_H}$) is given as

$$J(y) = \sum_{h=1}^{H} w^*_{i_h}$$

  denoting w^*_i the reward associated to the i-th square;
- The user feedback is emulated using the ridge noise model defined in section 7.3 (Equation 7.3), where δ is uniformly selected in $[0, M_E]$ (Hypothesis 1 Section 7.3), for different values of M_E $(0.25, 0.5, 1.0)$;
- The core optimization component (Section 7.5) implements a vanilla policy iteration algorithm, with $\gamma = .95$;
- The time horizon is set to $H = 300$;
- All results are averaged over 21 runs.

The grid and the reward function are shown in Fig. 7.3, that displays on each square the associated reward.

The choice of the gridworld problem, where the true utility function is known from vector $\mathbf{w}^*$, is meant to analyze the sensitivity of the PF framework w.r.t. the hyper-parameters of the noise model. The user's feedback is emulated using hyper-parameter M_E, referred to as *user's incompetency* for short (the higher M_E, the less competent the user); M_A is the hyper-parameter of the user's noise model estimated by the active computer (denoted by M in sections 7.4 and 7.5), referred to as *agent distrust* for short (the higher M_A, the more the active computer underestimates the user's competence), with M_E and M_A ranging in $\{1, .5, .25\}$ subject to $M_A \geq M_E$.

The performance indicator is the true utility of demonstration $\mathbf{y}_t$ in each PF time step (reminding that the true utility is unknown to the active computer). Fig. 7.4 shows that the performance primarily depends on the user's incompetency M_E, and secondarily on the active computer estimate M_A of the user's competence.

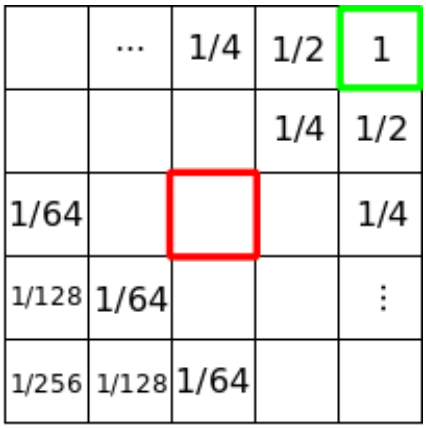


Figure 7.3: The Grid World. The starting state is in the middle of the grid. To each i-th square is associated a weight w_i^*. The true utility function is the sum of the w_i^* taken over all squares i visited during the trajectory (non discounted cumulative reward with time horizon 300).

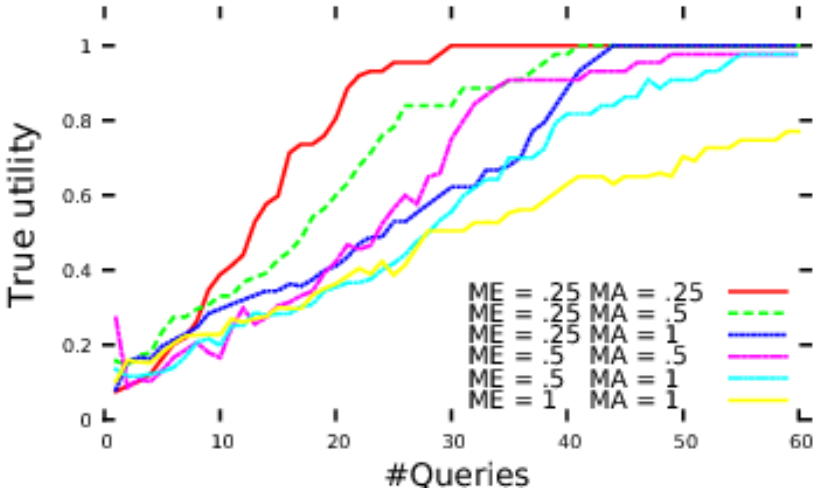


Figure 7.4: Results on the Grid World problem: Impact of the user's incompetency M_E and agent distrust M_A on the performance, measured as the true utility of the demonstration $\mathbf{y}_t$ displayed after t user's feedbacks (results averaged on 21 runs).

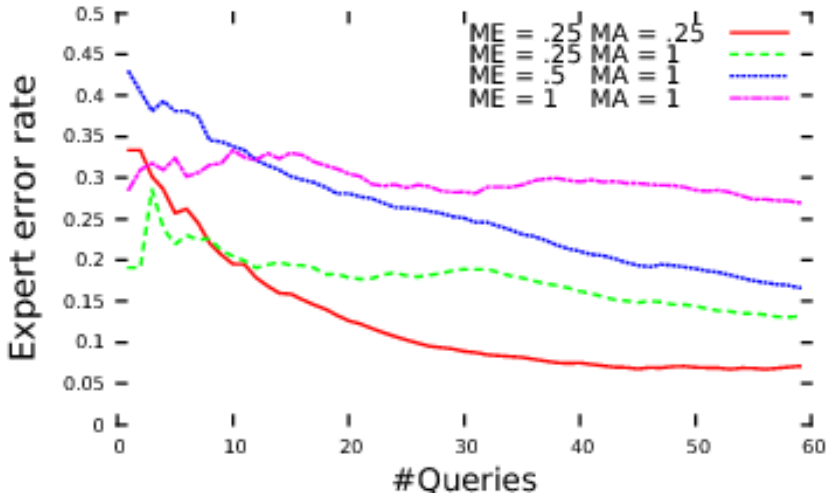


Figure 7.5: Impact of the user's incompetency and agent distrust on the average number of mistakes of the emulated user (average on 21 runs).

A more detailed analysis reveals that the number of the emulated user's mistakes does not only depend on the user's incompetency, as expected, but also on the agent distrust: the number of user's mistakes most surprisingly increases as the active computer underestimates the user's competence (high M_A), irrespective of the user's true competence M_E (Fig. 7.5). This fact is explained as the error rate does not only depend on the user's (in)competency but also on the relevance of the demonstrations he is provided with. For $M_A = .25$, the active computer learns faster, thus submitting more relevant demonstrations to the user, thus priming a virtuous educational process. This also explains the fast decrease of the error rate for $M_E = .25, M_A = .25$ which seems to follow a different regime (empirically faster than linear) than $M_E = .25, M_A = 1$. Fig 7.4 also supports the claim that PF most critical stage is the initial one, where the demonstrated trajectories are of low utility, making them harder to compare and increasing the probability of user's mistakes.

A second goal of the gridworld experiment is to study the PF scalability w.r.t. the size of the state space. The target behavior is reached in 30 PF interactions in a 25-state space. This relatively slow convergence is blamed on the poor representation of the demonstrations. The representation does not reflect the structure of the state space (the topology of the grid is only implicit from the transition model), which

explains that PF convergence is in $\mathcal{O}(|S|)$ like the standard PI mechanism.

7.6.2 Continuous Case, Unknown Transition Model

The *Cartpole* problem is concerned with balancing a pole fixed to a movable cart (Fig. 7.6). The goal of the experiments is to validate the PF approach in continuous state space. The same setting as in [Lagoudakis & Parr 2003a] is used, leading to the following experimental conditions:

- The state space is $\mathbb{R}^2$ (the angle and angular velocity of the pendulum);
- The action space includes 3 discrete actions (one step to the right, one step to the left, staying motionless);
- The transition model is estimated from 33,000 (s, a, s') triplets;
- The solution space is that of Q-value functions, from $\mathbb{R}^9 \times \{1, 2, 3\}$ onto $\mathbb{R}$;
- The demonstration space $\mathcal{Y}$ is $\mathbb{R}^9$, where the i-th coordinate corresponds to a Gaussian $\mathcal{N}(\mu_i, \sigma_i Id)$ in the $\mathbb{R}^2$ state space. The representation of a demonstration $(s_1, \ldots s_H)$ is given as $\mathbf{y} \in \mathbb{R}^9$ where

  $$y_i = \sum_{h=1}^{H} \gamma^h Pr(s_h | \mathcal{N}(\mu_i, \sigma_i Id))$$

  The discounted sum is computed using LSTD (with reward $r(s) = p(s|\mathcal{N}(\mu_i, \sigma_i))$;
- The true utility is the fraction of time the cartpole is in equilibrium;
- As in the Grid World experiment, the user feedback is emulated using the ridge noise model defined in section 7.3 (Equation 7.3), where δ is uniformly selected in $[0, M_E]$ (Hypothesis 1 Section 7.3), for the same values of M_E $(0.25, 0.5, 1.0)$;
- The PF core optimization component (Section 7.5) implements LSPI [Lagoudakis & Parr 2003a].
- The time horizon H is 3,000 time steps;
- All results are averaged over 21 runs.

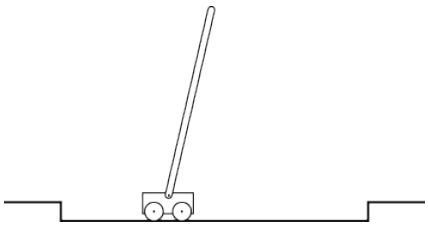


Figure 7.6: The cartpole problem. The starting state is at the (unstable) equilibrium. The time horizon in $H = 3,000$. The true utility function is the fraction of the time where the cartpole is in equilibrium.

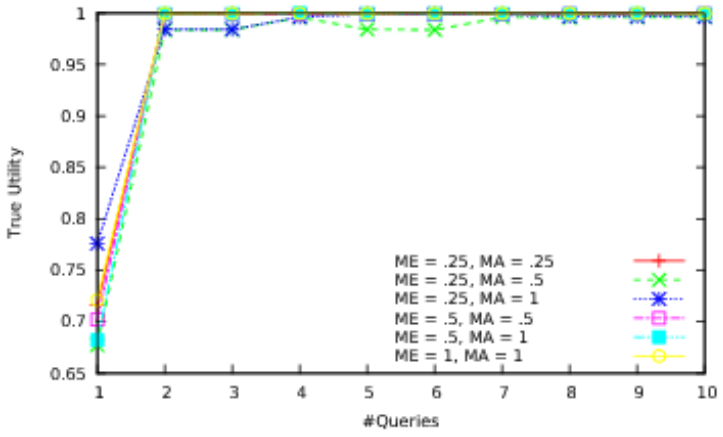


Figure 7.7: Results on the cartpole problem: PF performance, measured as the true utility of the demonstration $\mathbf{y}_t$ displayed after t user's feedback.

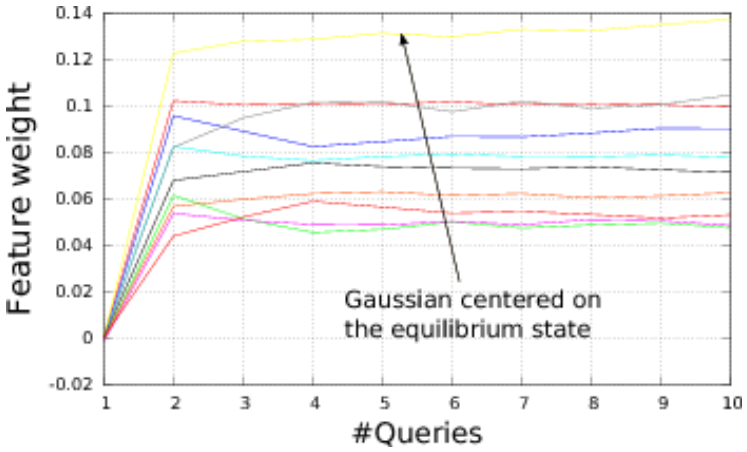


Figure 7.8: Results on the cartpole problem: Learned utility vector $\mathbf{w}_t$ after t user's feedbacks, in parallel coordinates; each curve depicts the weight for one of the representative features.

