Faculty of Science and
Bio-Engineering Sciences
Department of Computer Science
Theoretical Computer Science

Reinforcement Learning with Heuristic Information

Dissertation submitted in fulfilment of the requirements for the degree of Doctor of Science: Computer Science

Tim Brys

Promotors: Prof. Dr. Ann Nowé (Vrije Universiteit Brussel)
Prof. Dr. Matthew E. Taylor (Washington State University)

Print: Silhouet, Maldegem

2016 Uitgeverij VUBPRESS Brussels University Press
VUBPRESS is an imprint of ASP nv (Academic and Scientific Publishers nv)
Keizerslaan 34
B-1000 Brussels
Tel. +32 (0)2 289 26 50
Fax +32 (0)2 289 26 59
E-mail: info@vubpress.be
www.vubpress.be

ISBN 978 90 5718 448 2
NUR 958
Legal deposit D/2016/11.161/049

Soli Deo gloria

Abstract

Reinforcement learning is becoming increasingly popular in machine learning communities in academia and industry alike. Experimental successes in the past few years have hinted at the potential of reinforcement learning at tackling complex tasks, but the learning algorithms used typically still require impractically large amounts of training data. The main cause for this is a sparsity of rewards in a large state space. Because exploration of the environment is commonly driven by previously experienced rewards, initial exploration is usually too random to be practical. Two approaches to alleviating this problem are taken in the literature: creating more sample efficient algorithms, or introducing heuristic knowledge that aids the agent.

This thesis falls into the second category. Our first contribution is a non-exhaustive overview of the types of heuristic information sources commonly used in reinforcement learning, of the ways such information can be encoded and of the techniques that can be implemented to reuse this knowledge. The majority of our subsequent contributions concern the validation of various implementations of this blueprint.

The first is the investigation of the four knowledge injection techniques we identified in a variety of policy transfer settings. Policy transfer considers the case where behaviour learned in a previous task can be reused in the current task to be learned. We show the effects of high and low quality transferred policies on performance, the effect of policy transfer on multi-agent systems, and the way each of the techniques introduces bias in the learning process and how this relates to deterministic and stochastic transferred policies.

In the second such contribution, we investigate these knowledge injection techniques in the context of learning from demonstration, where examples of (expert) behaviour

are available to the learning agent. First we introduce a value-based encoding of such demonstrations which we then compare with an existing policy-based encoding. We analyse the effect that demonstration dataset size, quality, and demonstrator type has on the performance of learning agents using either encoding.

Our final and biggest contribution provides an answer to the question: what to do when multiple sources of heuristic knowledge are available. We developed a theoretically sound ensemble framework that allows for the inclusion of various pieces of heuristic knowledge through the learning in parallel of value functions, each initialized or shaped by one of the provided biases.

Acknowledgments

First of all, I am very grateful to my supervisor Ann Nowé, who made everything I did these past years possible (not *actually* everything of course...) by offering me a PhD position in her lab, and then funding[1] five years of travelling around the world in exchange for some research in between. My co-supervisor Matthew Taylor ranks second in thank-worthiness only because he has invested less money in me than Ann. Our scientific relationship was marked by two short intense bursts of productivity on opposite coasts of the States, from which most of the research described in this thesis flows. Thanks for those times Matt, and also for all the laughs!

Of course, I have to rank my supervisors first in my ranking-people-according-to-my-gratefulness charts. If they wouldn't notice though, I would of course put my sweet wife Naomi on top! I won't thank her for 'supporting me in the difficult times of my PhD' (I must have slept through those) or for 'carefully proofreading my thesis' (I would never do that to her!). While she did indeed do many things for me, I am simply grateful for who she is.

The next group of people I would like to express my gratefulness to would never figure on any charts drawn up by sensible humans. At the risk of losing my reputation, I thank Kristof, Maarten and Maurice. Kristof for the slapstick, the endless ping-pong matches and for being a frequent travel companion. Maarten not in the least for being my favourite bass-player, and Maurice for counter-balancing Kristof with some cynicism.

The chasing peloton in the race for my gratefulness contains many people who will remain unnamed (although not unthanked), including the vast majority of the AI lab's

[1] This research was financially supported by the Research Foundation Flanders (FWO).

members over the course of these five years. A special mention goes to Anna, Kyriakos, Mihail and The Anh.

Furthermore, I can't thank my parents enough for the massive amounts of sacrificial love they poured into my life. Their reward is not of this world.

And last, but certainly not least, I thank God for everything.

Contents

1 | Introduction

In the beginning...

Moses, *Genesis 1:1*

From the dawn of time, mankind has been ingeniously trying to make its life easier. We have always been building tools to simplify and automate the tasks we undertake, to offload as much work as possible: harnessing oxen and working metal ore to till more farmland than could ever be done with our bare hands, building intricate wheelwork mechanisms that harness the power of water and wind to grind more grain than we could eat, steam engines that accomplish the work of thousands of men, and now computers that think for us.

Artificial intelligence is the natural outflow of this age-old process of automation. Whereas until quite recently the work that was offloaded to machines has been of the sometimes refined, but usually dumb, physical form, in the past decades, automation has been breaching into what we perceive as the 'intelligent', automating not just 'work', but also decision making.

Whether people realize it or not, artificial intelligence (AI) is becoming more and more prevalent in our society. Perhaps not in the better known (or rather notorious) apocalyptic forms of Skynet or HAL 9000, but rather hidden behind the Google webpage that helps you with your questions, on the Amazon store that serves you relevant recommendations on books and movies, on your iPhone, making conversation under the name of Siri, or maybe even in your fancy car, assisting you with your below par parking skills.

And AI is quickly developing. Every year, the boundaries of what is possible are being pushed further and further. Since 2015, computers can play our video games from the

80's at a level comparable to that of an experienced gamer [Mnih et al., 2015]. In 2016, they first beat the world champion of Go, the holy grail of board games, pulling moves that are inhuman, but "so beautiful" (*dixit Fan Hui*, reigning European champion). Fully self-driving cars have been around for at least a couple of years [Google, 2016], and in a few more years Amazon drones will be whizzing around delivering packages to anybody and everybody [Amazon, 2016].

One of the strengths of many of these systems is their ability to learn from data. The rules they follow, the behaviour they exhibit, is not exclusively programmed by some smart engineer. Rather, the engineer implements a learning algorithm, which is then fed data relevant for the task at hand. The learning algorithm then finds patterns in the data, discovers what are 'good' decisions for which situations, and an 'intelligent' system emerges.

This is *Machine Learning*.

1.1 Reinforcement Learning

Some of the examples cited above use a specific Machine Learning approach called reinforcement learning. This approach to learning is inspired by behaviourist psychology, where human and animal behaviour is studied from a reward and punishment perspective. A small illustrative example conveys the main principle of this learning theory:

Example 1.1

Say you want to train your dog to sit.
You take your dog outside and shout 'sit'.
The dog realizes it needs to do something (you shouting and pointing to the ground is a definite clue), but it doesn't know what to do.
It barks, but nothing happens...
It gives a paw (something it learned before), but nothing happens...
It sits on the ground, and lo and behold, a dog cookie appears!

If you repeat this process many times, your dog will probably learn to associate the situation (you shouting 'sit') and its own action (sitting down), with the positive stimulus (a tasty cookie) and will repeat this behaviour on future occasions. In Thorndike's words:

> Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will

> be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond. [Thorndike, 1911]

He called this principle *the law of effect*.

In essence, the learner is considered to crave 'something' that it receives depending on its behaviour; it receives more of it when it exhibits desirable behaviour, and less (or even something opposite) when it does not. Whether this 'something' be cookies for a dog, or dopamine in the human brain, or a simple numerical value, an increase of it tells the learner that it has done something right, and an intelligent learner will repeat that behaviour when it encounters a similar situation in the future.

This same principle was successfully used in the examples cited above to train AIs to play video games and play Go.[1] The former using the score in the game as reward, the latter the win or loss as reward or punishment. One of the big downsides of this approach to learning is the often large number of experiences required to learn good behaviour, due to only sparsely given reward and a lack of prior knowledge. In Go for example, the learner only gets feedback at the very end of the game, with professional games taking on average about 200 moves. Therefore it is difficult for the learner to evaluate which moves were key factors in achieving the win or loss [Minsky, 1961]. In the video game setting, the generally successful AI we referred to completely fails on the specific game called 'Montezuma's revenge', which requires long sequences of quite specific actions before a score can be obtained (imagine giving your dog a cookie only after it has done the full sequence of sitting, a backflip, barking, sitting again, another bark, and a roll). A learner without prior knowledge has no chance[2] of stumbling on these sequences of actions.

A common way to overcome this problem is to not start learning from scratch, but to introduce some prior knowledge that guides the learner. For example, in the Go research we discussed above, before any reinforcement learning took place, the learning system was seeded with a huge database of human expert moves, an excellent example on how to use demonstration data to improve learning. Others have shown that using expert knowledge expressed as simple rules can be used to make a helicopter learn to fly a number of complex manoeuvres [Kim et al., 2004], or that using knowledge learned in previous tasks can help

[1] Do not dismiss these results as only academically interesting due to the 'game' nature of the problems: the complexity of these problems approaches and surpasses that of many more useful applications [Silver et al., 2016]. Furthermore, the past has shown that breakthrough advances in games have lead to breakthroughs in other fields. Monte Carlo Tree Search, initially developed for Go, is one example [Browne et al., 2012].

[2] To be fair, there is an infinitesimal probability, yet it is negligible for all practical purposes.

the current learning process [Taylor and Stone, 2009]. Many more examples like this exist, differing in where the prior information comes from, and how it is used.

1.2 Research Question and Contributions

The research question we investigate in this thesis is:

> **Definition 1.1: Research Question**
>
> How can one incorporate prior or external knowledge in a temporal difference reinforcement learning process, aiming to increase the sample efficiency of this process?

The contributions described in this thesis answer this question in several ways. Our first contribution is a unified view of the different ways of including prior knowledge encoded as a value function or policy in a temporal difference process (Chapter 3). Our next contribution is the evaluation of these different ways in the problem of policy transfer, i.e. reusing behaviour learned in a previous, similar task (Chapter 4). Our third contribution is a novel Learning from Demonstration technique, which is then also evaluated using the different ways of incorporating the demonstration knowledge, and extensively compared with the state-of-the-art in a variety of settings (Chapter 5). The last major contribution we describe in this thesis is the introduction of heuristic ensembles in reinforcement learning, i.e. the semi-automatic combination of various pieces of prior knowledge (Chapter 6).

But before we describe these contributions in detail, we give a more formal definition of reinforcement learning and its related concepts and notation required for the reading of this thesis.

2 | Reinforcement Learning

There is a reward for your work.

God, *Jeremiah 31:16*

In this chapter, we provide some background on reinforcement learning, the learning theory within which we make our contributions in subsequent chapters. We only elaborate on the concepts and notation used in this thesis. For a broader treatment of the reinforcement learning framework, we refer the reader to Sutton and Barto's seminal book [Sutton and Barto, 1998], and a more recent overview of the state-of-the-art [Wiering and Van Otterlo, 2012].

2.1 The Reinforcement Learning Problem

We set the stage with the classic reinforcement learning (RL) diagram, displayed in Figure 2.1. It shows how an RL *agent* interacts with its *environment*. First, to say what an agent exactly is, is surprisingly difficult; definitions abound in AI literature. In this thesis, we adopt the following simple definition [Russell and Norvig, 1995]:

> **Definition 2.1: An agent**
>
> An agent is just something that perceives and acts.

What the agent perceives and acts upon, we call the environment. This environment typically changes due to the agent's actions and possibly other factors outside the agent's

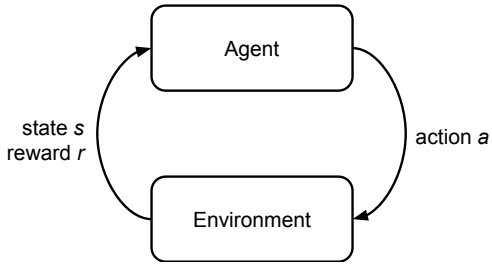


Figure 2.1: The reinforcement learning agent-environment interaction loop.

influence. The agent perceives the state (s) of the environment (a potentially incomplete observation), and must decide which action (a) to take based on that information, such that the accumulation of rewards (r) it receives from the environment is maximized. These interactions occur at discrete subsequent time steps.

This agent-environment interaction process is most commonly formulated as a Markov Decision Process (MDP):

Definition 2.2: Markov Decision Process

A Markov Decision Process is a tuple $\langle S, A, T, \gamma, R \rangle$.

- $S = \{s_1, s_2, \ldots\}$ is the possibly infinite set of states the environment can be in.
- $A = \{a_1, a_2, \ldots\}$ is the possibly infinite set of actions the agent can take.
- $T(s'|s, a)$ defines the probability of ending up in environment state s' after taking action a in state s.
- $\gamma \in [0, 1]$ is the discount factor, which defines how important future rewards are.
- $R(s, a, s')$ is the possibly stochastic reward given for a state transition from s to s' through taking action a. It defines the goal of an agent interacting with the MDP, as it indicates the immediate quality of what the agent is doing.

It is called a *Markov* Decision Process, because the state signal is assumed to have the Markov property:

Definition 2.3: Markov Property

A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it. [Bhattacharya and Waymire, 2007]

In other words, the state signal should contain enough information to reliably predict future states.

The way an agent acts based on its perceptions, i.e., its behaviour, is commonly referred to as *a policy*, denoted as $\pi : S \times A \rightarrow [0, 1]$. It formally describes how likely an agent is to do something (action) in a given situation (state), by mapping state-action pairs to action selection probabilities. In this thesis, we use this notation for policies interchangeably with the following notation $\pi : S \rightarrow A$, which is a reformulation where not the action selection probabilities are output for a given state-action pair, but given a state and these probabilities, $\pi(s)$ outputs a probabilistically selected action.

The goal of an agent interacting with an MDP is to learn behaviour, a policy, that maximizes the discounted accumulation of rewards collected during its lifetime in the environment. This accumulation of reward up to a given time horizon or into infinity is called the return:

$$\begin{aligned}\mathcal{R}_t &= R(s_t, a_t, s_{t+1}) + \gamma R(s_{t+1}, a_{t+1}, s_{t+2}) + \gamma^2 R(s_{t+2}, a_{t+2}, s_{t+3}) + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1})\end{aligned}$$

The discount factor γ determines the current value of future rewards. As $\gamma \rightarrow 1$, the agent becomes more farsighted, and will prefer large future rewards over smaller short-term rewards.

Given a state s and a policy π, we can express the return an agent can expect when starting from that state and following that policy as follows:

$$V^{\pi}(s) = E\left\{\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1}) | s_t = s\right\}$$

This *value function* expresses the quality of being in state s when following policy π, given the MDP to-be-solved that generates state transitions and rewards for these transitions. The expectation $E\{\}$ accounts for the stochasticity in these transition and reward functions, as well as in the policy that generates the action sequence.

Similarly, we can define the quality of being in state s, taking action a, and subsequently following policy π. This is called the action-value function:

$$Q^{\pi}(s,a) = E\left\{\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1}) | s_t = s, a_t = a\right\}$$

The expected returns encoded in these value functions yield a way to evaluate the quality of policies. A policy π is better than another policy π' if it has higher expected returns. A reinforcement learning agent needs to learn a policy that maximizes the expected return:

$$\forall s \in S, \forall a \in A : \pi^* = \arg\max_{\pi} Q^{\pi}(s,a)$$

π^* is called an optimal policy,[1] as it represents the behaviour that gets the highest return in expectation for all states, thus solving the task encoded in the reward function.

2.2 Reinforcement Learning Algorithms

If the MDP's transition and reward functions are known, Dynamic Programming techniques can be used to optimally solve the problem [Bertsekas, 1995]. Yet, it is uncommon to have a full specification of a system's dynamics or the reward function, and thus the use of techniques that can work with only knowledge of state and action spaces is necessary. These techniques must generate policies that maximize the expected return in environments with unknown dynamics and goals through trial-and-error. Learning a model of the environment may be part of this process, but it is not necessary and many techniques are successful without this component.

The learning algorithms used in this paper are of this 'model-free' type. More specifically, we focus on temporal difference (TD) learning algorithms, because they are still one of the most popular and successful classes of RL algorithms, notwithstanding the success of for example policy gradient algorithms [Sutton et al., 1999; Degris et al., 2012; Silver et al., 2016].

Definition 2.4: TD learning

Temporal difference learning is an approach to reinforcement learning that keeps estimates of expected return and updates these estimates based on experiences and differences in estimates over successive time-steps.

[1] There is at least one [Puterman, 2014], but there may be many (deterministic or stochastic). Still, all their (action-)value functions will be the same.

In other words, in TD learning, the agent incrementally updates estimates of a value function, using observed rewards and the previous estimates of that value function. One of the best known and simplest temporal difference learning algorithms is Q-learning [Watkins, 1989]. It estimates the optimal Q-function Q^* by iteratively updating its estimates $\hat{Q}$ after each (s, a, r, s') interaction with the environment:

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha\delta$$

$0 \leq \alpha \leq 1$ is the step size, controlling how much the value function is updated in the direction of the temporal difference error δ. The temporal difference error δ is the difference between the previous estimate and the observed sample:

$$\delta = r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)$$

Q-learning performs *off-policy* learning. This means that it learns about a different policy than the one generating the interactions with the environment, which is called the behaviour policy. In the case of Q-learning, the policy being learned about is the optimal policy.

The *on-policy* variant of Q-learning is called SARSA. It modifies the temporal difference error in such a way that the algorithm learns about the behaviour policy, using the action a' actually executed in next state s', instead of using the action with the highest estimate in that state:

$$\delta = r + \gamma\hat{Q}(s',a') - \hat{Q}(s,a)$$

If all state-action pairs are visited infinitely often, given some boundary conditions, Q-learning and SARSA are guaranteed to converge to the true Q-values [Tsitsiklis, 1994; Singh et al., 2000]. In practice, a finite number of experiences is usually sufficient to generate near-optimal behaviour.

From an estimated Q-function, an agent can easily derive a greedy deterministic policy π:

$$\pi(s) = \arg\max_{a} \hat{Q}(s,a)$$

If the estimates have converged to the optimal Q-values Q^*, then this formula generates an optimal policy.

Since an agent typically needs to sufficiently explore the state-action space in order to find optimal behaviour (infinitely often in the case of Q-learning and SARSA), it is in most cases insufficient during learning to just execute the greedy policy derived from the agent's estimates to generate interactions with the environment. That is because with the greedy policy, actions that are initially underestimated will likely never be executed again, nor will

their estimates be updated again, because the greedy policy always executes the action with the highest estimated return. This results in the agent ceasing exploration prematurely, and the value function converging to a suboptimal solution. Instead of always using the greedy policy with respect to the estimates to select actions, it is therefore often advisable to inject stochasticity into the policy to generate the necessary exploration.

One way is to take a random action at every time-step with probability ϵ. This ensures that every reachable state-action pair has a non-zero visitation probability, irrespective of the estimated Q-values at that time. Let $\xi \in [0, 1]$ be a randomly drawn real number, an ϵ-*greedy* exploration policy is given by:

$$\pi(s) = \begin{cases} \text{a random action} & \textit{if } \xi < \epsilon \\ \arg\max_a \hat{Q}(s, a) & \textit{otherwise} \end{cases}$$

Another popular approach is softmax action selection, which determines the probability of every action based on the relative magnitude of the actions' estimates:

$$\pi(s, a) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a'} e^{\frac{Q(s,a')}{\tau}}}$$

The 'temperature' parameter τ determines how random (high τ) or greedy (low τ) action selection is. Actions with higher estimated Q-values will have relatively higher probabilities of being selected, and actions with lower estimated Q-values will have proportionally lower probabilities.

2.2.1 Function Approximation and Eligibility Traces

The basic versions of the algorithms described above are defined for discrete state-action spaces. They use a simple table to store the Q estimates: one entry for every possible state-action pair. Since many practical reinforcement learning problems have very large and/or continuous state spaces,[2] basic tabular learning methods are impractical, due to the sheer size of storage required, or even unusable, due to a table's inherent inability to faithfully represent continuous spaces. Therefore, function approximation techniques are required to render the learning problem tractable. Many different approximators exist, with deep neural networks being currently very much in vogue [Mnih et al., 2015; Silver et al., 2016]. We have found a simpler and more traditional approach to be sufficient in the experiments performed for this thesis. We consider tile-coding function approximation [Albus, 1981], a linear approximator which overlays the state space with multiple randomly-offset, axis-parallel tilings. See Figure 2.2 for an illustration. This allows for a discretization of the

[2] Not to speak of continuous *action* spaces. That is not considered in this thesis.

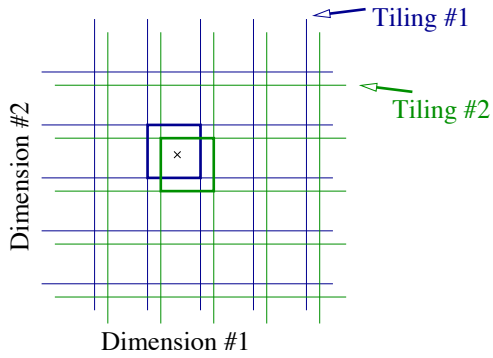


Figure 2.2: An illustration of tile-coding function approximation on a two-dimensional state space, with two tilings. The indicated cells indicate the active tiles. Figure taken from [Taylor, 2008].

state-space, while the overlapping tilings guarantee a certain degree of generalization. The Q-function can be approximated by learning weights that map the tiles activated by the current state s and action a to an estimated Q-value:

$$\hat{Q}(s,a) = \theta^T \phi(s,a)$$

$\phi(s,a)$ is the feature vector representing state-action pair (s,a), i.e., a binary vector indicating the tiles activated by this state and the action, and θ is the weight vector that needs to be learned to approximate the actual Q-function. This weight vector is updated using an update-rule similar to the one used in the tabular case:[3]

$$\theta \leftarrow \theta + \alpha\delta$$

Besides function approximation, a last mechanism we use to construct a reinforcement learning agent to conduct experiments with in this thesis is called eligibility traces. Eligibility traces [Klopf, 1972] are records of past occurrences of state-action pairs. These give a sense of how 'long ago' a given action was taken in a given state. They can be used to propagate reward further into the past (n-step) than the algorithms discussed until now do (one step). Using eligibility traces, not only the Q-value of the currently observed state-action pair is updated, but also those of past state-action pairs, inversely

[3]Note that with function approximation, Q-learning is no longer guaranteed to converge and may even diverge because the Markov property is violated [Baird, 1995]. In practice, this is not typically problematic.

proportional to the time since they were experienced. Concretely, a (replacing) eligibility trace $e(s, a)$ for state s and action a is updated as follows [Singh and Sutton, 1996]:

$$e(s,a) \leftarrow \begin{cases} 1 & s = s_t, a = a_t \\ \gamma\lambda e(s,a) & \textit{otherwise} \end{cases}$$

It is set to 1 if (s, a) is currently observed, and otherwise it is decayed by $\gamma\lambda$, with $0 \leq \lambda \leq 1$ the eligibility trace decay parameter. Higher λ results in rewards being propagated further into the past. This eligibility trace update is performed at every step, thus making traces decay over time. The eligibility traces are included as a vector e in the Q update rule as follows:

$$\theta \leftarrow \theta + \alpha e \delta$$

2.2.2 $Q(\lambda)$-Learning with Tile-Coding

All of the literature described above amounts to the reinforcement learning algorithm we will be using throughout this thesis: $Q(\lambda)$-learning with tile-coding function approximation. It performs off-policy learning, which is necessary for the ensemble learning in Chapter 6. It uses function approximation to generalize experiences and deal with continuous state-spaces, which is necessary for most of the benchmark problems we consider (in Mario we use tabular learning – we argue why when discussing that benchmark problem below). And it uses eligibility traces to speed up the propagation of rewards through the agent's representation of the value function and thus to speed up convergence.

2.3 Sample Efficiency

As in general machine learning, *sample efficiency* is important in reinforcement learning. Sample efficiency represents the number of environment (s, a, r, s') samples an agent requires to perform a task well. Obtaining samples usually carries a cost, often greater than just the computational cost associated with processing the sample. Making a robot spend hours, days and weeks to learn a task is very costly. It takes a lot of electricity, several engineers to attend to the robot, and physical space for the robot to execute the task, none of which are cheap to obtain.

Therefore, one of the primary goals of reinforcement learning algorithms, besides convergence and (near-) optimality, is an efficient use of samples. The fewer samples an algorithm requires to achieve some desirable level of behaviour, the better. Broadly speaking, researchers take either one of two approaches to reduce the number of samples an agent

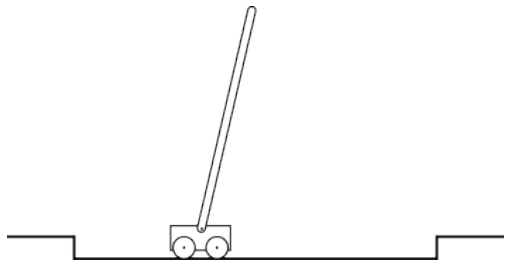


Figure 2.3: An illustration of the Cart Pole problem.

requires. They either build algorithms and techniques that inherently require fewer samples, or they use some prior/external knowledge to bias the agent.[4] Some argue that the former is superior to the latter, as it is the more general approach [Sutton, 2016]. Yet, we believe that both will always be intertwined. One can see this for example in the great success of AlphaGo, which definitely is a great example of new algorithms using the provided samples in a better way, yet still it required a great deal of human demonstrations to work well.

This thesis falls into the second category, and investigates the inclusion of prior and external knowledge to reduce the number of samples an agent requires to achieve good behaviour.

2.4 Benchmark Problems

In order to experimentally compare reinforcement learning algorithms, a number of benchmark problems have been created and have seen extensive use in the literature. Below, we describe the problems we consider in this thesis. These represent a diverse set of problems, with very different environment dynamics, state and action spaces.[5]

2.4.1 Cart Pole

Cart Pole [Michie and Chambers, 1968] is a task in which the learning agent controls a cart with a pole on top, see Figure 2.3. The goal is to keep the pole balanced for as long

[4] Throughout this thesis, when we talk about biasing exploration, biasing an agent, etc., we use the word in the sense that guidance is introduced, that the agent is made to prefer some actions over others when acting in the environment. Not that the learner can not learn the target policy any more.