The results show that only two user's feedback are required on average to solve the cartpole problem, irrespective of the noise model hyper-parameters M_A and M_E (Fig. 7.7). Beside confirming that the cartpole is indeed an easy problem [Igel 2003], these results favorably compare to [Wilson *et al.* 2012], where circa 15 queries are required to maintain the cartpole in equilibrium more than 1,400 time steps. Fig. 7.8, displaying the estimated utility vector $\mathbf{w}_t$ in parallel coordinates, shows that the feature closest to the equilibrium position gets the highest weight: PF does learn the expected utility function.

7.6.3 Continuous Case, with Known Transition Model

The considered *Bicycle Balancing* problem aims at riding the bicycle and preventing it from falling down [Wilson *et al.* 2012]. Note that this task is a subtask of the more complex task of *Bicycle Balancing and Riding* proposed by [Lagoudakis & Parr 2003a] where an additional goal was to ride the bicycle toward a given target location. In particular, compared to the latter work, all state variables and features related to the target of the ride are not relevant and have been omitted. The goal of these experiments for PF is to investigate the scalability of the approach w.r.t. the dimensionality of the solution space. The experimental conditions are the following:

- The state space is $\mathbb{R}^4$ (the angle ω and angular velocity $\dot{\omega}$ of the bicycle, the angle θ and angular velocity $\dot{\theta}$ of the handlebars, see Fig. 7.9);
- The action space is $\mathbb{R}^2$ (the torque applied to the handlebars and the displacement of the rider on the saddle);
- The generative model is the simulator proposed in [Randløv & Alstrøm 1998] (and used by [Lagoudakis & Parr 2003a]), but re-implemented following [Ernst *et al.* 2005];
- The solution space $\mathcal{X}$ is $\mathbb{R}^{210}$: the controller is a 1-layer feed-forward NN with 4 inputs, 29 hidden neurons and 2 outputs.
- The demonstration space is $\mathbb{R}^{14}$, i.e., the 14^{th} first base functions proposed in [Lagoudakis & Parr 2003a] for the complete task of balancing and riding;
- The true utility measures the deviation from the vertical equilibrium position of the bicycle averaged over the whole demonstration, i.e.,

$$\frac{1}{T_{end}} \sum_{t=0}^{t=T_{end}} (1 - \omega^2)$$

  where T_{end} is the total demonstration length (T_{max} below, or the time until the bicycle falls on the ground), and $\omega(t)$ the instantaneous value of the angle of the bicycle.
- The user's feedback is emulated by considering that the best demonstration is the longest one, subject to the ridge noise model (Equation 7.3), where δ is uniformly selected in $[0, M_E]$ (Hypothesis 1 Section 7.3), for again the same values of M_E $(0.25, 0.5, 1.0)$;
- The PF core optimization component (Section 7.5) implements the CMA-ES black-box optimization algorithm [Hansen & Ostermeier 2001] (see discussion below);
- The maximum demonstration length T_{max} is 30,000 time steps; however controllers tend to either fall down before 300 time steps, or to maintain the bicycle until the end of the trajectory.
- All results are averaged over 21 runs.

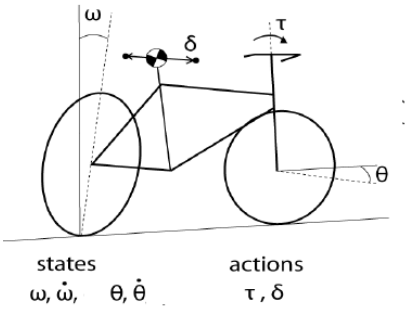


Figure 7.9: The bicycle problem. The starting state is the (unstable) equilibrium. The time horizon in $H = 30,000$. The true utility function is (1 - squared angle of bicycle), averaged on the whole demonstration.

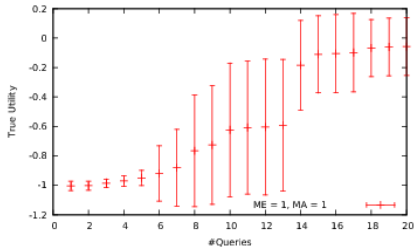


Figure 7.10: The bicycle problem. Performance measured as the true utility function of the demonstration $\mathbf{y}_t$ after t user's feedback for $M_e = M_A = .25$.

Our first experiments used LSPI as core optimization component, on the basis of the learned utility function, LSPI repeatedly failed to deliver a decent controller. Inspecting the utility function, it was visible that many coordinates of vectors $\mathbf{w}_t$ were non zero (as might be expected in an early training stage), which apparently misled LSPI. Our tentative interpretation for this failure is that the Q-value is learned by minimizing the mean-square error criterion whereas the LSPI efficiency requires the L_∞ error be bounded, as discussed in [Munos 2003]).

Overall, 15 user'r feedback are required on average to solve the bicycle problem for the low noise setting ($M_E = M_A = 1$); the results gracefully degrade as the noise increases (Fig. 7.10).

Figure 7.11: The Nao robot. A single degree of freedom is considered in the experiments, the angle of the arm.

These results favorably compare to those in [Wilson *et al.* 2012], where circa 20 preference queries are required to reach the equilibrium although their setting only involves 5 discrete actions.

The improvement is explained as [Wilson *et al.* 2012] operates in the policy parameter space whereas PF operates on the trajectory utility space, thus reducing the number of parameters by a factor 5. This reduction amounts to the difference in the number of parameters between a Q-Value and a V-Value function. A more compact representation of a policy can be used (for instance the weights of a neural network instead of the Q-Value of a policy) but it may yield a highly non-smooth optimization landscape.

7.6.4 The Nao robot

A robotic experiment with discrete state space and supposedly no simulator aims at training the **Nao robot**, a 58cm high humanoid robot [Aldebaran 2013] to reach a target state, e.g., raising the right hand (Fig. 7.11). The goal of this experiment is to validate the PF approach in a real, though yet simple, robotic context. The experimental conditions, however, are similar to those of the Grid World problem (section 7.6.1):

- The state space involves only one degree of freedom, the position of the right arm. Two settings are considered, with different discretizations of the arm position (namely 13 and 20).
- The action space includes 3 actions, move one step up, move one step down, and do nothing;

- The transition matrix is estimated from 1,000 random (s, a, s') triplets, in order to simulate the case of an unknown transition model, even though here it is in fact known (from one state to the neighbor one, up or down, in case of the corresponding action);
- The solution space is the controller space, functions from the state space (of size 13 or 20) onto the action set (of size 3);
- The demonstration space is $\mathbb{R}^d$, where d is the number of states, and a trajectory stores for each state the number of times the arm was in that state;
- The true utility of demonstration $(y = s_{i_1}, \ldots s_{i_H})$ is given as

$$J(y) = \sum_{h=1}^{H} w^*_{i_h}$$

  denoting w^*_i the reward associated to the i-th square, with a maximum reward on the target state, and decreasing rewards on the neighbor states depending on their distance to the target state;
- The user feedback is emulated by ranking first the arm position that is closest to the target position (e.g., the higher the better if the target is the highest position), plus using the ridge noise model defined in section 7.3 (Equation 7.3), where δ is uniformly selected in $[0, M_E]$ (Hypothesis 1 Section 7.3), for $M_E = 0.5$;
- The core optimization component (Section 7.5) implements a vanilla policy iteration algorithm, with $\gamma = .95$;
- The maximum trajectory length (time horizon) is 10; the initial state is fixed.
- Results are averaged over 5 runs.

Fig. 7.12 shows that 10 PF interactions are required to reach a target state in a 13-state space (the shortest demonstration reaching the target state is 5-actions long). In the finer-grained discretization of the state space (20 states), the experiments show that about 24 interactions (average results on 5 runs) are required to reach the target state (the shortest demonstration reaching the target state is 10-actions long).

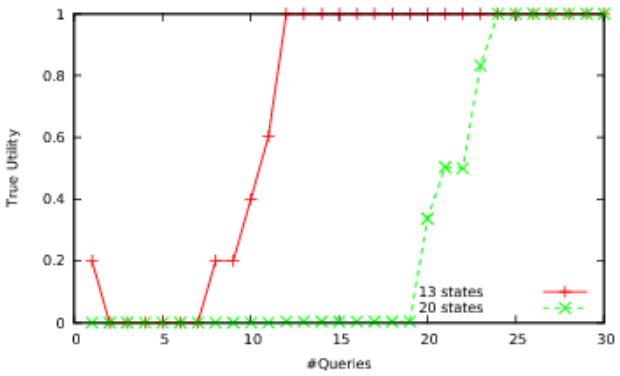
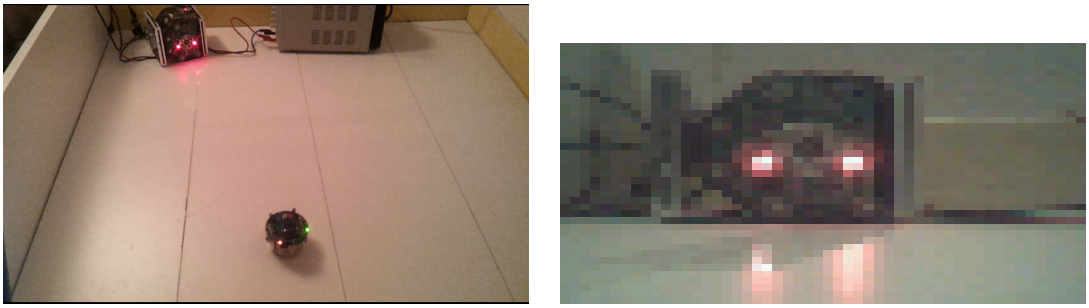


Figure 7.12: The Nao robot: performance vs number of user's feedback (average on 5 runs).



(*a*) (*b*)

Figure 7.13: The robot docking problem: a) Initial state from an observer's perspective; b) the e-puck camera perspective.

7.6.5 Robot docking

Another problem in the domain of robotics, that also belongs to the "discrete case and known transition model" category was motivated by the Symbrion project [Symbrion 2009]. It is concerned with a robot precisely aligning itself with another robot for docking purpose, i.e., in order to start creating a multi-robot organism (Fig. 7.13). The simplified setting considered here involves a single e-puck robot equipped with a (52x39, 4img/s) camera; the target behavior is to reach a motionless robot that has its LEDs on. The starting state, from the perspective of the expert, is displayed in Fig. 7.13.a.

- The state space is made of 16 states that have been manually defined from the

current camera image: using hand-tuned thresholds, a set of pixel positions S_{lum} corresponding to the target LEDs is extracted at each time step. S_{lum} is then used to estimate both the distance and the alignment w.r.t. the target robot. An estimate of the distance is obtained by segmenting the image with four horizontal lines (resulting in 5 segments) and returning the segment number to which the highest pixel position of S_{lum} belongs to. Similarly, three vertical lines are used to determine whether the target robot is on the left, on the right or centered in the camera image by looking at the segment to which belongs the average between the leftmost and rightmost pixel position of S_{lum}. The sixteenth state corresponds to $S_{lum} = \{\}$ (no target in sight).