[5] It is hard to exactly quantify how different tasks are. Recent work on similarity metrics between MDPs makes some steps towards answering that question [Ammar et al., 2014; Song et al., 2016], although limited to MDPs of the same domain (e.g., different versions of Cart Pole), and fully known MDPs respectively.

as possible by moving the cart left and right within a given interval in a single dimension. The state space consists of the position of the cart, its velocity, the angle of the pole and its angular velocity $(x, \dot{x}, \theta, \dot{\theta})$. The agent receives step rewards of 0, and a final reward of -1 when the pole falls. In the experiments, performance is measured as the number of steps the pole is balanced, and a learning episode is limited to 1000 steps.

Definition 2.5: Cart Pole Agent

State space 4 variables:

1. x position of the cart, $((-2.4, 2.4))$
2. $\dot{x}$ velocity of the cart $((-6.0, 6.0))$
3. θ angle of the pole $((-0.21, 0.21))$
4. $\dot{\theta}$ angular velocity of the pole $((-6.0, 6.0))$

Action space $\{\text{left}, \text{right}\}$

Reward Step reward of 0, -1 when the pole falls

Learning parameters $\alpha = \frac{0.25}{16}$,[a] $\gamma = 1$, $\epsilon = 0.05$, $\lambda = 0.25$, 16 randomly offset tilings of $10 \times 10 \times 10 \times 10$

[a] Here, as in the Pursuit Domain, we divide the learning rate by the number of tilings to ensure an effective $\alpha < 1$.

2.4.2 Pursuit Domain

The Pursuit domain, or Predator/Prey, was proposed by Benda et al. [1986] to investigate coordination mechanisms in a multi-agent system. The basic idea of pursuit is that a number of predators must capture a (number of) prey(s) by moving through a simple gridworld, see Figure 2.4. Stone and Veloso [2000] identify many variants of the problem and our implementation is as follows. There are two predators and one prey, and these can move in the four cardinal directions in a non-toroidal grid of 20×20 cells, as well as choose to stay in place. The prey is caught when a predator moves onto the same gridworld cell as the prey, and predators are not allowed to share the same cell. The prey takes a random action 20% of the time, with the rest of the time devoted to moving away from the predators. To do that, it takes the action that maximizes the summed distance from both predators, making the problem harder than with a fully random prey. The two predators are learning agents, and both receive a reward of 1 when the prey is caught by

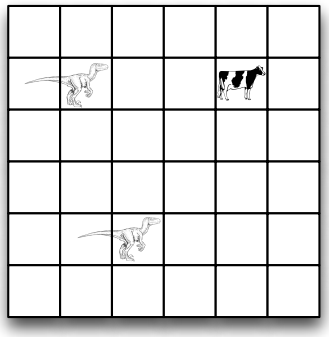


Figure 2.4: An illustration of the Pursuit domain.

either one of them, and a reward of 0 the rest of the time; performance is measured in our experiments as the number of steps needed to catch the prey. The predators observe the relative x and y coordinates of the other predator and the prey.

Definition 2.6: Pursuit Agents

State space 4 variables:

- **1-2** relative x and y position of the other predator ($(-19, 19)$)
- **3-4** relative x and y position of the prey ($(-19, 19)$)

Action space $\{\text{north}, \text{east}, \text{south}, \text{west}, \text{no move}\}$

Reward function step reward of 0, 1 for both predators when either catches the prey

Learning parameters $\alpha = \frac{1}{10\times 32}$, $\gamma = 0.9$, $\epsilon = 0.1$, $\lambda = 0.9$, 32 tilings with tile-width 10.[a]

[a] Tile-width is measured in whatever units each of the state variables is measured.

2.4.3 Super Mario

The Mario benchmark problem [Karakovskiy and Togelius, 2012] is based on Infinite Mario Bros, which is a public reimplementation of the original 80's game Super Mario Bros®.

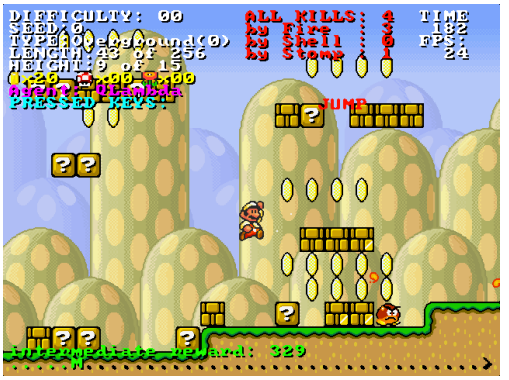


Figure 2.5: A screenshot of Mario in a random level.

See Figure 2.5 for a screenshot of the Mario game. In this task, Mario needs to collect as many points as possible, which are attributed for killing an enemy (10), devouring a mushroom (58) or a fireflower (64), grabbing a coin (16), finding a hidden block (24), finishing the level (1024), getting hurt by a creature (-42) or dying (-512). The actions available to Mario correspond to the buttons on the NES controller, which are (left, right, no direction), (jump, don't jump), and (run/fire, don't run/fire). One action from each of these groups can be taken simultaneously, resulting in 12 distinct combined or 'super' actions. The state space in Mario is quite complex, as Mario observes the exact locations of all enemies on the screen and their type, he observes all information pertaining to himself, such as what mode he is in (small, big, fire), and furthermore he is surrounded by a gridlike receptive field in which each cell indicates what type of object is in it (such as a brick, a coin, a mushroom, a goomba (enemy), etc.).

Our reinforcement learning agent for Mario is inspired by Liao et al. [2012]. The state space of the agent consists of 27 discrete variables (see below) that make for 3.65×10^{10} different states, and $3.65 \times 10^{10} \times 12 = 4.38 \times 10^{11}$ Q-values. Even though this is a huge state-space, it does not pose a problem computationally, because the visitation pattern of states follows an exponentially decaying curve: a small set of states receives the majority of visits [Liao et al., 2012]. This is illustrated in Figure 2.6. Therefore, we opt for a simple tabular representation of the value function like Liao et al. [2012], even though it is possible that introducing generalization between states would be beneficial.

In the experiments, every learning episode is run on a procedurally generated level based on a random seed $\in [0, 10^6]$, with difficulty 0. The mode Mario starts in (small, large, fire) is randomly selected for each episode. Making an agent learn to play Mario this way helps avoid overfitting on a specific level, and makes for a more generally applicable Mario

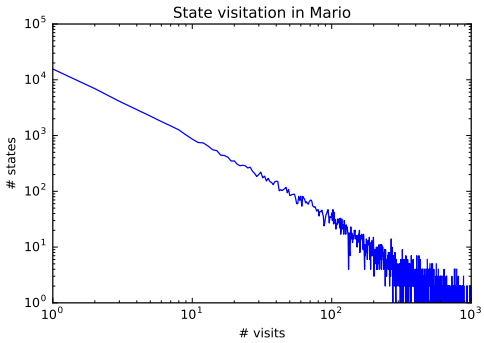


Figure 2.6: The state visitation pattern in Mario for $Q(\lambda)$-learning during the first 1000 episodes, learning from scratch. Log-scale. A large number of states has a small number of visits and vice versa.

agent. This fits the 'generalized environment' concept of Whiteson et al. [2011], indeed intended to avoid environment overfitting (in this case, that means level-overfitting).

Definition 2.7: Mario Agent

State space 27 variables:

- **1** is Mario able to jump? (boolean)
- **2** is Mario on the ground? (boolean)
- **3** is Mario able to shoot fireballs? (boolean)
- **4-5** Mario's direction in the horizontal and vertical planes ($\{-1, 0, 1\}$)
- **6-9** is there an obstacle in one of the four vertical grid cells in front of Mario? (boolean)
- **10-17** is there an enemy within one grid cell removed from Mario in one of eight different directions (left, up-left, up, up-right, etc.) (boolean)
- **18-25** as the previous, but for enemies within two to three grid cells. (boolean)
- **26-27** the relative horizontal and vertical positions of the closest enemy ($(-10, 10)$, measured in grid cells, plus one value indicating an absence of enemies)

Action space 12 actions, taking one element from each of these sets: $\{\text{left}, \text{right}, \text{no direction}\}$, $\{\text{jump}, \text{don't jump}\}$, and $\{\text{run/fire}, \text{don't run/fire}\}$

Reward function points collected in the Super Mario game are attributed as reward

Learning parameters $\alpha = 0.001$, $\gamma = 0.9$, $\epsilon = 0.05$ and $\lambda = 0.5$, with tabular learning

2.5 Summary

The paradigm of reinforcement learning frames learning as the adaptation of behaviour in order to achieve goals defined by means of a feedback signal. This signal rewards and punishes the learner for the behaviour it exhibits, and a learning agent seeks to accumulate as much of these rewards as possible. The agent achieves this by observing the effect its behaviour has on the situation it is in and the rewards associated with it (short and long-term), and adjusting its behaviour accordingly.

We described these concepts in the conventional mathematical frameworks and discussed reinforcement learning algorithms and their components only to the extent necessary for understanding this thesis. The same applies to the reinforcement learning benchmark problems described.

In the following chapter, we discuss ways of incorporating prior or external knowledge into a reinforcement learning process. Our first contribution can be found in that chapter, in a unified view of the different existing ways of injecting knowledge encoded as policies or value functions.

3 | Incorporating Prior or External Knowledge

> The way of a fool is right in his own eyes, but a wise man listens to advice.
>
> King Solomon, *Proverbs 12:15*
>
> "I have the right to do anything," you say – but not everything is beneficial. "I have the right to do anything" – but not everything is constructive.
>
> St. Paul, *1 Corinthians 10:23*

We reiterate that the focus of this thesis lies on the incorporation of prior or external knowledge in a reinforcement learning process, aiming to increase the sample efficiency of this process. Or, in basic terms: how can we make an agent learn faster using some information that is available? This raises a number of questions:

1. Where does the information come from?
2. What type of information is considered?
3. How do we encode this information?
4. Given the encoding, how do we inject it into the RL process?

We will address these four questions in three separate sections in this chapter. In the first section we discuss the types of knowledge that we and others have considered to aid a learning agent and where they may come from. In the second section, we look at two general ways of encoding such knowledge in a way that is usable by an RL agent. And finally we identify different approaches that can be used to inject such encoded knowledge into the RL process.

3.1 Type of Knowledge

The source of knowledge (in the sense of who/what provides the knowledge to the learning agent) will always be an agent. Whether it be the same agent as is currently trying to learn a task by using some previously learned information, a human providing some expert knowledge he has distilled from a description of the task to be learned, or another AI agent that demonstrates how he would solve the task – an agent it is.

A more important factor to consider is the type of information, i.e., what the information describes. Below we will discuss various popular such types of information that have been proposed as useful to increase an RL agent's sample efficiency.

3.1.1 Transfer Learning

The type of knowledge considered in transfer learning is knowledge previously learned in a different task, by the agent in question or a different agent [Taylor and Stone, 2009]. The idea is that knowledge learned about a previous task could provide a useful prior in the current task, given some similarities between the tasks. Where tasks differ, inter-task mappings may need to be defined for the transferred information to be useful [Taylor et al., 2007], although these mappings may also be learnable [Bou Ammar et al., 2013]. Since its introduction in RL, transfer learning has known great successes, and a wide range of techniques now exists, the techniques differing in what exactly is transferred to the new task. A few examples:

- low-level knowledge, such as the Q-function [Selfridge et al., 1985]
- a black-box policy, queryable by the agent [Fernández and Veloso, 2006]
- a model of the task [Atkeson and Santamaria, 1997]
- (s, a, r, s') samples for batch RL [Lazaric, 2008]

In one sense, all the techniques that we describe in this chapter are transfer learning techniques, because they are all about transferring knowledge between two agents. In this thesis, we use the stricter definition that transfer learning is about transferring knowledge

specifically obtained in a previous, related task. Chapter 4 is specifically dedicated to the investigation of the case where an agent's policy is transferred as a black-box.

3.1.2 Demonstrations

Demonstrations are examples of behaviour generated by some other agent, usually in the form of sequences of state-action pairs $((s_1, a_1), \ldots, (s_n, a_n))$. Compared to transfer learning, this is a more generally applicable type of knowledge exchange, as it relies less on the internals of the source agent being available and compatible. For example, a human can not copy the value function it has in its brain for flipping a pancake, if any, to the housekeeping robot it wants to perform that task. Demonstrating samples of the desired behaviour can then be a useful way to still get a significant portion of this information across.

The demonstrated samples can be used for initial passive learning, making the learner process these samples as if it were really generating these experiences itself [Schaal, 1997; Smart and Kaelbling, 2002], or by processing them first using a Learning from Demonstration technique [Taylor et al., 2011b; Brys et al., 2015a] and subsequently 'transferring' the obtained value function or policy. Another possible avenue is inferring a reward function that explains the demonstrations, and using that to learn [Russell, 1998]. Such a learned reward function can then even be combined with the environment's reward signal [Suay et al., 2016] to increase sample efficiency.

In Chapter 5, we perform an in-depth investigation of two techniques that integrate demonstrations with an RL process.