- The action space includes 5 discrete actions: stay motionless during one time step, move forward (resp. backward) during 3 time steps, and rotate to the left (resp. to the right) during one time step;

- The transition model is partly estimated and partly gathered from expert knowledge: the transition matrix was first estimated from 1,000 random (s, a, s') triplets, but then had to be manually corrected due to the high level of noise in order to allow any meaningful results to be obtained;

- The solution space is finite, associating one action to each state;

- The demonstration space is defined as follows. Letting the current trajectory generated from policy $\pi_{\mathbf{x}}$ be noted $s_0, \ldots s_{H-1}$, where $s_i \in \{1, \ldots, 16\}$ is the index of the robot state at time i, the associated behavioral representation is set to $\Phi(\mathbf{x}) \in \mathbb{R}^{16}$ with $\Phi(\mathbf{x})[j] = \sum_{h=0}^{H-1} \gamma^h \delta_{j,s_h}$ where δ_{j,s_h} is 1 iff the robot is in state j at time h, and 0 otherwise. Parameter γ is set to .95 in the experiment. The resulting $\mathbf{y}$ was normalized thereafter s.t. $||\mathbf{y}||_1 = 1$.

- The true utility is the level of precision of the docking;

- The user feedback is the author's actual feedback, estimating the precision of the docking; no noise (other than the natural noise of a real human) was added; However, the active computer robot did estimate the user's noise model, with agent's distrust $M_A = 1$ (the M in sections 7.4 and 7.5);

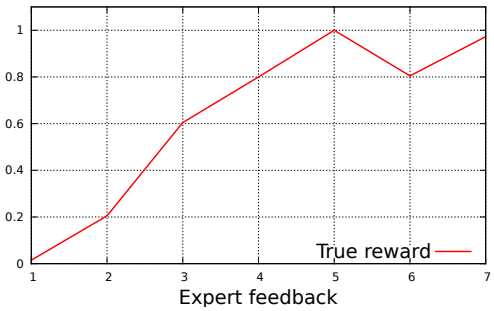


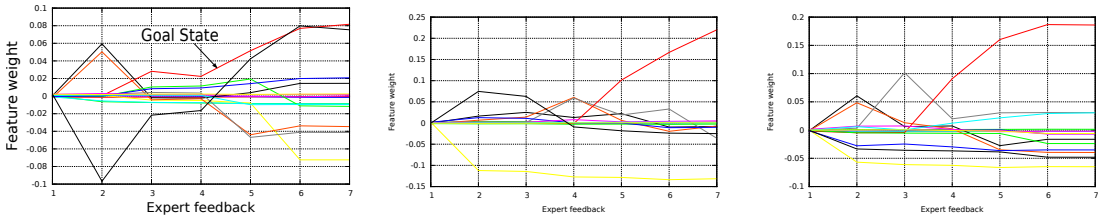
Figure 7.14: The robot docking problem: Performance of the policy maximizing the average utility *vs* number of interactions with the expert, averaged out of 5 runs.

- The PF core optimization component (Section 7.5) proceeds as follows: 50 initial utilities $\mathbf{w}_0$ sampled from $p(w, \theta_t)$ were considered and a Policy Iteration algorithm was used, taking as input a provided transition matrix and $\mathbf{w}$ and returning the average trajectory $\bar{\mathbf{y}}$ of the policy maximizing $\mathbf{w}$;
- The robot demonstrates the selected policy for 80 time steps, it then gets the user's feedback and the user sets the robot back to its initial position;
- All results are averaged over 5 runs.

In order to keep the whole process on-board, the user's feedback is directly interpreted using a built-in procedure: the user simply activates the front (resp.) the back e-puck sensors to indicate that the current behavior is better (resp. worse) than the previous best one;

Figure 7.14 shows the visit to the goal state (averaged out of 5 runs) *vs* the number of expert's feedback, a.k.a. number of interactions. Interestingly, all 5 policies were found to reach the goal state after 5 interactions; the observed performance decrease is explained by inspecting the logs, showing that the expert made an error in one of the runs, favoring a trajectory that spent significantly less time in the goal state.

Fig. 7.15.a, b, c) shows the average utility weight vector *vs* the number of interactions for three particular runs. Interestingly, the weight associated to some states increases after the first interactions, and thereafter decreases, due to the fact that these states are intermediate between the starting and the goal states. Fig.



a *b* *c*

Figure 7.15: The robot docking problem: Utility weight vector in $\mathbb{R}^{16}$ *vs* number of interactions for three different runs.

7.15.c is the run where the expert made a mistake (at the third feedback), causing the weight associated to the "nothing in sight" state to increase and out-pass the other weights; this in turn leads astray the optimal policy; however PF recovers one iteration later. Concerning the execution time, since the RL is based on the transition matrix, almost all of the time is solely consumed by the demonstrations of trajectories to the expert.

7.7 Partial Conclusion and Discussion

The state of the art, from inverse reinforcement learning to preference-based policy learning, from co-active policy learning to programming by feedback, shows an evolution of the trade-off among two interdependent issues: the level of expertise required from the expert and the level of autonomy endowed to the agent. Simultaneously, the expert's knowledge requirement is relaxed, and the agent autonomy is increased.

In decreasing expertise order, the expert is required to

i) associate a reward to any state in the state space (RL);
ii) demonstrate a solution behavior (LfD);
iii) correct and improve an agent demonstration (CPL);
iv) rank two demonstrations (PPL, CPL);
v) rank two demonstrations with limited reliability (PF).

To see that iii) requires more expertise than iv), consider the following analogy: option iii) provides the direction of the gradient in the demonstration space; option

iv) merely indicates the half space the gradient belongs to. Along the same line, preference mistakes might severely harm the learning process as they point at the wrong half space; mistakes in the direction of the gradient are easier to cope with in the general case.

In parallel, in increasing autonomy order, the agent

i) computes the optimal policy based on the instant rewards (RL);
ii) imitates *verbatim* the expert's demonstration (LfD);
iii) imitates the expert's demonstrations with some generalization and variants (LfD);
iv) learns the expert's utility function and produces a (near) optimal demonstration according to this function (CPL, IRL);
v) learns the expert's utility function and produces a (near) optimal demonstration in the sense of the information gained about the utility function and its impact on the overall solution quality (expected posterior utility) (PPL);
vi) learns the expert's utility function and produces a (near) optimal demonstration as in PPL, while accounting for the possibly limited competency of the expert (PF).

As already noted, the ILO goal significantly differs from that of active ranking: active ranking is interested in learning the true preference function, whereas ILO only aims at its optimum.

A third issue regards the modelling of the uncertainty and priors on the expert's utility function. In [Brochu *et al.* 2007], the utility function is learned as a GP, thus providing both a utility estimate and the confidence thereof. Interestingly however, [Brochu *et al.* 2010] note that ILO being a "small data" problem, does not provide enough evidence to accurately tune the GP hyper-parameters. The process thereby suffers a loss of performance due to under-optimal utility estimate. Additionally, the exploration *vs* exploitation tradeoff is hardly controlled in a Bayesian setting: the zero-mean utility function requires non visited regions to be explored instead of quickly exploiting the most promising ones, despite the fact that most regions are uninteresting.

An original contribution of the PF framework in this respect is to account for the uncertainties in the expert's utility function, primarily due to the expert's

mistakes, while remaining short of the full GP handling of uncertainties as in [Brochu *et al.* 2007, Brochu *et al.* 2010].

This contribution led to most unexpected experimental observations, regarding the interplay between the user's competence and the agent trust into the user's competence. It was most unexpected that the agent trust (hyper-parameter M_A) might have an impact on the actual number of mistakes done by the user. Commonly, the user's competence is considered to be a given, which is not supposed to vary at least in the short term.

In retrospect however, it is quite clear that the *observed* user's competence does not only depend on her true competence (hyper-parameter M_E in the experiments), but also on the agent queries: in brief, there is no way to yield good answers to confuse questions. In the early training phases in particular, the experiments suggest that, up to certain limits naturally, it might be more effective for the agent to trust more a less competent expert, than to trust less a more competent one. A cumulative (dis)advantage phenomenon is observed. On the one hand, a pessimistic competence estimate leads the active computer to present the user with poorly informative queries, *thereby increasing the probability for the user to make errors* and be inconsistent everything else being equal. On the other hand, when the active computer trusts a competent user, its skills steadily improve while the user's error rate stays low. Overall, the importance of a good educational start is witnessed: poor initial demonstrations lead to a poor utility model, leading itself to poorly informative queries.

Early evidence supported the claim that some users deal with their computers as if they were persons [Reeves & Nass 1996], showing that the computer "social skills" (intended as, interaction skills) influence the user's perception of the computer "objective skill". The whole domain of Human Machine Interaction suggests more generally that one should design the computer interaction and functional skills in an integrated way, the whole being different from the sum of its parts.

The PF framework offers an even more integrated perspective on the human and the computer in the loop; as the computer autonomy increases, the human partner is no longer the only master of the game. Along this new and intricate interaction, it is

shown that the human observed performance can actually depend on the computer priors.

The fact that the human performance might depend on the priors of his partners and on subtle signals in the environment, has been vastly documented in social studies (see [Inzlicht 2011]), e.g. related to gender and prejudices). It is the first time however, to our best knowledge, that the observed human performance is shown to depend on his non-human partner. This result naturally inspires a major challenge: if the computer priors influence the human observed competence, how to tune these priors in order to get the best of the interaction ?

Chapter 8

Conclusion and Perspectives

An integrated framework has been presented in this manuscript, aimed at allowing a computational agent to get the best training from weak experts. The first version of this framework, the Preference-based Policy Learning 5, was limited as it failed to consider the expert's guidance as a rare resource... For this reason, the second version of the framework, the Active Preference-based Reinforcement Learning 6, focused on reducing the number of queries to the expert needed for the computational agent to acquire the desired skills. A side effect of this reduction however is that the lesser the number of queries, the more harmful the expert's mistakes are. Still, there is little doubt that experts – even strong experts – are bound to make mistakes. Finally, the third and current version of the framework, the Programming by Feedback system 7, endows the computational agent with a (fixed) model of the expert, affording to robustly cope with the expert's mistakes. The experiments on thoroughly studied RL benchmark problems confirm that a handful of binary queries to the expert is enough to yield a satisfactory controller.

Let us first comment upon the nature of the expertise weaknesses accounted for by PF, and our motivations for handling them. The short and mid-term perspectives opened for further research are thereafter discussed.

8.1 Facing weak experts

The expertise requirements involved in PF differ from those in the state of the art in two main respects. Firstly, the expert is only required to give a binary feedback – as opposed to [Jain *et al.* 2013] where the expert can manually modify and improve the agent behavior. Secondly, the expert gives her feedback on full trajectories, as opposed to [Wilson *et al.* 2012] where sub-trajectories are considered, and [Knox *et al.* 2013] where the feedback is related to the recent state-action pairs.

As discussed in chapter 7, judging a sub-behavior actually requires more expertise than judging a full trajectory: in the latter case, the expert knows what has been done; in the former case, the expert must be able to assess what could have been done (whether the sub-behavior is conducive to reaching the eventual goals). Additionally, in order to consider informative sub-behaviors, quite some knowledge is required about the distribution of "interesting" starting points (the pivotal points of the trajectories) [Wilson *et al.* 2012].

These assumptions of little or no prior knowledge, and weak expertise, are relevant to several contexts.

The first and most critical context is when the expert does not know how to implement the desired behavior of the computational agent: the expert knows what she wants, she does not know how to get it. A current societal pitfall is that computer-illiterate persons find it hard to get what they want from an environment crowded with computational agents. The PF framework aims at relieving this pitfall, afford such persons to interact with computational agents and exert some control on (and feel in control of) the situation. The only necessary condition for this affordance to come true is that the weak expert must know whether the agent behaviour is evolving in the right direction[1].