3.1.3 Off-line Advice

Commonly, there exist some (human) expert agents with useful exact or heuristic knowledge pertaining to the task the learning agent attempts to solve. This information could be given to the agent during learning, as we will see in the next section, but it can be even more useful to hand the agent this knowledge before learning starts, i.e., off-line. The agent can then use this information from the start. Many examples exist of rules-of-thumb, general guidelines or advice for specific situations being useful biases for an RL process:

- encouraging player role diversification in RoboCup Soccer [Devlin et al., 2011]
- advising the use of human strategies in StarCraft [Efthymiadis and Kudenko, 2013]
- indicating the general direction to move in to super Mario [Brys et al., 2014a]

In Chapter 6 we investigate how several such heuristics can be combined to maximize the increase in sample efficiency.

3.1.4 On-line Advice/Feedback

Knowledge from related tasks and demonstrations is usually obtained off-line, before learning in the task under consideration is initiated. But, the agent can also be fed useful information during learning, information that was not available beforehand, through qualitative feedback on its behaviour, or actual advice from another agent. With the TAMER+RL framework, Knox and Stone show that an RL agent can greatly benefit from the positive and negative feedback of a human trainer given during learning, by learning a model of the human's hypothetical internal value function and learning from that or combining it with the environment's rewards [Knox and Stone, 2010]. Loftin et al. [2016] show that taking into account the feedback strategy of the trainer can yield even better results.

A trainer agent can also more directly influence the learning agent by providing actual action advice during learning. A 'good' trainer attempts to intelligently advise the agent about good actions by concentrating advice on critical situations or situations where he predicts the student will make mistakes [Taylor et al., 2014].

3.2 Encoding Knowledge

The agent providing the knowledge already has this knowledge encoded in some form of representation. For successful knowledge transfer, this knowledge must be encoded in such a form that the RL agent on the receiving end understands it and can use it. Here we list some concepts an RL agent understands that can be used to transfer knowledge:

- state-action(-reward-state) samples
- a value function
- a policy
- a reward function
- a transition function

In this thesis, we are concerned with only two types of encoding: a value function and a policy. We choose these because they allow for a straightforward encoding of full behaviour that is immediately usable by a temporal-difference learning agent. The other concepts mentioned above require processing or extra information before they could be used as such. In Chapter 4, we consider *policy* transfer, in Chapter 5, we consider two Learning from Demonstration techniques that respectively output a *policy* and a *value function*, and in the final contribution chapter, we consider *value functions* defined by a human domain expert.

It is important to note that a policy can be derived from a state-action value function (Q), and that a pseudo state-action value function can be constructed from a policy. For

example, a greedy deterministic policy can be derived from a Q-function:

$$\pi(s) = \arg\max_a Q(s, a)$$

In the reverse direction, a pseudo value function can be generated from a policy by using the policy's action selection probabilities to give an indication of the quality of state-action pairs [Brys et al., 2015b]:

$$Q(s, a) = \pi(s, a)$$

Of course, this is not a true value function for the given MDP (hence the pseudo), as it does not depend on the MDP's reward and transition functions. Since these are usually (partially) unknown at the time of encoding this knowledge, we simply can not construct the true value function for the given policy a priori. Therefore, we consider this straightforward encoding of a policy as a 'value-function' in the absence of known reward and transition functions. In reward shaping literature, such a value function would be called a potential function [Ng et al., 1999; Wiewiora et al., 2003].

Furthermore, note that in the case of function approximation, we consider the value function defined over state-action features $\phi(s, a)$ and not over the raw (s, a). This is a requirement to maintain convergence guarantees in reward shaping and initialization.[1]

3.3 Injecting Knowledge

Given information contained in a policy or value function, we must consider how to include this in the learning process. We discuss four common ways that we find in the literature and that we will use in this thesis:

- Q-value initialization
- Reward shaping
- Probabilistic Policy Reuse
- Extra action

We briefly review the relevant theory and discuss the benefits and drawbacks of each approach. Note that we do not provide an exhaustive list of all possible ways of incorporating a value function or policy, we simply describe a diverse and representative subset.

[1] Personal communication with Anna Harutyunyan.

3.3.1 Q-function Initialization

Input: V or Q

With Q-function initialization, the RL agent's estimate of the Q-function is seeded with the given input values Q_{input}: $\hat{Q} = Q_{\text{input}}$. When no prior information is available, Q_{input} is typically set to random values or uniformly to 0. It can also be optimistically set to the highest possible reward in the environment to encourage uniform exploration of the state-action space [Kaelbling et al., 1996]. On the other hand, when a prior quality-assessment on state-action pairs is available, this can be used to provide the agent with a useful initial value function. The principle is simple: attribute higher values to a state-action pair if that action is deemed desirable in that state, and lower values if the reverse is true. If the agent derives its behaviour policy from its estimated values, it will then start out following a policy that largely follows the given advice.

Q-function initialization is straightforward in the basic case of a tabular representation, as one can actually enumerate every (s, a)-pair in the $\hat{Q}$-table and store the corresponding $Q_{\text{input}}(s, a)$-value. In the case of function approximation, even for the simplest forms such as linear tile coding, it is usually unclear how to directly initialize the weights θ. This problem is avoided by using techniques such as Q-value reuse [Taylor et al., 2007] or potential-based reward shaping, given the proven equivalence with Q-value initialization, see below.

3.3.2 Reward Shaping

Input: V or Q

The modern version of reward shaping, a technique with roots in behavioural psychology [Skinner, 1938], provides a learning agent with extra intermediate rewards, much like a dog trainer would reward a dog for completing part of a task. This extra reward can enrich a sparse base reward signal (for example a signal that only gives a non-zero feedback when the agent reaches the goal), providing the agent with useful gradient information. This shaping reward F is added to the environment's reward R to create a new composite reward signal that the agent uses for learning:

$$R_F(s, a, s') = R(s, a, s') + F(s, a, s')$$

Of course, since the reward function defines the task, modifying the reward function may actually modify the task, changing the total order over policies, and making the agent converge to suboptimal policies (with respect to the environment's original reward).

If we define a potential function $\Phi : S \to \mathbb{R}$ over the state space, and take F as the difference between the new and old states' potential, Ng et al. [1999] proved that the total

order over policies remains unchanged, and convergence guarantees are preserved:

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) \tag{3.1}$$

Prior knowledge can be incorporated by defining the potential function Φ accordingly.

The definition of F and Φ was extended by others ([Wiewiora et al., 2003; Devlin and Kudenko, 2012; Harutyunyan et al., 2015b]) to include actions and timesteps, allowing for the incorporation of behavioural knowledge that reflects the quality of actions as well as states, and allowing the shaping to change over time:

$$F(s, a, t, s', a', t') = \gamma\Phi(s', a', t') - \Phi(s, a, t)$$

This extension also preserves the total order over policies and therefore does not change the task, given Ng's original assumptions.

Harutyunyan et al. [2015b] use this result to show how any reward function $R^\dagger$ can be transformed into a potential-based shaping function, by learning a secondary Q-function $\Phi^\dagger$ in parallel on the negation of $R^\dagger$, and using that to perform dynamic shaping on the main reward R. The secondary value function $\Phi^\dagger$ must be learned on-policy, with a technique such as SARSA [Rummery and Niranjan, 1994]:

$$\Phi^\dagger(s, a) \leftarrow \Phi^\dagger(s, a) + \beta\delta^{\Phi^\dagger}$$

where

$$\delta^{\Phi^\dagger} = -R^\dagger + \gamma\Phi^\dagger(s', a') - \Phi^\dagger(s, a)$$

When $\Phi^\dagger$ converges, $F(s, a, s', a') = R^\dagger(s, a)$ in expectation. In other words, when the secondary value function has converged, the main value function will be supplied with a potential-based reward shaping that is equivalent to the reward function $R^\dagger$. Of course, even before convergence, $\Phi^\dagger$ will reflect some useful information pertaining to $R^\dagger$, just as the main Q-function will reflect useful information towards good policies before convergence to the optimal policy.

Note that the non-dynamic variants of reward shaping $\Phi(s)$ and $\Phi(s, a)$ have been shown to be equivalent to Q-function initialization [Wiewiora, 2003; Wiewiora et al., 2003], and thus will not be treated as a separate case from Q-function initialization. The dynamic version has been shown to overcome some problems with static $\Phi(s, a)$ shaping (and thus problems with initialization as well) [Harutyunyan et al., 2015b]. We therefore consider the dynamic variant, and specifically, as defined by Harutyunyan et al. [2015b], as learned Q-values from a reward function $R^\dagger(s, a)$. Since, in this thesis, we consider heuristic information to be encoded in value functions (and policies), we will always set $R^\dagger(s, a) = V_{\text{input}}(s)$ or $R^\dagger(s, a) = Q_{\text{input}}(s, a)$, with $V_{\text{input}}(s)$ or $Q_{\text{input}}(s, a)$ encoding the provided information.

3.3.3 Probabilistic Policy Reuse

Input: π

Fernández and Veloso [2006] propose Probabilistic Policy Reuse (PPR) as a way to transfer a policy learned in a previous task to speed up learning in a new task, by actually reusing the policy in the new task. Whereas a typical reinforcement learning agent will probabilistically choose to either exploit the knowledge it has learned, or explore a random action (the exploration-exploitation trade-off), PPR adds a third option, which is the exploitation of the input policy. With probability ψ, an action is selected according to this reused policy; with probability $1 - \psi$, a standard action selection mechanism, such as ϵ-greedy is used. ψ is decayed over time to allow the learner to emphasize new knowledge more as it learns more in the new task.[2] This adds a bias to the exploration of the agent, intended to guide it towards good policies in the new task.

Although this technique was first introduced to transfer a policy learned in a previous task to a new, similar task, the technique can also be used to transfer any policy, irrespective of where it comes from. For example an existing human-defined policy for the current task can be transferred to the learning agent using PPR, which can then start from a state-of-the-art expert policy and refine it. Even human demonstrations can be taken as an initial phase of PPR with $\psi = 1.0$, the RL agent passively learning, and then switching to active learning, with $\psi = 0.0$ [Schaal, 1997; Smart and Kaelbling, 2002].

3.3.4 Extra Action

Input: π

The 'extra action' technique creates a bias by adding a new action to the reinforcement learning agent's action set A, that, when called, reuses the provided heuristic knowledge [Taylor and Stone, 2007]. In other words, at any time, the agent can decide to either choose one of the regular actions itself, or to 'call the help line,' which then chooses one of the regular actions for the agent to execute. More formally, denote the regular action A, and the new action set A^+, $|A| + 1 = |A^+|$. When in state s, if the agent chooses to use the extra action, i.e. the heuristic information, $\pi_{\text{input}} : S \to A$ is executed.

Assume for the sake of illustration that the input policy is an optimal policy in the current task. Then, simply adding an extra action that will always make the agent execute the best actual action increases this learner's probability of selecting this best action (as there are

[2] In the original PPR algorithm, ψ is reset at the beginning of every episode and decays during the episode, so that the agent relies more on the old policy in the beginning of an episode and less at the end. This makes little sense in the domains considered here, as one would rather want ψ to decay over episodes, so that the agent relies more on the old policy in early episodes and less in later episodes. PPR was also interpreted this way in [Torrey and Taylor, 2012].

now at least two options in its action set that achieve the optimal effect). Furthermore, if the agent generalizes across the state space, as it learns in one state that the extra action is 'good,' it will spread this information to nearby states, even though in those states, a different actual action may be optimal.

3.3.5 Convergence and Optimality

From a theoretical perspective, all of these four techniques leave any convergence guarantees unaffected. That is, if used properly, the reinforcement learner that injects some knowledge into its learning process using one of these techniques may learn faster or slower, but it will still learn to solve the task, given guarantees provided by its learning algorithm. Q-value initialization and reward shaping are guaranteed not to change the task [Ng et al., 1999; Wiewiora, 2003]. If PPR's ψ is properly decayed, it stops affecting the learning process after a while. And the extra action method simply duplicates one of the actions in each state, thus leaving the actual problem unchanged, as well as leaving any convergence guarantees in place.

3.4 A Sampling from the Literature

To demonstrate that the categorisation we outlined above is relevant, we provide a brief overview of how a representative selection of techniques from the literature falls into this categorisation. We only consider papers that discuss injecting a TD-learner with prior or external information and classify them regarding knowledge encoding and injection technique, see Table 3.1.

For example in **transfer learning**, we see Selfridge et al. [1985] employing Q-function initialization (injection) using Q-values (encoding) learned in a previous Cart Pole task to learn in a differently parameterized Cart Pole task. Fernández and Veloso [2006] on the other hand proposed probabilistic policy reuse (injection) to enable the transfer of a previously learned policy (encoding).

In a **demonstration** setting, we see Taylor et al. [2011b] distilling the provided demonstrations into a demonstrator policy (encoding) using a rule-based classifier, and using this knowledge through Q-function initialization, probabilistic policy reuse and the extra action method (injection). Schaal [1997] replays the human demonstrations in an initial passive learning phase for the agent, after which further active reinforcement learning takes place. This can be seen as reusing the demonstrator's policy (encoding) through probabilistic policy reuse (injection) with a probability of 1.0 for policy reuse during the initial passive learning phase.

Paper	Encoding	Injection
Transfer Learning		
[Selfridge et al., 1985]	Q	Q
[Fernández and Veloso, 2006]	π	PPR
[Konidaris and Barto, 2006]	V	Q, Shaping
Demonstrations		
[Schaal, 1997]	π	PPR
[Taylor et al., 2011b]	π	Q, PPR, EA
[Brys et al., 2015a]	Q	Q, Shaping
Off-line Advice		
[Ng et al., 1999]	V	Shaping
[Smart and Kaelbling, 2000]	π	PPR
[Devlin et al., 2011]	V, Q	Shaping
On-line Advice		
[Knox and Stone, 2010]	Q	Q, Shaping, PPR, EA
[Taylor et al., 2014]	π	PPR

Table 3.1: A non-exhaustive list of papers describing work where some heuristic information is incorporated into a TD process using one or more of the ways we enumerate in this chapter.