A second context is when the computational agent(s) is(are) trained by an ensemble of weak experts. This might be the case when training a robotic swarm, which was the motivating application of the PhD. It might also be the case when the sought controller is a complex one and/or when the target behaviour should actually achieve different goals and address a multi-objective optimization problem. In such a case indeed the weak experts cannot be assumed to be consistent with each other. And even in the case where the inconsistency of the expert ensemble is alleviated by enabling the computational agent to identify the different experts, the agent must still be prepared for each individual expert to make mistakes.

A third and most interesting context is when the expert is deliberately modifying its educational purposes, along a predefined curriculum; in such a case, the expert has in mind a set of skills to be acquired by the computational agent, and a training schedule in order to do so. As mentioned, this situation is analogous to Fitness

[1] An alternative would be for the expert to reject all current behavioural directions proposed by the agent, and force the exploration of other behaviours, in the spirit of [Lim & Auer 2012].

shaping [Skinner 1951, Ng *et al.* 1999], also known as *Continuation Method* in the optimization literature [Wasserstrom 1973].

This curriculum-inspired approach might be the most effective answer to the main objection of the PF scheme, namely that it is too simple to acquire really complex skills: indeed, acquiring a complex skill from scratch might be unrealistic via PF; but gradually acquiring more complex skills, or acquiring the behavioural building-blocks involved in a complex skill, might be much more realistic.

8.2 Research perspectives

Some short- and mid-term research perspectives aim at extending the PF framework as follows.

8.2.1 Reconsidering the feature design

An agnostic, clustering-based, representation of the state and/or state-action space was used in chapters 5 and 6, mapping the continuous space onto discrete sensory-motor states. However, the experiments on the Grid World and the Bicycle problems (section 7.6) suggest that a discretized representation of the state space might be ill-suited to PF: while the Grid-World involves a much simpler state space and transition model compared to the Bicycle, it requires circa twice more interactions with the user to find the optimal policy. This is blamed on the fact that each state defines a coordinate of the search space in the Grid-World; the state topology structure is not reflected in the behavioural representation. To see the detrimental impact of this fact on the PF resolution, consider that two trajectories, each visiting a single state, are maximally dissimilar irrespective of whether these states are close or far.

A perspective for further research is to remedy the lack of structure of the feature mapping ϕ, by using a convolution with the state neighbourhood structure induced from the trajectories, e.g.

$$\phi(s) \rightarrow \frac{\sum_{j,s_k^{(j)}=s,s_{k+t}^{(j)}=s'} \phi(s')e^{-\alpha t}}{\sum_{j,s_k^{(j)}=s,s_{k+t}^{(j)}=s'} e^{-\alpha t}}$$

where j ranges over the available trajectories and α controls the convolution scope.

Such an extended mapping will be analyzed comparatively to the object-to-object and object-to-environment features of [Jain *et al.* 2013] (section 4.5).

8.2.2 PF Initialization and Transition Models

As mentioned, the expert might want to use the PF framework to consider a sequence of optimization problems along a behaviour shaping strategy, aimed at the gradual acquisition of different skills. In such cases, and as far as the considered skills refer to the same environment, the preliminary identification of the transition model might be factorized over the sequence of optimization problems: the learned transition model reduces the computation time for each subsequent task and support a smoother interaction.

To do so, the initialization step of the PF framework might target the simple exploration of the environment, using e.g. robust intrinsic motivation [Oudeyer *et al.* 2010], sensory-motor entropy maximization [Delarboulas *et al.* 2010] or multi-armed bandit-based exploration [Lim & Auer 2012].

8.2.3 Hybridizing PF and LfD

By construction, PF is robust to the choice of the first trajectory (generated along a random policy in all reported experiments[2]). Therefore, the PF approach should also stand its initialization with a user's demonstration, irrespective of the demonstration relevance.

On the other hand, as shown by [Kim *et al.* 2013], the use of sub-optimal demonstrations together with standard RL gets the best of both worlds of RL and LfD (assuming that the relevance of the demonstrations is somehow indicated through the trade-off parameter, section 3.5.2).

It thus comes naturally to explore the initialization of the PF framework with a user's demonstration, if available. Indeed, the computational agent would not know the relevance of this demonstration; however, as long as the expert keeps interacting with the computational agent, the demonstration cannot mislead durably the system. Still, large behavioural improvements especially in the first PF iterations

[2]Except for the first experiments in chapter 5, where the first trajectory was generated using a controller optimized using [Delarboulas *et al.* 2010].

can be expected as the demonstration would focus the search on interesting regions of the state space.

Independently, it might be encouraging for the expert to influence the behaviour of the computational agent and speed-up its first (and most wandering) steps in the search space.

8.2.4 Increasing the learner's autonomy and refining the expert's competence model

An independent issue concerns the model of the expert embedded in the computational agent. As noted, the expert noise model depends in a rough way of a single fixed hyper-parameter, interpreted as the "trust" the agent may have w.r.t. the expert competence. This model is robust: it makes room for the fact that the user's competence might vary along time (e.g. due to fatigue) and depend on the behavioural space region being explored (e.g. due to competence). Still, it makes sense, after a prolonged interaction, that the computational agent comes to know better the expert, and identify the δ parameter most consistent with the user's consistency for the sake of a faster convergence toward the desired skills. The challenge here is to refine the noise model while preserving the possibility for the user's to be, so to speak, inconsistently inconsistent.

8.2.5 Feedback and constraints

An important issue regards the interpretation of the expert feedback in terms of sub-behaviours. As said, the generation of sub-behaviours based on prior knowledge raises some issues [Wilson *et al.* 2012]. However, a long trajectory can be viewed *de facto* as a set of sub-behaviours.

In the meanwhile, it is likely that the expert's feedback is actually related to fragments of the trajectory: some sub-behaviours are no longer appreciated as they are taken for granted since they have been discovered for a while; a full trajectory might be appreciated despite some weaknesses as it contains a new and relevant fragment of behaviour.

Along this line, the PF framework can be extended to discover *a posteriori* the sub-behaviours relevant to explain the expert's feedback, by replacing the

propositional learning-to-rank module with a multiple-instance ranking approach [Bergeron *et al.* 2008]. MIP ranking would thus allow the computational agent to identify which sub-behaviours might be responsible for the user's feedback, e.g. to identify the most wrong or most promising sub-behaviours. Thereby, the MIP-extended PF would support the selection of interesting sub-behaviours, thus saving the user's time and attention as in [Wilson *et al.* 2012].

Unifying the feedback interpretation

A longer-term perspective is to allow the computational agent flexibly decide upon the granularity of the feedback causes (full trajectory, sub-behaviour, state-action pair), along a unified framework. Such a unified perspective could be centered on a fixed point problem formulation: based on the current distribution θ_t over the utility space $\mathbf{W}$ (see chapter 7), the agent would determine the segments of the current trajectory most likely to explain the expert's feedback; thereby, the distribution over $\mathbf{W}$ defines the preference constraints while the constraints shape the distribution over $\mathbf{W}$.

In addition to providing a more powerful exploitation of the user's feedback, a fixed-point framework brings two advantages. The first one is to allow the computational agent to retrospectively reinterpret a past user's feedback, increasing its robustness towards incompetent users. Secondly, the search for fixed points might provide an additional mechanism for selecting the next policy to demonstrate, e.g. breaking the tie between different fixed points (each of which entailing a different interpretation of the user's feedback) and dispelling the ambiguity.

Bibliography

[Abbeel & Ng 2004] Pieter Abbeel and Andrew Y. Ng. *Apprenticeship learning via inverse reinforcement learning.* In Carla E. Brodley, editeur, ICML, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004. (Cited on pages 2, 25, 26, 30, 31, 32, 56, 58, 61, 76, 77, 80, 83, 85 and 88.)

[Abbeel 2008] P. Abbeel. *Apprenticeship Learning and Reinforcement Learning with Application to Robotic Control.* PhD thesis, Stanford University, 2008. (Cited on page 2.)

[Ailon 2012] Nir Ailon. *An Active Learning Algorithm for Ranking from Pairwise Preferences with an Almost Optimal Query Complexity.* Journal of Machine Learning Research, vol. 13, pages 137–164, 2012. (Cited on page 5.)

[Akrour *et al.* 2011] R. Akrour, M. Schoenauer and M. Sebag. *Preference-Based Policy Learning.* In ECML/PKDD (1), numéro 6911 de LNCS, pages 12–27. Springer Verlag, 2011. (Cited on pages 5, 40, 55 and 76.)

[Akrour *et al.* 2012] R. Akrour, M. Schoenauer and M. Sebag. *APRIL: Active Preference Learning-Based Reinforcement Learning.* In ECML/PKDD (2), volume 7524, pages 116–131. Springer LNCS, 2012. (Cited on pages 5, 75 and 93.)

[Akrour *et al.* 2014] R. Akrour, M. Schoenauer, M. Sebag and J.-C. Souplet. *Programming by Feedback.* In Int. Conf. on Machine Learning (ICML), ACM Int. Conf. Proc. Series, 2014. (Cited on pages 6 and 91.)

[Aldebaran 2013] Ltd. Aldebaran. *Discover NAO*, 2013. (Cited on page 111.)

[Antos *et al.* 2007] András Antos, Rémi Munos and Csaba Szepesvári. *Fitted Q-iteration in continuous action-space MDPs.* In John C. Platt, Daphne Koller, Yoram Singer and Sam T. Roweis, editeurs, NIPS. Curran Associates, Inc., 2007. (Cited on page 17.)

[Antos *et al.* 2008] András Antos, Csaba Szepesvári and Rémi Munos. *Learning near-optimal policies with Bellman-residual minimization based fitted policy*

iteration and a single sample path. Machine Learning, vol. 71, no. 1, pages 89–129, 2008. (Cited on page 36.)

[Atkeson & Schaal 1997a] Christopher G. Atkeson and Stefan Schaal. *Learning tasks from a single demonstration.* In ICRA, pages 1706–1712. IEEE, 1997. (Cited on page 28.)

[Atkeson & Schaal 1997b] Christopher G. Atkeson and Stefan Schaal. *Robot Learning From Demonstration.* In Proceedings of the Fourteenth International Conference on Machine Learning, ICML '97, pages 12–20, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. (Cited on pages 25 and 28.)

[Auer *et al.* 2002] Peter Auer, Nicolo Cesa-Bianchi and Paul Fischer. *Finite-time Analysis of the Multiarmed Bandit Problem.* Machine Learning, vol. 47, pages 235–256, 2002. (Cited on page 2.)

[Auger 2005] A. Auger. *Convergence Results for the (1,λ)-SA-ES using the Theory of φ-irreducible Markov Chains.* Theoretical Computer Science, vol. 334, no. 1-3, pages 35–69, 2005. (Cited on pages 21, 63 and 64.)

[Bain & Sammut 1995] M. Bain and C. Sammut. *A framework for behavioural cloning.* In K. Furukawa, D. Michie and S. Muggleton, editeurs, Machine Intelligence, volume 15, pages 103–129. Oxford University Press, 1995. (Cited on pages 25, 26, 28 and 56.)

[Baird 1995] Leemon Baird. *Residual Algorithms: Reinforcement Learning with Function Approximation.* In In Proceedings of the Twelfth International Conference on Machine Learning, pages 30–37. Morgan Kaufmann, 1995. (Cited on page 17.)

[Bakir *et al.* 2006] G. Bakir, T. Hofmann, B. Scholkopf, A.J. Smola, B. Taskar and S.V.N. Vishwanathan. Machine learning with structured outputs. MIT Press, 2006. (Cited on pages 57 and 62.)

[Bellman & Dreyfus 1962] Richard Bellman and Stuart E. Dreyfus. Applied dynamic programming. Princeton University Press, 1962. (Cited on page 11.)