A lot of **off-line advice** work is found in shaping literature, with the knowledge encoded as a value function called potential function (encoding), injected using Q-function initialization or reward shaping [Ng et al., 1999; Devlin et al., 2011]. Smart and Kaelbling [2002] allow for an expert input policy (encoding) to be reused (injection) through passive learning, as Schaal's work referenced above.[3]

Finally, Knox and Stone use all four injection methods to provide a learning agent with **on-line advice** provided in the form of a learned Q-function (encoding) [Knox and Stone, 2010]. And Taylor et al. [2014] their use of on-line teacher advice can be cast as reusing the teacher's policy (encoding) using probabilistic policy reuse (injection), where the reuse probability is selected by the teacher who decides when to give advice.

3.5 A Brief Detour: How to Measure Improvement

As we investigate different techniques to improve an RL algorithm, the question arises of how to measure this improvement. Is it reflected in the behaviour the agent ultimately converges to? That is desirable if it is operating in conditions where convergence to the optimal policy is not guaranteed. Is it reflected in the number of experience samples required to achieve this behaviour (or a given level of performance)? Preferably, since sample efficiency is the major concern in reinforcement learning, much more than computational complexity. Environment samples are usually assumed to be very costly to obtain (making a robot flip a pancake 1000s of times is costly).

In principle, all the metrics outlined in the seminal RL transfer learning survey paper by Taylor and Stone [2009] for measuring improvement when using transfer versus no transfer are generally applicable to measure improvement of any RL algorithm over a baseline. These metrics are:

- Jumpstart: the difference in initial performance
- Asymptotic performance: the final performance after training
- Total reward (or average reward): the (averaged) sum of performance over the whole training time
- Reward ratio: the ratio between the total reward of the novel technique versus the baseline
- Time to threshold: the amount of training (samples, episodes, ...) required to achieve a given level of performance

A few of these are illustrated in Figure 3.1.

[3] Actually, both Schaal's work and Smart and Kaelbling's could be classified as learning from demonstration as well as off-line advice, as they allow for both expert demonstrations and predefined expert policies as input.

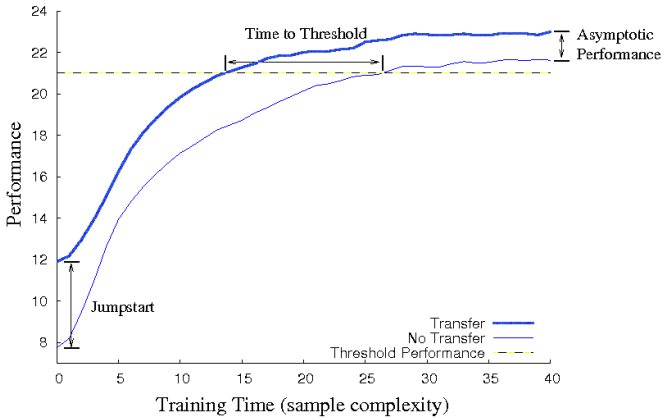


Figure 3.1: An illustration of some of the possible metrics usable to compare the learning curves of different RL learning techniques. Figure taken from [Taylor and Stone, 2009].

In this thesis we will mainly look at total/average reward as a summary statistic to compare learning techniques, since it subsumes the other metrics (a higher jumpstart will be reflected in a higher total reward, a lower asymptotic performance will be reflected in a lower total reward, etc.).

In general, our experimental set-up is as such: We assume a given budget of episodes ($\neq$ to experience samples), i.e. distinct attempts at the task. Then we measure the average or total performance during these episodes of learning to compare the different techniques.

3.6 Summary

In this chapter, we gave an overview of the use of prior or external knowledge in temporal difference learners in order to speed up their learning process. From existing literature, we first characterized four important sources of such heuristic knowledge: previous tasks, demonstrations, and off-line and on-line advice. We briefly discussed various ways of encoding such knowledge and narrowed down this range to two options we will explore in this thesis (policies and value functions) and showed how these can be interchangeable. Then we went on to describe four distinct approaches to injecting this knowledge into a TD-learner: Q-function initialization, reward shaping, probabilistic policy reuse and the extra action method. We then briefly showed how a selection of representative papers from

the literature fall into these distinct categories, and finally we described our methodology for comparing the performance of different reinforcement learning techniques and variants.

Now it is time to experimentally investigate the knowledge injection techniques discussed in this chapter. We do this in the context of *policy transfer*.

4 | Policy Transfer

> What you have [previously] learned ... practice these things.
>
> St. Paul, *Philippians 4:9*

Transfer learning in general formulates the question of how to reuse knowledge learned in a previous task in order to learn faster or better in the current task. Many transfer learning techniques transfer low-level information to achieve successful transfer. A more general transfer approach on the other hand would only assume access to the output of the learning algorithm in the previous task, i.e. the learned policy. Such an approach enables transfer irrespective of the internal workings of the learning algorithm used in the source task – policy gradient, dynamic programming, Q-learning, etc. This setting is called *policy transfer*. In this chapter, we evaluate Q-function initialization, dynamic reward shaping, probabilistic policy reuse and the extra action method, the knowledge injection techniques we identified in the previous chapter, in the case of policy transfer.

But before we do that, we give a slightly broader description of the field of transfer learning in reinforcement learning than we did above.

4.1 Transfer Learning

The field of transfer learning is much larger than just the reinforcement learning techniques we will concern ourselves with. It is not hard to see that learning agents, including humans,

typically face sequences of tasks to solve, and that solving each task in isolation, as if it has nothing to do with anything ever seen before, is quite limiting and generates much overhead in the agent [Thrun and Pratt, 2012]. Humans have long been recognized as employing transfer learning [Thorndike and Woodworth, 1901; Skinner, 1951] and it has been successfully applied in machine learning, well before it became popular in reinforcement learning [Caruana, 1995; Thrun, 1996]. Transfer learning in reinforcement learning was popularized by the successful and comprehensive work of Taylor and Stone [Taylor et al., 2007; Taylor and Stone, 2009] and has seen a great growth in techniques and applications since. We reiterate that the core of transfer learning is to re-use knowledge obtained in (a) previous task(s), in order to better learn, or learn faster in the current task. Some similarities between the tasks must be assumed for transfer to be beneficial. Such similarities are usually to be found in the systems' dynamics, reflected in their state transition functions, coupled with their reward functions. An example is the 'swinging'-dynamics present in the Inverted Pendulum, Cart Pole and Mountain Car tasks, which can be leveraged to successfully transfer between them [Bou Ammar et al., 2013].

Typically, in order to leverage these similarities in dynamics, transfer algorithms have to be provided an inter-task mapping that defines a translation between the state and action spaces of the source and target task. Such mappings can be seen as transforming one task into another, and if this transformation is defined well, the tasks' dynamics will be 'correctly' mapped onto each other and the knowledge transfer will prove to be useful. Mappings χ_S and χ_A, for state and action spaces respectively, take a state or action from the target task and map it onto a state or action in the source task:

$$\chi_S : S_{\text{target}} \rightarrow S_{\text{source}}$$

and

$$\chi_A : A_{\text{target}} \rightarrow A_{\text{source}}$$

A simple approach to transfer called Q-value reuse [Taylor et al., 2007] uses these mappings and learned Q-values from the source task to initialize the Q estimates in the target task: $\hat{Q}_{\text{target}}(s, a) = \hat{Q}_{\text{source}}(\chi_S(s_{\text{target}}), \chi_A(s_{\text{target}}))$. Of course, this approach presupposes the availability of such Q-values learned in the source task.

In the same way that χ is defined to be a mapping from target to source task, ρ defines a mapping from source to target task:

$$\rho_S : S_{\text{source}} \rightarrow S_{\text{target}}$$

and

$$\rho_A : A_{\text{source}} \rightarrow A_{\text{target}}$$

Note that $\chi_S(\rho_S(s_{\text{source}})) = s_{\text{source}}$ need not always be true (and analogously for χ_A and ρ_A), as both χ and ρ may be stochastic and non-injective, or even inconsistently defined.

That is, a state or action from the source (or target) task may map to several states and actions in the target (or source) task, and several states and actions from the source (or target) task may map to a single state or action in the target (or source) task.

Defining these mappings requires domain expertise and insights into what may be the similarities and differences between the dynamics of the source and target tasks under consideration. Attempts have been made to overcome this requirement of expertise by automating the design of these mappings using regression techniques [Bou Ammar et al., 2012, 2013], or by learning on-line which are the best ones from a huge set of possible mappings [Fachantidis et al., 2015]. These techniques were developed for other settings than the temporal difference learning one considered in this thesis.

4.2 Policy Transfer

In this chapter, we focus on policy transfer [Fernández and Veloso, 2006], a more general case with respect to the knowledge transferred compared to the more common value transfer for example [Taylor et al., 2007]. It assumes the only available knowledge from the source task is the output of the learning algorithm, i.e. the policy. Compare this with the lower-level Q-value reuse approach referred to above. That approach assumes that the learner in the source task has learned a Q-function, thereby excluding all direct policy search algorithms (including the successful policy-gradient methods). Policy transfer on the other hand can safely assume that a learned policy (if anything) will be available from the source task, since it is every RL agent's goal to learn a policy.[1]

Probabilistic policy reuse is, to the best of our knowledge, the only existing technique that transfers a full policy to a TD-learner. Other techniques transfer options , which are partial policies [Ravindran and Barto, 2003; Konidaris and Barto, 2007].

[1] Another difference between policy transfer and value transfer is that with value transfer, the reward function in the target task may not differ too much from the one in the source task. Say that a learning agent has learned to navigate a maze to the goal location (step-reward 0, goal-reward 100). Assume furthermore that we then transfer this agent's Q-values to a maze task with exactly the same layout, start and goal locations, except that now the agent gets a step-reward of -1 and a goal-reward of 0. The optimal policy in this new task is exactly the same as the policy in the first task (if $0 < \gamma < 1$). Yet, initializing the Q-function with the values learned in the first task will lead to a lot of random exploration, since these Q-values will be initialized very optimistically due to the large difference in reward magnitude. Policy transfer on the other hand is invariant to such syntactic changes without practical difference in a reward function.

4.2.1 Reusing a Policy using Mappings

In the previous chapter, we identified two techniques that allow one to directly reuse a given policy in a new task, and two techniques that require the construction of a pseudo-value function based on the given policy to be able to reuse it. We will first deal with the how-to's for policy transfer of the former two, which are probabilistic policy reuse (PPR) and extra action. Specifically, when the agent decides to reuse a previously learned policy in a new task using PPR or the extra action method, it needs to

- map the new task's state to the state space of the previous task,
- select an action according to the old policy given that state, and
- map that chosen action to the new task's action space.

This is summarized in the following formula:

$$\pi_{\text{reuse}}(s_{\text{target}}) = \rho_A(\pi_{\text{source}}(\chi_S(s_{\text{target}}))$$

Even though both the χ and ρ mappings may stochastically map to several states and actions, in the end only a single action can be executed.

In the case of reusing the policy through a value function (necessary for Q-function initialization and reward shaping), the stochasticity of the mappings can be leveraged more. But first let us look at the simplest case, where we assume that χ_S and χ_A are injective functions, mapping every state or action from the target task to a single unique state or action in the source task. In that case, a pseudo Q-function can be constructed as discussed in the previous chapter, including the mappings in a straightforward way:

$$Q_{\text{reuse}}(s, a) = \pi_{\text{source}}(\chi_S(s), \chi_A(a))$$

Now if χ_S and χ_A map to respectively n and m different states and actions in the source task, we can incorporate these different mappings as follows:

$$Q_{\text{reuse}}(s, a) = \frac{\sum_i^n \sum_j^m \pi_{\text{source}}(\chi_{S,i}(s), \chi_{A,j}(a))}{nm}$$

$\chi_{S,i}(s)$ indicates the i^{th} distinct state in the source task χ_S maps target task state s to. $\chi_{A,j}(a)$'s notation carries the same semantics.

4.3 Experiments

We will now describe a set of experiments designed to test the efficacy of Q-function initialization, dynamic reward shaping, probabilistic policy reuse and the extra action method in transferring a policy learned in a previous task related to the current one.

Our first experiments are conducted in the Cart Pole task.

4.3.1 Early and Late Policy Transfer in Cart Pole

Recall that the Cart Pole task involves the agent controlling a little cart, moving in a single dimension, and requires the agent to balance a pole that is standing on top of the cart. The transfer set-up is as follows: the agent first learns in a Cart Pole task with a pole of weight 0.1, and is then allowed to transfer its learned policy to learn to balance a pole of weight 1.0. The differences between the two tasks lie solely in the tasks' dynamics. State transitions will be different due to the differences in weight of the pole. No state or action space mappings are required to translate between the tasks, as these are identical.