[Bellman 1957] Richard Bellman. Dynamic programming. Princeton University Press, Princeton, NJ, USA, 1 édition, 1957. (Cited on page 9.)

[Bergeron *et al.* 2008] Charles Bergeron, Jed Zaretzki, Curt M. Breneman and Kristin P. Bennett. *Multiple instance ranking.* In Proc. ICML, pages 48–55, 2008. (Cited on page 125.)

[Bertsekas & Ioffe 1996] Dimitri P. Bertsekas and Sergey Ioffe. *Temporal differences-based policy iteration and applications in neuro-dynamic programming.* Rapport technique, MIT, 1996. (Cited on page 13.)

[Bertsekas & Tsitsiklis 1996] Dimitri P. Bertsekas and John N. Tsitsiklis. Neuro-dynamic programming. Athena Scientific, 1st édition, 1996. (Cited on pages 9, 12 and 13.)

[Billard & Grollman 2013] A. Billard and D. Grollman. *Robot learning by demonstration.* Scholarpedia, vol. 8, no. 12, page 3824, 2013. revision 138061. (Cited on page 25.)

[Bou-Ammar *et al.* 2013] Haitham Bou-Ammar, Decebal Constantin Mocanu, Matthew E. Taylor, Kurt Driessens, Karl Tuyls and Gerhard Weiss. *Automatically Mapped Transfer between Reinforcement Learning Tasks via Three-Way Restricted Boltzmann Machines.* In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen and Filip Zelezný, editeurs, ECML/PKDD (2), volume 8189 of *Lecture Notes in Computer Science*, pages 449–464. Springer Verlag, 2013. (Cited on page 26.)

[Boutilier 2002] Craig Boutilier. *A POMDP Formulation of Preference Elicitation Problems.* In Eighteenth National Conference on Artificial Intelligence, pages 239–246, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. (Cited on page 45.)

[Bradtke *et al.* 1996] Steven J. Bradtke, Andrew G. Barto and Pack Kaelbling. *Linear least-squares algorithms for temporal difference learning.* In Machine Learning, pages 22–33, 1996. (Cited on page 18.)

[Brafman & Tennenholtz 2003] Ronen I. Brafman and Moshe Tennenholtz. *R-max - a General Polynomial Time Algorithm for Near-optimal Reinforcement*

Learning. J. Mach. Learn. Res., vol. 3, pages 213–231, March 2003. (Cited on page 15.)

[Bredeche 2006] N. Bredeche, 2006. http://www.lri.fr/∼bredeche/roborobo/. (Cited on page 67.)

[Brochu *et al.* 2007] Eric Brochu, Nando de Freitas and Abhijeet Ghosh. *Active Preference Learning with Discrete Choice Data*. In NIPS, 2007. (Cited on pages 43, 117 and 118.)

[Brochu *et al.* 2008] E. Brochu, N. de Freitas and A. Ghosh. *Active Preference Learning with Discrete Choice Data*. In Proc. NIPS 20, pages 409–416, 2008. (Cited on pages 63, 76 and 77.)

[Brochu *et al.* 2010] E. Brochu, T. Brochu and N. de Freitas. *A Bayesian Interactive Optimization Approach to Procedural Animation Design*. In Z. Popovic and M. A. Otaduy, editeurs, Symposium on Computer Animation, pages 103–112. Eurographics Association, 2010. (Cited on page 118.)

[Brown & Peterson 2009] Thomas C. Brown and George L. Peterson. *An Enquiry Into the Method of Paired Comparison- Reliability, Scaling, and Thurstones Law of Comparative Judgment*. Gen Tech. Rep. United States Department of Agriculture, 2009. (Cited on page 43.)

[Brunskill & Li 2014] Emma Brunskill and Lihong Li. *PAC-inspired Option Discovery in Lifelong Reinforcement Learning*. In Proc. ICML 2014, volume 32 of *JMLR Proceedings*, pages 1–9. JMLR.org, 2014. (Cited on page 2.)

[Burges *et al.* 2013] Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani and Kilian Q. Weinberger, editeurs. Advances in neural information processing systems 26: 27th annual conference on neural information processing systems 2013. proceedings of a meeting held december 5-8, 2013, lake tahoe, nevada, united states, 2013. (Cited on pages 134, 136 and 137.)

[Cakmak & Thomaz 2012] Maya Cakmak and Andrea L. Thomaz. *Designing Robot Learners That Ask Good Questions*. In Proceedings of the Seventh Annual ACM/IEEE International Conference on Human-Robot Interaction, HRI '12, pages 17–24, New York, NY, USA, 2012. ACM. (Cited on pages 27 and 38.)

[Calinon & Billard 2005] Sylvain Calinon and Aude Billard. *Recognition and reproduction of gestures using a probabilistic framework combining PCA, ICA and HMM.* In Raedt & Wrobel [Raedt & Wrobel 2005], pages 105–112. (Cited on page 28.)

[Calinon *et al.* 2007] S. Calinon, F. Guenter and A. Billard. *On Learning, Representing and Generalizing a Task in a Humanoid Robot.* IEEE Trans. on Systems, Man and Cybernetics, Special Issue on Robot Learning by Observation, Demonstration and Imitation, vol. 37, no. 2, pages 286–298, 2007. (Cited on pages 25 and 56.)

[Chajewska *et al.* 2000] Urszula Chajewska, Daphne Koller and Ronald Parr. *Making Rational Decisions Using Adaptive Utility Elicitation.* In Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pages 363–369. AAAI Press, 2000. (Cited on page 45.)

[Chang *et al.* 2012] Ming-Wei Chang, Lev-Arie Ratinov and Dan Roth. *Structured learning with constrained conditional models.* Machine Learning, vol. 88, no. 3, pages 399–431, 2012. (Cited on page 1.)

[Cheng *et al.* 2011] Weiwei Cheng, Johannes Fürnkranz, Eyke Hüllermeier and Sang-Hyeun Park. *Preference-Based Policy Iteration: Leveraging Preference Learning for Reinforcement Learning.* In Dimitrios Gunopulos, Thomas Hofmann, Donato Malerba and Michalis Vazirgiannis, editeurs, Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2011, volume 6911 of *Lecture Notes in Computer Science*, pages 312–327. Springer Verlag, 2011. (Cited on pages 3, 40, 48, 78 and 86.)

[Chevallier *et al.* 2010] Sylvain Chevallier, Hélène Paugam-Moisy and Michèle Sebag. *SpikeAnts, a spiking neuron network modelling the emergence of organization in a complex system.* In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel and A. Culott, editeurs, NIPS'2010, pages 379–387, 2010. (Cited on page 3.)

[Chu & Ghahramani 2005a] Wei Chu and Zoubin Ghahramani. *Extensions of gaussian processes for ranking: semi-supervised and active learning.* In In NIPS Workshop on Learning to Rank, 2005. (Cited on pages 40 and 44.)

[Chu & Ghahramani 2005b] Wei Chu and Zoubin Ghahramani. *Preference learning with Gaussian processes.* In Raedt & Wrobel [Raedt & Wrobel 2005], pages 137–144. (Cited on pages 41, 43 and 96.)

[Coates *et al.* 2008] Adam Coates, Pieter Abbeel and Andrew Y. Ng. *Learning for control from multiple demonstrations.* In William W. Cohen, Andrew McCallum and Sam T. Roweis, editeurs, ICML, volume 307 of *ACM International Conference Proceeding Series*, pages 144–151. ACM, 2008. (Cited on pages 25, 27 and 28.)

[Cortes & Vapnik 1995] Corinna Cortes and Vladimir Vapnik. *Support-Vector Networks.* Machine Learning, vol. 20, no. 3, pages 273–297, 1995. (Cited on page 77.)

[Dann 2012] Christoph Dann. *Algorithms for Fast Gradient Temporal Difference Learning*, 2012. (Cited on page 19.)

[Dasgupta 2005] Sanjoy Dasgupta. *Coarse sample complexity bounds for active learning.* In Advances in Neural Information Processing Systems 18, 2005. (Cited on pages 39, 78 and 83.)

[Deisenroth *et al.* 2013] M. P. Deisenroth, G. Neumann and J. Peters. *A Survey on Policy Search for Robotics.* Foundations and Trends in Robotics, vol. 2, no. 1-2, pages 1–142, 2013. (Cited on pages 1, 2 and 94.)

[Dekel *et al.* 2008] Ofer Dekel, Shai Shalev-Shwartz and Yoram Singer. *The Forgetron: A Kernel-Based Perceptron on a Budget.* SIAM J. Comput., vol. 37, pages 1342–1372, January 2008. (Cited on page 73.)

[Delarboulas *et al.* 2010] Pierre Delarboulas, Marc Schoenauer and Michèle Sebag. *Open-Ended Evolutionary Robotics: An Information Theoretic Approach.* In Robert Schaefer, Carlos Cotta, Joanna Kolodziej and Günter Rudolph, editeurs, PPSN (1), volume 6238 of *Lecture Notes in Computer Science*, pages 334–343. Springer, 2010. (Cited on page 124.)

[Duda & Hart 1973] R.O. Duda and P.E. Hart. Pattern classification and scene analysis. John Wiley and sons, Menlo Park, CA, 1973. (Cited on pages 60, 61 and 80.)

[Erbas *et al.* 2011] Mehmet Dinçer Erbas, Alan F. T. Winfield and Larry Bull. *Towards imitation-enhanced Reinforcement Learning in multi-agent systems.* In Proc. ALIFE, pages 6–13. IEEE, 2011. (Cited on page 3.)

[Erbas *et al.* 2014] Mehmet Dinçer Erbas, Alan F. T. Winfield and Larry Bull. *Embodied imitation-enhanced reinforcement learning in multi-agent systems.* Adaptive Behaviour, vol. 22, no. 1, pages 31–50, 2014. (Cited on page 3.)

[Ernst *et al.* 2003] Damien Ernst, Pierre Geurts and Louis Wehenkel. *Iteratively extending time horizon reinforcement learning.* In N. Lavra, L. Gamberger and L. Todorovski, editeurs, Proceedings of the 14th European Conference on Machine Learning, pages 96–107, Dubrovnik, Croatia, September 2003. Springer-Verlag Heidelberg. (Cited on page 17.)

[Ernst *et al.* 2005] Damien Ernst, Pierre Geurts and Louis Wehenkel. *Tree-Based Batch Mode Reinforcement Learning.* Journal of Machine Learning Research, vol. 6, pages 503–556, 2005. (Cited on page 109.)

[Farahmand *et al.* 2008] Amir Massoud Farahmand, Mohammad Ghavamzadeh, Csaba Szepesvári and Shie Mannor. *Regularized Policy Iteration.* In Daphne Koller, Dale Schuurmans, Yoshua Bengio and Léon Bottou, editeurs, NIPS, pages 441–448, 2008. (Cited on page 36.)

[Fern *et al.* 2006] Alan Fern, Sungwook Yoon and Robert Givan. *Approximate Policy Iteration with a Policy Language Bias: Solving Relational Markov Decision Processes.* J. Artif. Int. Res., vol. 25, no. 1, pages 75–118, January 2006. (Cited on page 16.)

[Fonteneau *et al.* 2010] R. Fonteneau, S. A. Murphy, L. Wehenkel and D. Ernst. *A Cautious Approach to Generalization in Reinforcement Learning.* In Proc. ICAART, pages 64–73, 2010. (Cited on page 60.)

[Freund *et al.* 1997] Yoav Freund, H. Sebastian Seung, Eli Shamir and Naftali Tishby. *Selective Sampling Using the Query by Committee Algorithm.* Machine Learning, vol. 28, no. 2-3, pages 133–168, 1997. (Cited on page 39.)