Performance of our standard $Q(\lambda)$-learning agent in the light Cart Pole task is shown in Figure 4.1 (a). We will perform two sets of transfer experiments:

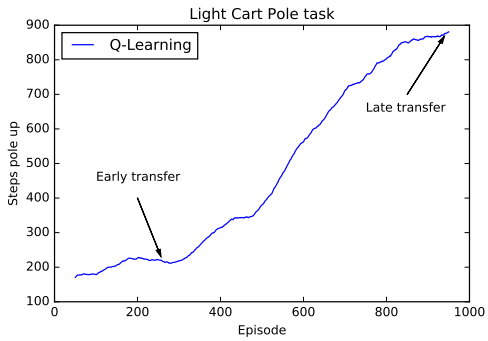
- one set where we transfer after learning for only 250 episodes in the light Cart Pole task, transferring a very suboptimal policy,
- one set where we transfer after learning for 1000 episodes, transferring a much improved policy.

The policy we transfer is deterministically greedy with respect to the Q-values learned in the source task. All experiments in this thesis consist of 100 different trials to test statistical significance. Statistical significance is tested using the Student's t-test with $p = 0.05$. PPR is parameterized with $\psi = 0.99^{\text{episode}}$ and dynamic shaping with $\beta = 2\alpha$, both decided based on a few preliminary experiments with different values.

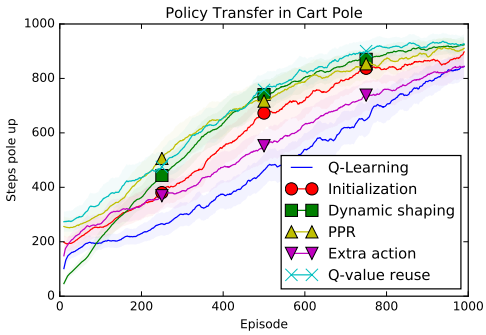
Early Transfer

The performance of our baseline learning agent, learning without transfer, as well as that of using one of the four injection techniques with our baseline learning agent is plotted in Figure 4.1 (b). We furthermore plot the performance of Q-value reuse, the technique that 'cheats' (at least in our policy transfer setting) and reuses the Q-values learned in the source task, instead of the policy. This will give an indication of how close in performance the more general policy transfer techniques get to a potentially more powerful technique whose application is more constrained as it requires low-level information that may not always be available.

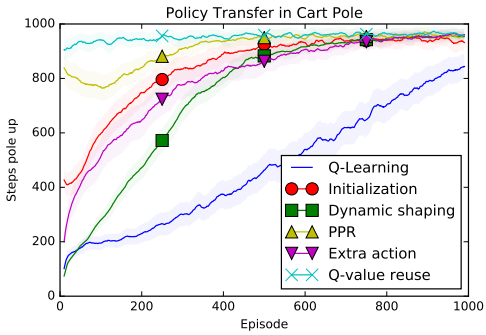
The simple observation we draw from this experiment is that, indeed, all four knowledge injection techniques we investigate can be used for successful policy transfer. Each of the techniques leverages the quite suboptimal information provided and helps the learning agent to learn balancing the heavy pole using fewer experiences. Probabilistic policy reuse comes closest to matching the performance of Q-value reuse, which is quite likely an upper bound on what is possible given the provided information.



(a) $Q(\lambda)$-learning performance in the light Cart Pole task.



(b) Learning performance in the heavy Cart Pole task after first learning for 250 episodes in the light Cart Pole task.



(c) The same as (b), only after 1000 learning episodes in the light Cart Pole task.

Figure 4.1: Evaluating policy transfer from a Cart Pole task with a light pole to one with a heavy pole.

Algorithm	Cumulative (early)	Cumulative (late)
Q-Learning	$4.7 \cdot 10^5 \pm 3 \cdot 10^4$	
Initialization	$6.0 \cdot 10^5 \pm 3 \cdot 10^4$	$8.4 \cdot 10^5 \pm 2 \cdot 10^4$
Dynamic shaping	$6.4 \cdot 10^5 \pm 3 \cdot 10^4$	$7.4 \cdot 10^5 \pm 1 \cdot 10^4$
PPR	$6.6 \cdot 10^5 \pm 4 \cdot 10^4$	$9.1 \cdot 10^5 \pm 2 \cdot 10^4$
Extra action	$5.4 \cdot 10^5 \pm 4 \cdot 10^4$	$8.0 \cdot 10^5 \pm 3 \cdot 10^4$
Q-value reuse	$6.9 \cdot 10^5 \pm 3 \cdot 10^4$	$9.5 \cdot 10^5 \pm 4 \cdot 10^3$

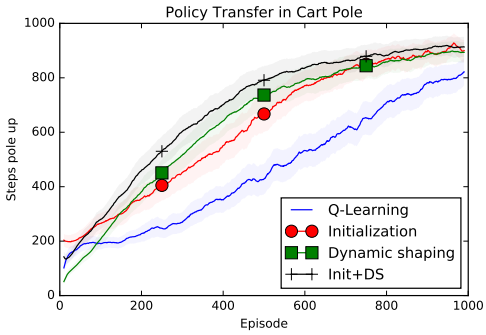
Table 4.1: Performance on the heavy Cart Pole task in the early and late transfer settings. The learning agent's performance improves with the quality of the transferred policy. Every technique significantly outperforms the $Q(\lambda)$-learning baseline

Late Transfer

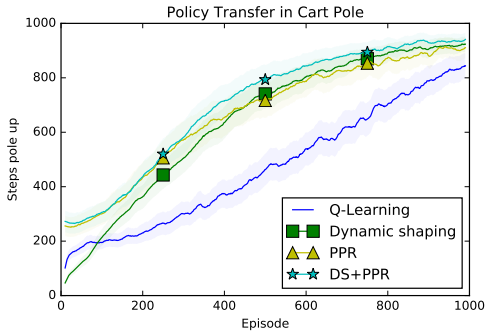
In the 'late' setting, the agent transfers a policy to the heavy Cart Pole task after learning four times longer in the light version of the task. The quality of the transferred policy is coincidentally also four times higher. Comparing the early and late experiments' results in Figures 4.1 (b) and (c), and Table 4.1, we observe the expected correlation between the quality of transferred policies and the 'speed' of learning in the target task, reflected in higher cumulative performance. Each of the transfer techniques benefits from this effect, and especially probabilistic policy reuse and Q-value reuse manage to leverage the transferred information really well. In the 'late' setting, these two techniques start out performing already close to optimally.

Combining Techniques

Since the different injection techniques we investigate operate on different components of the agent (Q-function initialization on the agent's estimates, dynamic reward shaping on the reward signal, PPR on the action selection mechanism and extra action on the action set) these techniques can be used in conjunction. We test two combinations. Figure 4.2 (a) demonstrates how Q-function initialization and dynamic shaping are complementary in this setting, as together they yield a significant improvement in performance over these techniques when used alone, despite the fact that they inject the same information. Figure 4.2 (b) shows how combining dynamic shaping and PPR results in performance statistically indistinguishable from PPR alone.



(a) Q-function initialization + dynamic shaping



(b) Dynamic shaping + PPR

Figure 4.2: Combining compatible techniques in the 'early' transfer setting.

4.3.2 Multi-Agent Policy Transfer in the Pursuit Domain

In the Cart Pole problem, we already found that the four injection techniques we are considering are all potentially useful in the case of policy transfer. In this section we continue our investigation by looking at the Pursuit domain, a multi-agent problem. Recall that it involves two learning agents navigating a grid world, cooperating to catch a prey.

To obtain a source task to transfer from, we simplify the Pursuit problem by making it a single-agent task in which the predator must learn to catch an immobile prey. This task should allow the learning agent to learn the simple concept of moving towards a prey, a behaviour that will then probably need to be refined in the multi-agent task with a moving prey, where encircling and other cooperative strategies may be required to actually efficiently capture the prey together with the other predator. Learning performance in the simplified task and the point of transfer are shown in Figure 4.3 (a).

The state space in the simplified task consists of only two variables (relative x and y distance to the prey), as opposed to four in the full task (relative x and y distances to the prey and the other predator). Therefore, we need to define a mapping from the target to the source task's state space to leverage the learned information in the source task:

$$\chi_S([x_{\text{pred}}, y_{\text{pred}}, x_{\text{prey}}, y_{\text{prey}}]) = [x_{\text{prey}}, y_{\text{prey}}]$$

The reverse mapping need not be defined since ρ_S is not used in the techniques we consider. Furthermore, the action spaces in both tasks are the same (the cardinal directions to move in plus an action for remaining in place) and thus χ_A and ρ_A are simply set to be the identity function. PPR is parameterized with $\psi = 0.99^{\text{episode}}$ and dynamic shaping with $\beta = \alpha$.

Algorithm	Cumulative (single)	Cumulative (multi)
Q-Learning	$4.5 \cdot 10^4 \pm 3 \cdot 10^3$	
Initialization	$2.8 \cdot 10^4 \pm 2 \cdot 10^3$	$2.2 \cdot 10^4 \pm 1 \cdot 10^3$
Dynamic shaping	$1.1 \cdot 10^5 \pm 1 \cdot 10^4$	$8.8 \cdot 10^5 \pm 7 \cdot 10^4$
PPR	$4.4 \cdot 10^4 \pm 5 \cdot 10^3$	$2.8 \cdot 10^4 \pm 4 \cdot 10^3$
Extra action	$2.5 \cdot 10^4 \pm 2 \cdot 10^3$	$2.0 \cdot 10^4 \pm 2 \cdot 10^3$
Q-value reuse	$1.6 \cdot 10^4 \pm 1 \cdot 10^3$	$1.1 \cdot 10^4 \pm 8 \cdot 10^2$

Table 4.2: Performance on the pursuit domain (measured in number of steps to catch the prey) in the single-agent transfer and multi-agent transfer settings. Dynamic shaping is significantly worse than the baseline in both settings, while PPR is indistinguishable from the baseline in the former and better in the latter. The other techniques are in both cases significantly better.

In this domain, we will not investigate the effect of source task policy quality on transfer. Rather, besides simply looking at whether or not the four injection techniques are useful in this domain, we will look at the effect of transferring a policy to only a single one of the predators (letting the other learn from scratch) or to both.

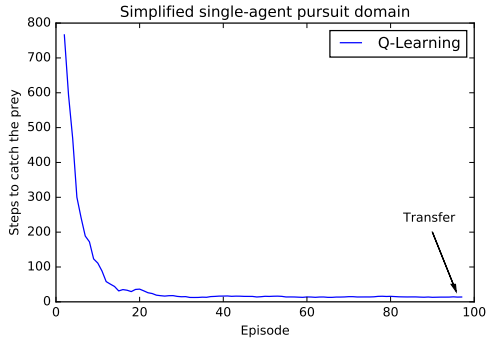
Single-agent Transfer

Figure 4.3 (b) compares the performance of our baseline agent with transferring a high quality policy from the simplified task to only one of the two predators using the transfer techniques we consider. Notably, using dynamic shaping or PPR in this case yields worse performance than the baseline agent, except at the initial phases of learning. Q-function initialization and the extra action method on the other hand yield improvements, approaching the level of Q-value reuse.

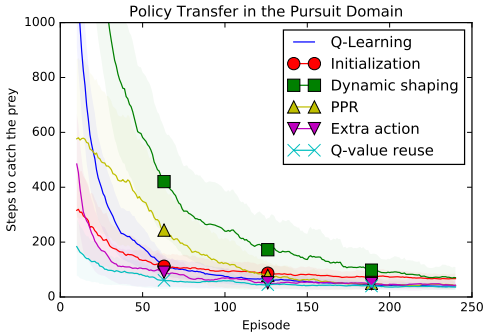
Multi-agent Transfer

Transferring the learned policy to both agents instead of only one improves performance for all techniques except for dynamic reward shaping, which performs extremely bad, see Figure 4.3 (c) and Table 4.3. Not considering dynamic reward shaping,[2] this is expected behaviour. Priming both learning agents with useful information results in faster learning of good cooperative behaviour.

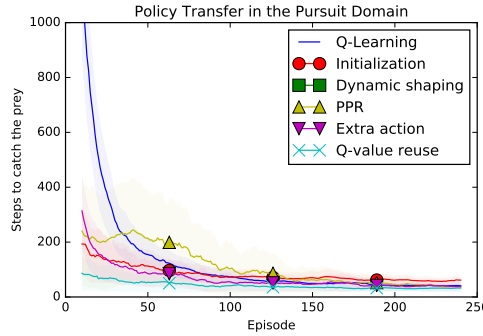
[2]Dynamic shaping can be unstable because two sets of Q-values which influence each other are learned in parallel on different time scales. This could be resolved by tuning the two learning rates' schedules. Private communication with Anna Harutyunyan.



(a) $Q(\lambda)$-learning performance in the simplified single-agent pursuit domain.

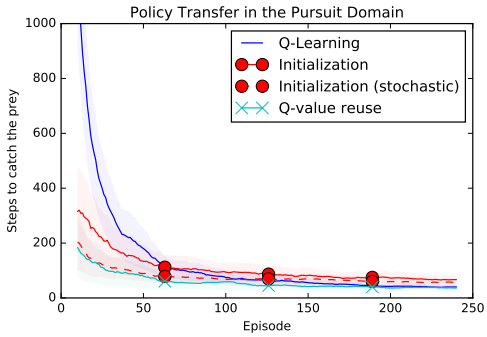


(b) Learning performance in the pursuit domain, with policy transfer to one of the two predators.