[Gelly & Silver 2007] Sylvain Gelly and David Silver. *Combining Online and Offline Knowledge in UCT.* In International Conference of Machine Learning, 2007. (Cited on pages 2 and 16.)

[Gelly *et al.* 2012] Sylvain Gelly, Marc Schoenauer, Michèle Sebag, Olivier Teytaud, Levente Kocsis, David Silver and Csaba Szepesvari. *The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions.* Communications-ACM, vol. 55, no. 3, pages 106–113, 2012. (Cited on page 2.)

[Geramifard *et al.* 2007] Alborz Geramifard, Michael Bowling, Martin Zinkevich and Richard S. Sutton. *iLSTD: Eligibility Traces and Convergence Analysis.* In B. Schölkopf, J.C. Platt and T. Hoffman, editeurs, Advances in Neural Information Processing Systems 19, pages 441–448. MIT Press, 2007. (Cited on page 19.)

[Ghavamzadeh *et al.* 2010] Mohammad Ghavamzadeh, Alessandro Lazaric, Odalric Maillard and Rémi Munos. *LSTD with Random Projections.* In J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel and A. Culotta, editeurs, Advances in Neural Information Processing Systems 23, pages 721–729. Curran Associates, Inc., 2010. (Cited on page 2.)

[Griffith *et al.* 2013] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L. Isbell and Andrea Lockerd Thomaz. *Policy Shaping: Integrating Human Feedback with Reinforcement Learning.* In Burges et al. [Burges *et al.* 2013], pages 2625–2633. (Cited on page 47.)

[Groß *et al.* 2006] R. Groß, M. Bonani, F. Mondada and M. Dorigo. *Autonomous Self-Assembly in Swarm-Bots.* IEEE Transactions on Robotics, vol. 22, no. 6, pages 1115–1130, 2006. (Cited on pages 3, 4 and 57.)

[Hachiya & Sugiyama 2010] Hirotaka Hachiya and Masashi Sugiyama. *Feature Selection for Reinforcement Learning: Evaluating Implicit State-Reward Dependency via Conditional Mutual Information.* In Proc. ECML/PKDD (1),

volume 6321 of *Lecture Notes in Computer Science*, pages 474–489, 2010. (Cited on page 80.)

[Hansen & Ostermeier 2001] N. Hansen and A. Ostermeier. *Completely Derandomized Self-Adaptation in Evolution Strategies.* Evolutionary Computation, vol. 9, no. 2, pages 159–195, 2001. (Cited on pages 20, 85 and 110.)

[Heidrich-Meisner & Igel 2009] V. Heidrich-Meisner and C. Igel. *Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search.* In ICML, volume 382 of *ACM Int. Conf. Proc. Series*, page 51, 2009. (Cited on pages 19, 20 and 88.)

[Herbrich *et al.* 2001] Ralf Herbrich, Thore Graepel and Colin Campbell. *Bayes Point Machines.* Journal of Machine Learning Research, vol. 1, pages 245–279, 2001. (Cited on pages 81, 82 and 84.)

[Herdy 1996] M. Herdy. *Evolution Strategies with Subjective Selection.* In H.-M. Voigt, W. Ebeling, I. Rechenberg and H.-P. Schwefel, editeurs, Proc. PPSN IV, volume 1141 of *LNCS*, pages 22–31. Springer Verlag, 1996. (Cited on page 39.)

[Hester *et al.* 2012] Todd Hester, Michael Quinlan and Peter Stone. *RTMBA: A Real-Time Model-Based Reinforcement Learning Architecture for robot control.* In ICRA, pages 85–90. IEEE, 2012. (Cited on page 37.)

[Hockley 1984] W. E. Hockley. *Analysis of response time distributions in the study of cognitive processes.* Journal of Experimental Psychology: Learning, Memory, and Cognition, vol. 10, no. 4, pages 598–615, 1984. (Cited on page 46.)

[Hoos 2012] H. H. Hoos. *Programming by optimization.* Commun. ACM, vol. 55, no. 2, pages 70–80, 2012. (Cited on page 92.)

[Hsu 2002] Feng-hsiung Hsu. Behind deep blue: Building the computer that defeated the world chess champion. Princeton University Press, 2002. (Cited on page 2.)

[Hüllermeier & Fürnkranz 2013] Eyke Hüllermeier and Johannes Fürnkranz. *Editorial: Preference learning and ranking.* Machine Learning, vol. 93, no. 2-3, pages 185–189, 2013. (Cited on page 3.)

[Hüllermeier *et al.* 2008] Eyke Hüllermeier, Johannes Fürnkranz, Weiwei Cheng and Klaus Brinker. *Label Ranking by Learning Pairwise Preferences.* Artif. Intell., vol. 172, no. 16-17, pages 1897–1916, November 2008. (Cited on page 48.)

[Igel 2003] Christian Igel. *Neuroevolution for Reinforcement Learning Using Evolution Strategies.* In R. Sarker et al., editeur, Proc. IEEE Congress on Evolutionary Computation, pages 2588–2595. IEEE Press, 2003. (Cited on pages 19 and 107.)

[Inzlicht 2011] Michael Inzlicht. Stereotype threat: Theory, process, and application. Oxford University Press, 2011. (Cited on page 119.)

[Jain *et al.* 2013] Ashesh Jain, Brian Wojcik, Thorsten Joachims and Ashutosh Saxena. *Learning Trajectory Preferences for Manipulators via Iterative Improvement.* In Burges et al. [Burges *et al.* 2013], pages 575–583. (Cited on pages 5, 25, 40, 50, 52, 53, 93, 100, 121 and 123.)

[Jaksch *et al.* 2010] Thomas Jaksch, Ronald Ortner and Peter Auer. *Near-optimal Regret Bounds for Reinforcement Learning.* J. Mach. Learn. Res., vol. 11, pages 1563–1600, August 2010. (Cited on page 15.)

[Joachims 2002] Thorsten Joachims. *Optimizing search engines using clickthrough data.* In KDD, pages 133–142. ACM, 2002. (Cited on page 31.)

[Joachims 2005] T. Joachims. *A Support Vector Method for Multivariate Performance Measures.* In Raedt & Wrobel [Raedt & Wrobel 2005], pages 377–384. (Cited on pages 5 and 57.)

[Joachims 2006] Thorsten Joachims. *Training Linear SVMs in Linear Time.* In Tina Eliassi-Rad, Lyle H. Ungar, Mark Craven and Dimitrios Gunopulos, editeurs, Proc. KDD, pages 217–226. ACM, 2006. (Cited on pages 62, 68 and 85.)

[Jones *et al.* 1998] Donald R. Jones, Matthias Schonlau and William J. Welch. *Efficient Global Optimization of Expensive Black-Box Functions.* J. of Global Optimization, vol. 13, no. 4, pages 455–492, December 1998. (Cited on pages 43 and 44.)

[Kaariainen 2006] Matti Kaariainen. *Lower Bounds for Reductions.* In Atomic Learning Workshop, 2006. (Cited on page 34.)

[Kaelbling *et al.* 1996] Leslie Pack Kaelbling, Michael L. Littman and Andrew W. Moore. *Reinforcement learning: a survey.* Journal of Artificial Intelligence Research, vol. 4, pages 237–285, 1996. (Cited on page 7.)

[Kaelbling *et al.* 1998] Leslie Pack Kaelbling, Michael L. Littman and Anthony R. Cassandra. *Planning and acting in partially observable stochastic domains.* Artificial Intelligence, vol. 101, no. 1–2, pages 99 – 134, 1998. (Cited on page 20.)

[Kakade & Tewari 2008] Sham M. Kakade and Ambuj Tewari. *On the Generalization Ability of Online Strongly Convex Programming Algorithms.* In Daphne Koller, Dale Schuurmans, Yoshua Bengio and Léon Bottou, editeurs, NIPS 21, pages 801–808, 2008. (Cited on page 35.)

[Kaneko & Tsuda 2000] K. Kaneko and I. Tsuda. Complex systems: Chaos and beyond. Springer-Verlag, 2000. (Cited on page 3.)

[Khamassi *et al.* 2005] Mehdi Khamassi, Loïc Lachèze, Benoît Girard, Alain Berthoz and Agnès Guillot. *Actor–Critic models of reinforcement learning in the basal ganglia: from natural to artificial rats.* Adaptive Behavior, vol. 13, no. 2, pages 131–148, 2005. (Cited on page 1.)

[Kim *et al.* 2013] Beomjoon Kim, Amir Massoud Farahmand, Joelle Pineau and Doina Precup. *Learning from Limited Demonstrations.* In Burges et al. [Burges *et al.* 2013], pages 2859–2867. (Cited on pages 27, 35 and 124.)

[Knox & Stone 2009] W. Bradley Knox and Peter Stone. *Interactively Shaping Agents via Human Reinforcement: The TAMER Framework.* In Proceedings of the Fifth International Conference on Knowledge Capture, K-CAP '09, pages 9–16, New York, NY, USA, 2009. ACM. (Cited on pages 40 and 46.)

[Knox & Stone 2010] W. Bradley Knox and Peter Stone. *Combining manual feedback with subsequent MDP reward signals for reinforcement learning.* In Wiebe van der Hoek, Gal A. Kaminka, Yves Lespérance, Michael Luck and Sandip Sen, editeurs, AAMAS, pages 5–12. IFAAMAS, 2010. (Cited on page 47.)

[Knox & Stone 2012] W. Bradley Knox and Peter Stone. *Reinforcement learning from simultaneous human and MDP reward.* In Wiebe van der Hoek, Lin Padgham, Vincent Conitzer and Michael Winikoff, editeurs, AAMAS, pages 475–482. IFAAMAS, 2012. (Cited on pages 5, 40, 46 and 47.)

[Knox *et al.* 2013] W. B. Knox, P. Stone and C. Breazeal. *Training a Robot via Human Feedback: A Case Study.* In Int. Conf. on Social Robotics, volume 8239 of *LNCS*, pages 460–470. Springer, 2013. (Cited on pages 93 and 121.)

[Kocsis & Szepesvári 2006] Levente Kocsis and Csaba Szepesvári. *Bandit based Monte-Carlo Planning.* In Proc. ECML-06, numéro 4212 de LNCS, pages 282–293. Springer, 2006. (Cited on page 2.)

[Koller & Parr 2000] Daphne Koller and Ronald Parr. *Policy Iteration for Factored MDPs.* In In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00, pages 326–334. Morgan Kaufmann, 2000. (Cited on page 18.)

[Kolter *et al.* 2007] J. Zico Kolter, Pieter Abbeel and Andrew Y. Ng. *Hierarchical Apprenticeship Learning with Application to Quadruped Locomotion.* In NIPS. MIT Press, 2007. (Cited on pages 56, 61 and 88.)

[Lagoudakis & Parr 2003a] Michail Lagoudakis and Ronald Parr. *Least-Squares Policy Iteration.* Journal of Machine Learning Research (JMLR), vol. 4, pages 1107–1149, 2003. (Cited on pages 18, 101, 106, 107 and 109.)

[Lagoudakis & Parr 2003b] Michail G. Lagoudakis and Ronald Parr. *Reinforcement Learning as Classification: Leveraging Modern Classifiers.* In in Proceedings of the Twentieth International Conference on Machine Learning, pages 424–431, 2003. (Cited on pages 16, 17 and 48.)

[Lange *et al.* 2012] Sascha Lange, Martin Riedmiller and Arne Voigtländer. *Autonomous reinforcement learning on raw visual input data in a real world application.* In Proc. IJCNN, pages 1–8. IEEE, 2012. (Cited on page 60.)