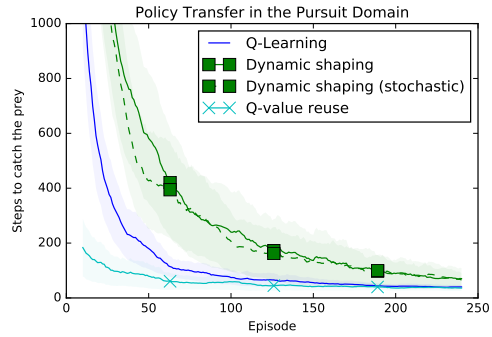


(c) Policy transfer to both predators (the same policy).

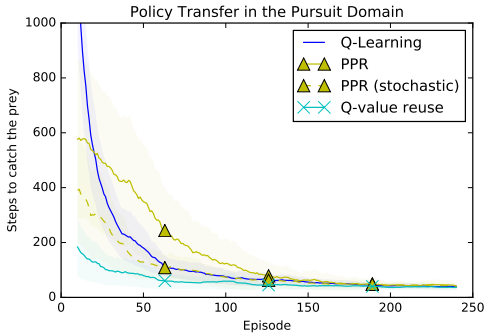
Figure 4.3: Evaluating policy transfer in the pursuit domain, transferring to either a single or to both predators. In the latter case, dynamic shaping is not featured on the figure, because it performs an order of magnitude worse.



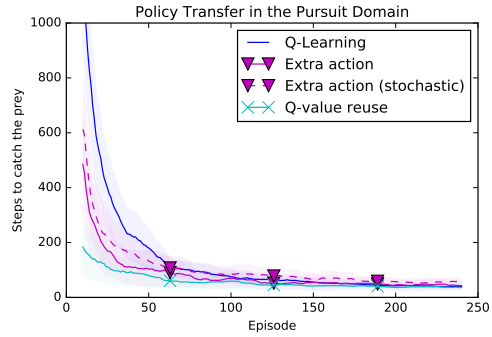
(a) Q-function initialization



(b) Dynamic reward shaping



(c) PPR



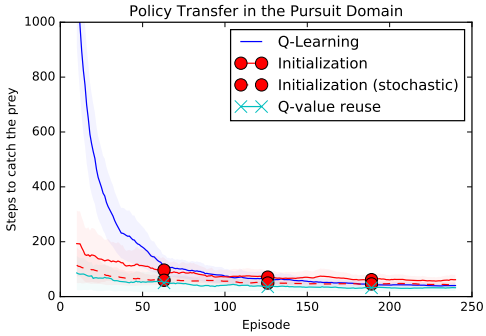
(d) Extra action

Figure 4.4: Transferring a deterministic vs a stochastic policy to a single of the predators.

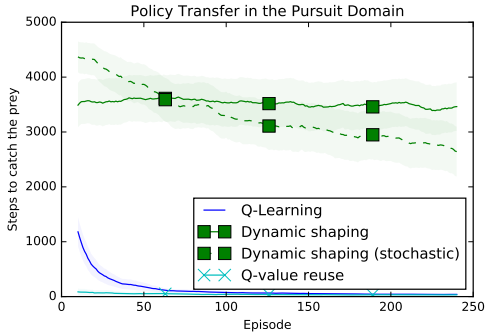
Transferring a Stochastic Policy

Both in the Cart Pole task and in the Pursuit domain, we have been transferring deterministic policies. Such deterministic policies carry only information on which action in a given state is deemed the best, and are indiscriminate about the other actions, even though some of these actions may lead to returns almost as high as the best action (or even higher in the case of suboptimal policies). When transferring such a policy, and the 'best' action in a state (according to the reused policy) turns out not to be the best action, we are left with a strong bias towards that action, and have no information on the other actions.

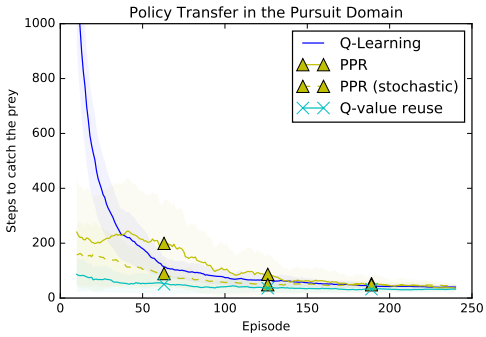
Stochastic policies may resolve this problem. Policies that generate action-selection probabilities based on the relative expected returns of actions, such as soft-max action selection, do carry such information. Like deterministic policies, they attribute the highest



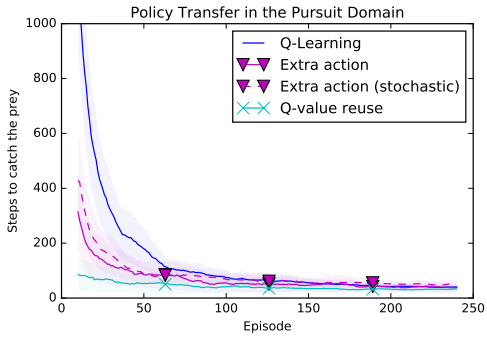
(a) Q-function initialization



(b) Dynamic reward shaping



(c) PPR



(d) Extra action

Figure 4.5: Transferring a deterministic vs a stochastic policy to both predators. Notice the larger y-scale on the dynamic reward shaping sub-figure.

Algorithm	**Deterministic**	**Stochastic**
Q-Learning	$4.5 \cdot 10^4 \pm 3 \cdot 10^3$	
Initialization	$2.2 \cdot 10^4 \pm 1 \cdot 10^3$	$1.4 \cdot 10^4 \pm 7 \cdot 10^2$
Dynamic shaping	$8.8 \cdot 10^5 \pm 7 \cdot 10^4$	$8.3 \cdot 10^5 \pm 5 \cdot 10^4$
PPR	$2.8 \cdot 10^4 \pm 4 \cdot 10^3$	$1.8 \cdot 10^4 \pm 2 \cdot 10^3$
Extra action	$2.0 \cdot 10^4 \pm 2 \cdot 10^3$	$2.5 \cdot 10^4 \pm 2 \cdot 10^3$
Q-value reuse	$1.1 \cdot 10^4 \pm 8 \cdot 10^2$	

Table 4.3: Performance on the pursuit domain in the two-agent-transfer setting, transferring either deterministic or stochastic policies. Initialization and PPR perform significantly better with stochastic transferred policies, extra action significantly worse and dynamic shaping better, but not significantly.

action selection probability to the action deemed best, but the other actions are also attributed non-zero probabilities relative to their expected quality. We hypothesize that deterministic policies define a narrow bias, that, if wrong, can be harmful to the learning process. Stochastic policies on the other hand have the potential to define a broader, more nuanced bias, possibly ensuring better transfer.

In Figures 4.4 and 4.5, we demonstrate for the two transfer scenarios respectively that transferring a stochastic policy[3] instead of a deterministic one results in improved performance for Q-function initialization, dynamic reward shaping and PPR.[4] The agent incorporating such a stochastic policy benefits from the broader bias provided.

Figure 4.6 shows the result of an experiment investigating the effect of the greediness versus the randomness of transferred policies in the first pursuit transfer scenario, by varying the τ parameter in the transferred soft-max policy. A good exploration-exploitation trade-off (balancing between fully greedy and fully random) provides the best results in transfer, as this yields a good balance between the too-broad uniform exploration and the too-narrow deterministic policy transfer.

[3] A soft-max policy with temperature parameter $\tau = 0.1$, selected from a brief parameter search.

[4] It results in slightly worse performance for the extra action method. The peculiarity of that method is that it does not affect exploration heavily. Rather, it adds an extra action that executes what is believed to be a good action. Besides that, the agent starts out with equal estimates for all actions, has no bias in its reward signal towards some actions, and its action selection is not overruled by the transferred policy. Thus, it appears that, at least in this setting, this already small bias should not be diluted using a stochastic policy, favouring a deterministic policy as input. We elaborate on this in the next section.

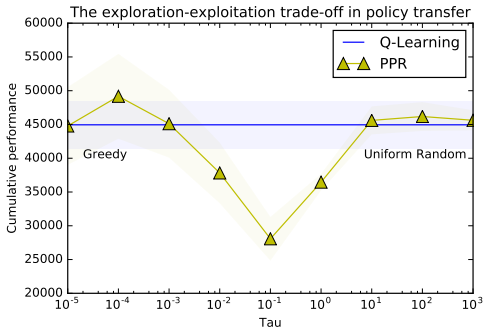


Figure 4.6: The effectiveness of policy transfer using PPR for various exploration-exploitation trade-offs in the transferred stochastic policy. Lower values are better.

4.3.3 Small and Large Bias in Mario

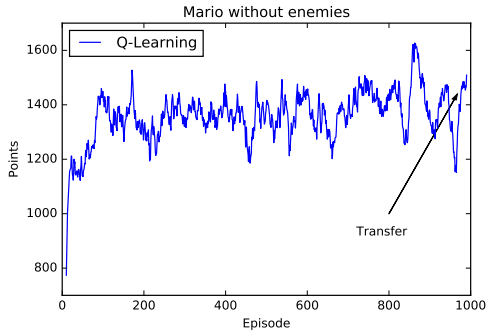
Our last experiments are executed in the super Mario environment, a more complex problem than the Cart Pole task and the pursuit domain. Recall that it involves a state space of 27 different variables. Assuming that the presence of enemies makes completing a level harder, we set-up a super Mario source task without enemies. Due to the absence of enemies, the source task agent only considers 9 state variables, disregarding all those pertaining to enemies, which leads to a vastly reduced state-space. In this source task, the agent can learn to navigate super Mario levels efficiently, without the complications that the presence of enemies introduce. Figure 4.7 (a) shows learning in the source task and the transfer point.

The mapping from the target task's state space to the source task's state space is straightforward, selecting all the variables that do not pertain to enemies (the first 9 out of the 27):

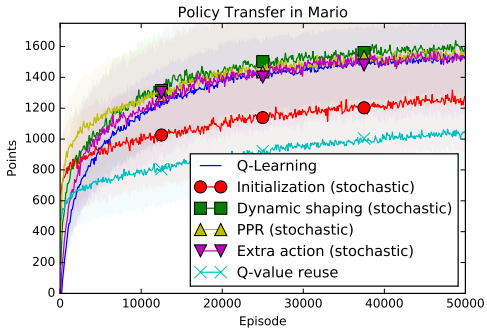
$$\chi_S(s_{1..27}) = s_{1..9}$$

Since also in this setting the action spaces in the source and target tasks are the same, we do not need to define χ_A nor ρ_A. We transfer stochastic policies, generated using $\tau = 5$, selected from a brief parameter search. PPR is parameterized with $\psi = 0.99^{\text{episode}}$ and dynamic shaping with $\beta = \alpha$.

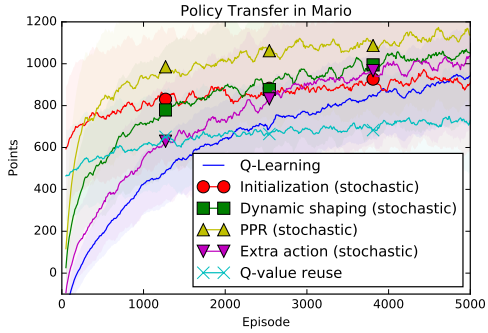
Figure 4.7 (b) and Table 4.4 show a comparison between the baseline Q-learning, the four policy transfer injection techniques and Q-value reuse. Once more we see a different



(a) $Q(\lambda)$-learning performance in enemy-free super Mario.



(b) Learning performance in the hostile super Mario environment, after transfer from the enemy-free environment.



(c) Detail of the first 5000 episodes shown in (b).

Figure 4.7: Evaluating policy transfer in super Mario.

Algorithm	**Cumulative**
Q-Learning	$6.5 \cdot 10^7 \pm 7 \cdot 10^5$
Initialization	$5.5 \cdot 10^7 \pm 3 \cdot 10^6$
Dynamic shaping	$7.0 \cdot 10^7 \pm 1 \cdot 10^6$
PPR	$6.9 \cdot 10^7 \pm 1 \cdot 10^6$
Extra action	$6.7 \cdot 10^7 \pm 1 \cdot 10^6$
Q-value reuse	$4.5 \cdot 10^7 \pm 2 \cdot 10^6$

Table 4.4: Performance in super Mario. Initialization and Q-value reuse are significantly worse than Q-learning, the other techniques are significantly better than Q-learning.

picture emerge. Dynamic shaping and PPR perform best (they are statistically indistinguishable), and the extra action method also manages to outperform the baseline. Q-value initialization and Q-value reuse on the other hand perform much worse than the baseline, except initially (the former until episode 5000, the latter until episode 2000).

The explanation for these results is to be found in a combination of the peculiar state visitation distribution in Mario, the fact that the transferred information is quite likely not that good in many states, and the 'size' of the bias each injection technique causes the agent to have in unseen states.

First, recall that the state visitation pattern in super Mario is an exponential distribution (Section 2.4.3). Many states are only visited once (approximately 30% of the state visits in a 5000 episode run of baseline Q-learning are unique), and few are visited many times. This means that the learning agent is constantly visiting states it has never seen before, and thus has no internal information on the quality of actions in those states. Of course, in our setting, the agent receives an external bias, provided by the transferred information.