[LaValle & Jr. 2001] Steven M. LaValle and James J. Kuffner Jr. *Randomized Kinodynamic Planning.* I. J. Robotic Res., vol. 20, no. 5, pages 378–400, 2001. (Cited on page 51.)

[LeCun *et al.* 2006] Y. LeCun, U. Muller, E. Cosatto and B. Flepp. *Off-Road Obstacle Avoidance through End-to-End Learning.* In NIPS - Advances in Neural Information Processing Systems 18, 2006. (Cited on page 26.)

[Lehman & Stanley 2008] J. Lehman and K.O. Stanley. *Exploiting Open-Endedness to solve problems through the search for novelty.* In Proc. of the Eleventh International Conference on AI Life (AILife-08), pages 329–336. MIT Press, 2008. (Cited on pages 68 and 70.)

[Levine *et al.* 2010] S. Levine, Z. Popovic and V. Koltun. *Feature Construction for Inverse Reinforcement Learning.* In NIPS 23, pages 1342–1350, 2010. (Cited on page 61.)

[Lim & Auer 2012] Shiau Hong Lim and Peter Auer. *Autonomous Exploration For Navigating In MDPs.* In Shie Mannor, Nathan Srebro and Robert C. Williamson, editeurs, Proc. COLT 2012, volume 23 of *JMLR Proceedings*, pages 40.1–40.24. JMLR.org, 2012. (Cited on pages 122 and 124.)

[Lipson *et al.* 2006] Hod Lipson, Josh C. Bongard, Viktor Zykov and Evan Malone. *Evolutionary Robotics for Legged Machines: From Simulation to Physical Reality.* In IAS, pages 11–18, 2006. (Cited on page 56.)

[Littman *et al.* 2002] Michael L. Littman, Richard S. Sutton and Satinder Singh. *Predictive Representations of State.* In Neural Information Processing Systems 14, page 1555–1561, 2002. (Cited on page 76.)

[Liu & Winfield 2010] Wenguo Liu and Alan F. T. Winfield. *Modeling and Optimization of Adaptive Foraging in Swarm Robotic Systems.* Intl J. Robotic Research, vol. 29, no. 14, pages 1743–1760, 2010. (Cited on page 57.)

[Liu *et al.* 2011] Chengju Liu, Qijun Chen and Danwei Wang. *Locomotion control of quadruped robots based on CPG-inspired workspace trajectory generation.* In Proc. ICRA, pages 1250–1255. IEEE, 2011. (Cited on page 79.)

[Lizotte 2008] D. Lizotte. *Practical Bayesian Optimization.* PhD thesis, University of Alberta, 2008. (Cited on page 93.)

[Lörincz *et al.* 2007] A. Lörincz, V. Gyenes, M. Kiszlinger and I. Szita. *Mind Model Seems Necessary for the Emergence of Communication.* Neural Information Processing - Letters and Reviews, vol. 11, no. 4-6, pages 109–121, 2007. (Cited on page 92.)

[Luce 1959] R. D. Luce. Individual choice behavior. John Wiley, New York, 1959. (Cited on pages 91 and 96.)

[Lund *et al.* 1998] Henrik Hautop Lund, Orazio Miglino, Luigi Pagliarini, Aude Billard and Auke Ijspeert. *Evolutionary Robotics — A Children's Game.* In In Proceedings of IEEE 5th International Conference on Evolutionary Computation, pages 154–158. IEEE Press, 1998. (Cited on pages 39 and 40.)

[MacKay 1994] D. J. C. MacKay. *Bayesian Methods for Backpropagation Networks.* In E. Domany, J. L. van Hemmen and K. Schulten, editeurs, Models of Neural Networks III, chapitre 6, pages 211–254. Springer-Verlag, New York, 1994. (Cited on page 42.)

[Maei *et al.* 2010] Hamid Reza Maei, Csaba Szepesvári, Shalabh Bhatnagar and Richard S. Sutton. *Toward Off-Policy Learning Control with Function Approximation.* In Johannes Fürnkranz and Thorsten Joachims, editeurs, ICML, pages 719–726. Omnipress, 2010. (Cited on page 18.)

[Maes *et al.* 2009] Francis Maes, Ludovic Denoyer and Patrick Gallinari. *Structured prediction with reinforcement learning.* Machine Learning, vol. 77, no. 2-3, pages 271–301, 2009. (Cited on page 1.)

[Milani Fard *et al.* 2013] Mahdi Milani Fard, Yuri Grinberg, Amir massoud Farahmand, Joelle Pineau and Doina Precup. *Bellman Error Based Feature Generation using Random Projections on Sparse Spaces.* In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K.Q. Weinberger, editeurs, Advances

in Neural Information Processing Systems 26, pages 3030–3038. Curran Associates, Inc., 2013. (Cited on page 2.)

[Mnih *et al.* 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning.* CoRR, vol. abs/1312.5602, 2013. (Cited on pages 17 and 20.)

[Munos 2003] R. Munos. *Error Bounds for Approximate Policy Iteration.* In ICML, pages 560–567. AAAI Press, 2003. (Cited on page 110.)

[Ng & Russell 2000] A.Y. Ng and S. Russell. *Algorithms for Inverse Reinforcement Learning.* In P. Langley, editeur, Proc. 17th ICML, pages 663–670. Morgan Kaufmann, 2000. (Cited on pages 2, 25, 29 and 30.)

[Ng *et al.* 1999] Andrew Y. Ng, Daishi Harada and Stuart J. Russell. *Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping.* In Ivan Bratko and Saso Dzeroski, editeurs, ICML, pages 278–287. Morgan Kaufmann, 1999. (Cited on pages 23 and 122.)

[O'Dowd *et al.* 2011] Paul J. O'Dowd, Alan F. T. Winfield and Matthew Studley. *The distributed co-evolution of an embodied simulator and controller for swarm robot behaviours.* In Proc. IROS, pages 4995–5000. IEEE Press, 2011. (Cited on pages 3, 4 and 57.)

[O'Regan & Noë 2001] J.K. O'Regan and A. Noë. *A sensorimotor account of vision and visual consciousness.* Behavioral and Brain Sciences, vol. 24, page 939–973, 2001. (Cited on pages 60 and 80.)

[Oudeyer *et al.* 2010] Pierre-Yves Oudeyer, Adrien Baranes and Frédéric Kaplan. *Intrinsically Motivated Exploration for Developmental and Active Sensorimotor Learning.* In From Motor Learning to Interaction Learning in Robots, numéro 264 de Studies in Computational Intelligence, pages 107–146. Springer Verlag, 2010. (Cited on pages 73 and 124.)

[Peters & Schaal 2008] Jan Peters and Stefan Schaal. *Reinforcement Learning of Motor Skills with Policy Gradients.* Neural Networks, vol. 21, no. 4, pages 682–697, 2008. (Cited on pages 2, 56, 58, 63, 76 and 77.)

[Pineau *et al.* 2009] J. Pineau, A. Guez, R. Vincent, G. Panuccio and M. Avoli. *Treating epilepsy via adaptive neurostimulation: A reinforcement learning approach.* International Journal of Neural Systems, vol. 19, no. 4, pages 227–240, 2009. (Cited on page 1.)

[Poggio & Girosi 1990] T. Poggio and F. Girosi. *Networks for approximation and learning.* Proceedings of the IEEE, vol. 78, no. 9, pages 1481–1497, Sep 1990. (Cited on page 92.)

[Pomerleau 1989] Dean Pomerleau. *ALVINN: An Autonomous Land Vehicle In a Neural Network.* In D.S. Touretzky, editeur, Advances in Neural Information Processing Systems 1. Morgan Kaufmann, 1989. (Cited on pages 25, 26 and 28.)

[Puterman 1994] Martin L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. John Wiley & Sons, Inc., New York, NY, USA, 1st édition, 1994. (Cited on pages 12 and 13.)

[Raedt & Wrobel 2005] Luc De Raedt and Stefan Wrobel, editeurs. Machine learning, proceedings of the twenty-second international conference (icml 2005), bonn, germany, august 7-11, 2005, volume 119 of *ACM International Conference Proceeding Series.* ACM, 2005. (Cited on pages 131, 132 and 136.)

[Randløv & Alstrøm 1998] Jette Randløv and Preben Alstrøm. *Learning to drive a bicycle using reinforcement learn- ing and shaping.* In Jude W. Shavlik, editeur, Proc. 15th Intl Conf. on Machine Learning, page 463–471, 1998. (Cited on page 109.)

[Ranzato *et al.* 2006] Marc'Aurelio Ranzato, Christopher S. Poultney, Sumit Chopra and Yann LeCun. *Efficient Learning of Sparse Representations with an Energy-Based Model.* In NIPS, pages 1137–1144, 2006. (Cited on page 57.)

[Ratliff *et al.* 2006] Nathan D. Ratliff, J. Andrew Bagnell and Martin Zinkevich. *Maximum margin planning.* In ICML, volume 148, pages 729–736. ACM Int. Conf. Proc. Series, 2006. (Cited on page 26.)

[Ratliff 2009] Nathan Ratliff. *Learning to Search: Structured Prediction Techniques for Imitation Learning.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2009. (Cited on page 26.)

[Reeves & Nass 1996] Byron Reeves and Clifford Nass. How people treat computers, television, and new media like real people and places. CSLI Publications and Cambridge university press, 1996. (Cited on page 119.)

[Ross & Bagnell 2010] Stéphane Ross and Drew Bagnell. *Efficient Reductions for Imitation Learning.* In Yee Whye Teh and D. Mike Titterington, editeurs, AISTATS, volume 9 of *JMLR Proceedings*, pages 661–668. JMLR.org, 2010. (Cited on pages 27, 32 and 34.)

[Ross *et al.* 2011a] Stéphane Ross, Geoffrey J. Gordon and Drew Bagnell. *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning.* In Geoffrey J. Gordon, David B. Dunson and Miroslav Dudík, editeurs, AISTATS, volume 15 of *JMLR Proceedings*, pages 627–635. JMLR.org, 2011. (Cited on pages 32 and 35.)

[Ross *et al.* 2011b] Stéphane Ross, Joelle Pineau, Brahim Chaib-draa and Pierre Kreitmann. *A Bayesian Approach for Learning and Planning in Partially Observable Markov Decision Processes.* J. Mach. Learn. Res., vol. 12, pages 1729–1770, July 2011. (Cited on page 20.)

[Ross 2013] Stéphane Ross. *Interactive Learning for Sequential Decisions and Predictions.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, June 2013. (Cited on page 26.)

[Rummery & Niranjan 1994] G. A. Rummery and M. Niranjan. *On-Line Q-Learning Using Connectionist Systems.* Rapport technique, Cambridge University Engineering Department, 1994. (Cited on page 14.)

[Russell 1998] Stuart Russell. *Learning Agents for Uncertain Environments (Extended Abstract).* In Proceedings of the Eleventh Annual Conference on Computational Learning Theory, COLT' 98, pages 101–103, New York, NY, USA, 1998. ACM. (Cited on page 29.)

[Saxena *et al.* 2008] A. Saxena, J. Driemeyer and A.Y. Ng. *Robotic Grasping of Novel Objects using Vision.* International Journal of Robotics Research, 2008. (Cited on page 56.)

[Schaal 1996] Stefan Schaal. *Learning from Demonstration.* In Michael Mozer, Michael I. Jordan and Thomas Petsche, editeurs, NIPS, pages 1040–1046. MIT Press, 1996. (Cited on page 28.)

[Schaal 1999] Stefan Schaal. *Is imitation learning the route to humanoid robots?* Trends in cognitive sciences, vol. 3, no. 6, pages 233–242, 1999. (Cited on page 26.)

[Schwefel 1981] H.-P. Schwefel. Numerical optimization of computer models. John Wiley & Sons, New-York, 1981. 1995 – 2^{nd} edition. (Cited on pages 21 and 68.)

[Secretan *et al.* 2011] Jimmy Secretan, Nicholas Beato, David B. D'Ambrosio, Adelein Rodriguez, Adam Campbell, Jeremiah T. Folsom-Kovarik and Kenneth O. Stanley. *Picbreeder: A Case Study in Collaborative Evolutionary Exploration of Design Space.* Evolutionary Computation, vol. 19, no. 3, pages 373–403, 2011. (Cited on page 39.)

[Seo *et al.* 2000] Sambu Seo, Marko Wallat, Thore Graepel and Klaus Obermayer. *Gaussian Process Regression: Active Data Selection and Test Point Rejection.* In IJCNN (3), pages 241–246, 2000. (Cited on page 44.)

[Shepard 1957] R. N. Shepard. *Stimulus and response generalization: A stochastic model relating generalization to distance in psychological space.* Psychometrika, vol. 22, pages 325–345, 1957. (Cited on page 96.)

[Shivaswamy & Joachims 2012] Pannaga Shivaswamy and Thorsten Joachims. *Online Structured Prediction via Coactive Learning.* In Proc. ICML'12. icml.cc / Omnipress, 2012. (Cited on pages 50 and 93.)

[Siegel & Castellan 1988] Sidney Siegel and N. John Castellan. Nonparametric statistics for the behavioral sciences. McGraw–Hill, Inc., second édition, 1988. (Cited on pages 43 and 53.)

[Singh *et al.* 1998] Satinder Singh, Tommi Jaakkola, Michael L. Littman and Csaba Szepesvári. *Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms.* In MACHINE LEARNING, pages 287–308, 1998. (Cited on page 15.)

[Skinner 1951] B. F. Skinner. *How to Teach Animals.* Scientific American, vol. 185, pages 26–29, 1951. (Cited on page 122.)

[Snoek *et al.* 2012] J. Snoek, H. Larochelle and R. P. Adams. *Practical Bayesian Optimization of Machine Learning Algorithms.* In NIPS, pages 2960–2968, 2012. (Cited on page 92.)

[Sondik 1971] E. J. Sondik. *The Optimal Control of Partially Observable Markov Decision Processes.* PhD thesis, Stanford, California, 1971. (Cited on page 20.)

[Stanley *et al.* 2009] Kenneth O. Stanley, David B. D'Ambrosio and Jason Gauci. *A Hypercube-based Encoding for Evolving Large-scale Neural Networks.* Artif. Life, vol. 15, no. 2, pages 185–212, April 2009. (Cited on page 20.)

[Stern 1992] Hal Stern. *Are all linear paired comparison models empirically equivalent?* Mathematical Social Sciences, vol. 23, no. 1, pages 103 – 117, 1992. (Cited on page 43.)

[Stirling *et al.* 2010] T. S. Stirling, S. Wischmann and D. Floreano. *Energy-efficient indoor search by swarms of simulated flying robots without global information.* Swarm Intelligence, vol. 4, no. 2, pages 117–143, 2010. (Cited on page 57.)

[Strehl *et al.* 2006] A. L. Strehl, L. Li, E. Wiewiora, J. Langford and M. L. Littman. *PAC Model-free Reinforcement Learning.* In Proc. ICML, pages 881–888, 2006. (Cited on pages 56, 65, 68 and 72.)

[Stulp & Sigaud 2013] Freek Stulp and Olivier Sigaud. *Robot Skill Learning: From Reinforcement Learning to Evolution Strategies.* Paladyn. Journal of Behavioral Robotics, vol. 4, no. 1, pages 49–61, 2013. (Cited on pages 2 and 19.)

[Suga *et al.* 2005] Yuki Suga, Yoshinori Ikuma, Daisuke Nagao, Shigeki Sugano and Tetsuya Ogata. *Interactive evolution of human-robot communication in real*

world. In M. Meng et al., editeur, Proc. IEEE/RSJ IROS'05, pages 1438–1443. IEEE, 2005. (Cited on pages 39 and 41.)

[Sutton & Barto 1998a] R. Sutton and A. Barto. Reinforcement learning: An introduction. MIT Press, Cambridge, 1998. (Cited on page 1.)

[Sutton & Barto 1998b] Richard S. Sutton and Andrew G. Barto. Introduction to reinforcement learning. MIT Press, Cambridge, MA, USA, 1st édition, 1998. (Cited on pages 7 and 10.)

[Sutton *et al.* 1999] Richard S. Sutton, Doina Precup and Satinder Singh. *Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning*. Artif. Intell., vol. 112, no. 1-2, pages 181–211, August 1999. (Cited on page 25.)

[Sutton *et al.* 2009] Richard S. Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári and Eric Wiewiora. *Fast gradient-descent methods for temporal-difference learning with linear function approximation*. In Andrea Pohoreckyj Danyluk, Léon Bottou and Michael L. Littman, editeurs, ICML, volume 382 of *ACM International Conference Proceeding Series*, page 125. ACM, 2009. (Cited on pages 18 and 101.)

[Syed & Schapire 2007] Umar Syed and Robert E. Schapire. *A Game-Theoretic Approach to Apprenticeship Learning*. In NIPS, 2007. (Cited on page 31.)

[Symbrion 2009] Symbrion. *FP7 European Project FET IP 216342*, June 2009. (Cited on page 114.)

[Szepesvari 2010] Csaba Szepesvari. Algorithms for reinforcement learning. Morgan & Claypool, 2010. (Cited on page 1.)

[Takagi 2001] Hideyuki Takagi. *Interactive Evolutionary Computation: Fusion of the Capabilities of EC Optimization and Human Evaluation*. Proceedings of the IEEE, vol. 89, no. 9, pages 1275–1296, September 2001. Invited Paper. (Cited on page 39.)

[Tesauro 1995] Gerald Tesauro. *Temporal Difference Learning and TD-Gammon.* Commun. ACM, vol. 38, no. 3, pages 58–68, 1995. (Cited on pages 1, 17 and 20.)

[Tesauro 2007] Gerald Tesauro. *Reinforcement Learning in Autonomic Computing: A Manifesto and Case Studies.* IEEE Internet Computing, vol. 11, no. 1, pages 22–30, 2007. (Cited on page 1.)

[Thiéry 2010] Christophe Thiéry. *Itération sur les Politiques Optimiste et Apprentissage du Jeu de Tetris.* PhD thesis, Université e Henri Poincaré-Nancy 1, 2010. (Cited on page 13.)

[Thomaz & Breazeal 2006] Andrea L. Thomaz and Cynthia Breazeal. *Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance.* In Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1, AAAI'06, pages 1000–1005. AAAI Press, 2006. (Cited on pages 40 and 46.)

[Thorndike 1911] E. L. Thorndike. Animal intelligence. New York: Macmillan (Reprinted Bristol:Thoemmes, 1999), 1911. (Cited on page 1.)

[Trianni *et al.* 2006] V. Trianni, S. Nolfi and M. Dorigo. *Cooperative Hole Avoidance in a* Swarm-bot. Robotics and Autonomous Systems, vol. 54, no. 2, pages 97–103, 2006. (Cited on page 57.)

[Tsochantaridis *et al.* 2005] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann and Yasemin Altun. *Large Margin Methods for Structured and Interdependent Output Variables.* Journal of Machine Learning Research, vol. 6, pages 1453–1484, 2005. (Cited on page 77.)

[Vamplew *et al.* 2009] Peter Vamplew, Richard Dazeley, Ewan Barker and Andrei Kelarev. *Constructing Stochastic Mixture Policies for Episodic Multiobjective Reinforcement Learning Tasks.* In Australasian Conference on Artificial Intelligence, volume 5866 of *Lecture Notes in Computer Science*, pages 340–349. Springer Verlag, 2009. (Cited on page 24.)

[Viappiani & Boutilier 2010] Paolo Viappiani and Craig Boutilier. *Optimal Bayesian Recommendation Sets and Myopically Optimal Choice Query Sets.*

In NIPS, pages 2352–2360, 2010. (Cited on pages 5, 41, 44, 45, 75, 76, 77, 78, 80, 81, 93, 98 and 99.)

[Viappiani 2012] Paolo Viappiani. *Monte-Carlo Methods for Preference Learning.* In Y. Hamadi and M. Schoenauer, editeurs, Proc. Learning and Intelligent OptimizatioN, LION 6, pages 503–508. LNCS 7219, Springer Verlag, 2012. (Cited on pages 78 and 88.)

[Wasserstrom 1973] E. Wasserstrom. *Numerical solutions by the continuation method.* SIAM Review, vol. 15, no. 1, page 89–119, 1973. (Cited on pages 93 and 123.)

[Watkins & Dayan 1992] Christopher J. C. H. Watkins and Peter Dayan. *Q-learning.* Machine Learning, vol. 8, no. 3, pages 272–292, 1992. (Cited on page 15.)

[Watkins 1989] Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, UK, May 1989. (Cited on page 14.)

[Whiteson & Stone 2006] Shimon Whiteson and Peter Stone. *Evolutionary Function Approximation for Reinforcement Learning.* Journal of Machine Learning Research, vol. 7, pages 877–917, May 2006. (Cited on page 19.)

[Whiteson *et al.* 2010a] Shimon Whiteson, Matthew E. Taylor and Peter Stone. *Critical Factors in the Empirical Performance of Temporal Difference and Evolutionary Methods for Reinforcement Learning.* Journal of Autonomous Agents and Multi-Agent Systems, vol. 21, no. 1, pages 1–27, 2010. (Cited on pages 87 and 94.)

[Whiteson *et al.* 2010b] Shimon Whiteson, MatthewE. Taylor and Peter Stone. *Critical factors in the empirical performance of temporal difference and evolutionary methods for reinforcement learning.* Autonomous Agents and Multi-Agent Systems, vol. 21, no. 1, pages 1–35, 2010. (Cited on pages 19 and 20.)

[Wiering & van Otterlo 2012] M. Wiering and M. van Otterlo. Reinforcement learning: State-of-the-art. Adaptation, Learning, and Optimization. Springer, 2012. (Cited on pages 7 and 21.)

[Wilson *et al.* 2012] Aaron Wilson, Alan Fern and Prasad Tadepalli. *A Bayesian Approach for Policy Learning from Trajectory Preference Queries.* In NIPS, pages 1142–1150, 2012. (Cited on pages 5, 40, 49, 93, 96, 100, 102, 107, 109, 110, 111, 121, 122, 125 and 126.)

[Y. LeCun et al. 1989] Y. LeCun et al. *Handwritten Digit Recognition with a Back-Propagation Network.* In D. S. Touretzky, editeur, NIPS, pages 396–404. Morgan Kaufmann, 1989. (Cited on page 92.)

[Yue & Joachims 2009] Y. Yue and T. Joachims. *Interactively optimizing information retrieval systems as a dueling bandits problem.* In ICML, numéro 382:151 de ACM Int. Conf. Proc., 2009. (Cited on page 93.)

[Zhao *et al.* 2009] Zhao, M R Kosorok and D Zeng. *Reinforcement learning design for cancer clinical trials.* Stat Med, September 2009. (Cited on pages 48, 76, 83 and 85.)

[Ziebart *et al.* 2008] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell and Anind K. Dey. *Maximum Entropy Inverse Reinforcement Learning.* In AAAI, pages 1433–1438. AAAI Press, 2008. (Cited on pages 26, 30 and 32.)