Second, we believe that the transferred information in this case is useful only in the small minority of states that is visited exponentially many times, and not so useful in many of the states that are visited only very sparsely. That is because the transferred information completely ignores enemies, because the source task does not contain enemies. This is perfectly fine in the states where there are no enemies (exactly those states which are visited mostly – only 72 states), but this can be problematic in states where there are enemies (the states that are visited sparsely – $36,540,776,376$ states), making the agent commit suicide by running into enemies. Now if the provided bias is too strong in those risky, sparsely visited states, the agent will not have time to overcome this bias.

This leads us to the third and final point. Let us consider the biases provided by each of the knowledge injection techniques in a previously unseen state, by looking at the action selection probability they assign to the greedy action according to the transferred information. These probabilities are shown in Figure 4.8 on a log-scale. With Q-value

initialization, the greedy action according to the transferred information is taken with probability $1-\epsilon$, simply because the agent's policy is ϵ-greedy with respect to the Q-values. In our setting, the probability is 0.95, because $\epsilon = 0.05$. With PPR, there is a probability ψ of taking this greedy action, which within 1000 episodes decays to uniformly random: $\frac{1}{12} \approx 0.083$ for 12 actions in Mario. With the extra action method, we have probability $\frac{2}{13} \approx 0.154$ that this action is taken (12 actions in Mario plus the extra action), slightly higher than purely random action selection. With dynamic shaping, we have the uniformly random probability $\frac{1}{12} \approx 0.083$ that this action is taken, because in unseen states, the potential function is still initialized to 0.

It is clear that initializing the Q-values always provides a very large bias towards the heuristic information, while the others are near-random (after a while) for unseen states. That is why the agent initially benefits a lot using initialization, because it provides a strong bias in unseen states, and since a large number of visits is to states where this information is useful, the agent performs well really quickly. But, because the information is not useful in those states that are visited only sparsely, and many of such states are visited, the agent has a bias that is too strong there, and therefore learning slows down. Since the other techniques provide a smaller bias, the initial improvement compared to the baseline is not as pronounced as for initialization, but their bias is small enough not to cause problems in the large number of sparsely visited states where the information is incorrect. PPR and dynamic shaping's effect is largest on states that are visited more regularly, while extra action's effect is equal over all states, but small. Initialization's effect is also equal over all states, but also very large. That is why these three former techniques work well, while initialization performs well initially, but slows down learning later on. This is true for both policy transfer through Q-value initialization and for Q-value reuse.

4.4 On Bias

From the last section, we can find some simple intuitions on how each of the techniques provides a bias for the learning process. We distinguish between a bias at the first state visit, and after a number of visits.

Q-value initialization provides a strong bias at the first state visit (actually even at the first state-action visit), but this bias gets increasingly replaced by the actual returns sampled, until it eventually completely disappears in the limit.

Dynamic shaping provides no bias over uniform random at the first visit of a state. Its bias on the other hand increases with each visit, as after each update, the dynamic potential function gets updated, and the shaping bias gets applied to the learner's Q-values. In the limit, this bias again disappears, given the theoretical guarantees of reward shaping.

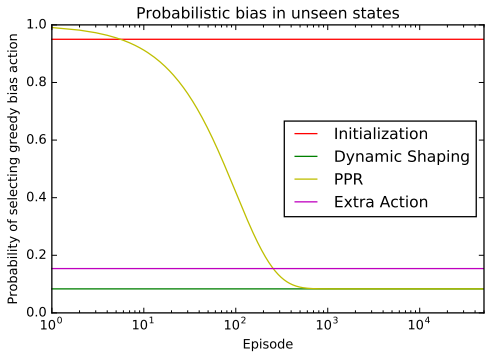


Figure 4.8: Assume an unseen state. This figure plots the probability that the learning agent selects the greedy action according to the heuristic information in that state. Plotted for each of the injection techniques. Initialization provides a very large and lasting bias. When the bias is incorrect, this is very harmful when many state visits are (near-) unique.

PPR is parametrizable to provide any possible bias at any given timestep. But, typically, the initial bias is highest (in our case 1.0), and is set to decrease over time.

Extra action provides initially a bias slightly higher than uniformly random, since its 'extra action' is the shadow copy of one of the other actions. This bias also decreases progressively as the agent updates the Q-values in a state.

4.5 Summary

We evaluated the usefulness of the four knowledge injection techniques we identified in the previous chapter and in general found that indeed each of the techniques is plausible and works, but that none of them performs consistently well across benchmarks. Different interesting effects were observed by looking at the correlation between transferred policy quality and performance in the target task, between single and multi-agent transfer and performance, and the effect of a narrow or broad bias provided by respectively deterministic and stochastic transferred policies. We also note the fact that policy transfer can even unexpectedly outperform a more low-level technique in certain cases, which can be explained by some simple intuitions on how each of the techniques provides their bias, how strong this bias is, and how it evolves over time.

In this chapter we have taken a first step towards making the case that the four knowledge injection techniques we identified in the previous chapter are all valid options in general for injecting a policy or value function into a TD process. Even though up to this point, we have only demonstrated this in the case of policy transfer, we have provided evidence on three different benchmark problems and in a number of different scenarios. Furthermore, we provided evidence from the literature in Section 3.4, and further evidence will be provided in the next chapter in the case of demonstrations. We notice that authors often make a mistake in publishing work proposing a 'new' technique, which is largely the same as other work, only replacing the knowledge injection technique used in that other work with a different one. We are for example guilty of this in our policy transfer paper [Brys et al., 2015b], where we propose a 'novel' policy transfer technique using shaping, as opposed to PPR in older work. We suggest that the injection techniques considered here are established to be interchangeable, although with different effects on performance (there is no such a thing as a free lunch) and using a different one compared to other work does not in itself constitute a contribution.

As stated before, we continue our investigation of the four knowledge injection techniques in the next chapter, in the context of learning from demonstration.

5 | Reinforcement Learning from Demonstration

> Very truly I tell you, the Son can do nothing by himself; he can do only what he sees his Father doing, because whatever the Father does the Son also does.
>
> Jesus Christ, *John 5:19*

Reinforcement learning and learning from demonstration are two common approaches applied to learning in control systems. As we have already seen, the former is typified by a reward signal, which provides a (usually) objective evaluation of the exhibited behaviour. The latter on the other hand relies on demonstrations provided by a different controller (expert or otherwise) to learn good behaviour. Of course, in absence of an objective evaluation of the behaviour, such as a reward signal, 'good' is hard to define, especially considering that potential demonstrators are often not that 'good.' Yet, learning from demonstration techniques are typically much more successful at learning behaviour in complex systems than reinforcement learning, simply because they are much more sample efficient. Reinforcement learning techniques typically require a prohibitively large number of behaviour experiences to learn objectively good behaviour as task complexity increases.

In this chapter we investigate what Schaal [1997] called Reinforcement Learning from Demonstration (RLfD), the natural intersection between these two fields. At this intersection, both a reward signal and demonstrations are assumed to be available for learning. The former is used as an objective evaluation of behaviour, while the latter are used as

important heuristic knowledge. When combined, the downsides of each individual approach can be mitigated, while their strengths result in more powerful techniques. Recall for example that AlphaGo, the reinforcement learning system that for the first time ever beat a human professional player, supposedly one of the strongest players in the history of Go, used both a reward signal and demonstrations to succeed.

As in the previous section, we investigate the four knowledge injection techniques. In this case the heuristic knowledge comes in the form of demonstrations. Furthermore, we develop a novel learning from demonstration technique based on Gaussian distributions that encode the demonstrations as a value function to inject into reinforcement learning. We compare this novel technique with the state-of-the-art in RLfD, investigating the properties of different variants of each by experimenting with different demonstration datasets. But we start out with a brief overview of Learning from Demonstration and Reinforcement Learning from Demonstration.

5.1 Learning from Demonstration

As in RL, an agent operating in a Learning from Demonstration (LfD) setting looks for a policy π that allows it to execute a task [Argall et al., 2009]. While LfD settings vary greatly, there is typically no ground truth available, no reward signal that allows the agent to evaluate its behaviour. Instead, the agent must rely on a number of teacher demonstrations of the task to derive a policy that reproduces and generalizes these demonstrations. The teacher providing these demonstrations can be a human expert, a suboptimal AI agent, a simple hand-coded agent, etc.

In essence, the LfD problem is a *supervised learning* problem, with the agent being provided labeled training data. This training data consists of the demonstrations in the form of a set of state-action pairs $\{(s_0, a_0), \ldots, (s_n, a_n)\}$.[1] The state features are the input features, and the actions are the class labels or output. The learner should approximate the function (the demonstrator's policy) that generated these samples, mapping states to actions, thus learning to replicate the demonstrator's behaviour. A straightforward application of classification or regression algorithms to such data in order to get a policy is entirely possible. Sammut et al. [2002] and Crick et al. [2011] for example have demonstrated that decision trees can be very successful at mimicking and generalizing demonstrator behaviour.

Alternatively, the agent can learn action plans by learning pre and post-conditions for actions, which typically requires further annotations by the teacher beyond the state-action

[1] Some techniques require these to be chronological sequences, although none of the techniques we consider have this requirement.

demonstrations [Nicolescu and Mataric, 2003]. Or the agent can infer transition and reward functions from the demonstration data and apply RL to the learned model [Abbeel et al., 2007]. This latter approach falls under the intersection between RL and LfD, which we discuss in the next section.

While LfD techniques can be very sample efficient, there are a number of factors that can limit the quality of the policy derived from demonstrations by LfD techniques. One is that demonstrations often do not cover the whole state space, and that generalizations to unseen states may be far from correct [van Lent and Laird, 2001; Nicolescu and Mataric, 2003]. Also, the capabilities of the demonstrator can significantly affect the demonstration quality and therefore the quality of the policies derived from these demonstrations [Atkeson and Schaal, 1997]. These problems are recognized in the community, and one of the proposed directions forward is to use demonstrations to guide an RL process [Argall et al., 2009]. The following section describes such combinations of RL and LfD.

5.2 Reinforcement Learning from Demonstration

As noted before, the setting we investigate in this chapter is one where both the ground truth (reward) and demonstrations are available. We refer to this intersection of RL and LfD as Reinforcement Learning from Demonstration, or RLfD, after Schaal [1997].[2]

In his groundbreaking paper, Schaal proposed two approaches to using demonstrations in a reinforcement learning setting that were later developed by other researchers. The first is the generation of an initial value-function for temporal difference learning by using the demonstrations as passive learning experiences for the RL agent. This was later developed by Smart and Kaelbling [2002] as described below. The second approach Schaal proposed was to derive an initial policy from the demonstrations and to use that to kick-start the RL agent. This approach was picked up and further developed by Taylor et al. [2011b]. In this chapter, we experiment with both these approaches alongside our own contribution, although the former approach's results are not included because they proved to be statistically indistinguishable from the baseline in the domains we investigate. The latter approach is included as the Human-Agent Transfer algorithm, or HAT, as proposed by [Taylor et al., 2011b].

HAT leverages transfer learning principles to combine demonstrations and RL. The algorithm derives a policy from the demonstrated samples using a simple classifier, much like some standard LfD techniques would do. This policy is then 'transferred' to the reinforcement learning process that learns on the environment's reward. The authors achieved this transfer by either initializing Q-values, probabilistically reusing the classifier's policy,

[2]Not to be confused with Robot Learning from Demonstration.

or using the extra action method (three of the four knowledge injection techniques we investigate in this thesis). AlphaGO [Silver et al., 2016] uses the same principle, although not in TD-learning but with Policy Gradient.

Smart and Kaelbling [2002] have pursued the first approach to RLfD proposed by Schaal. They propose to split learning into two phases. In the first phase, the demonstrator is in control, choosing actions, and the RL agent passively learns from the demonstrations. In the second phase, the RL agent is put in control of the system and continues learning. The tele-operation phase can be seen as an instance of probabilistic policy reuse with the reuse probability $\psi = 1.0$ during that whole phase.

Other research considering the intersection of LfD and RL focusses on learning a model of the environment (and possibly a reward function), and using RL on simulated experiences in that model to build a good policy [Abbeel and Ng, 2005; Argall et al., 2009]. Learning a model is non-trivial, and the quality of the policy derived from this model of course depends on the quality of the model itself. We are more interested in work that avoids this model learning phase, and learns in the actual environment or a simulation.

As a sidenote: we argue that most LfD settings can be turned into an RLfD setting. We recognize that in some domains, it can be challenging to define an informative reward signal for a task, but we believe that one can usually construct a very sparse signal that only gives non-zero feedback when the goal is reached or when the task can no longer be completed.Of course, such sparse reward signals typically make learning slow because it takes many experiences for the sparse information to propagate throughout the agent's representation of the state-space, but this is then alleviated by using the demonstrations as a bias for exploration, helping the agent find and propagate the sparse rewards much faster.

5.3 Constructing a Value Function from Demonstrations

Whereas the HAT algorithm first approximates the demonstrator's policy and then injects this policy into an RL process using various injection techniques, we believe that creating a value function to inject into the learning process may be equally valuable.

To create a value function based on demonstrations, we use the following intuition: we want the value $Q^D(s, a)$ of a state-action pair (s, a) to be high if action a was demonstrated in a state s^d similar to s, and we want the value to be low if this action was not demonstrated in the neighbourhood of s. To achieve this, we need a similarity metric for state-action pairs. In this work, we use a non-normalized multi-variate Gaussian to calculate similarity between state-action pairs. More precisely, assuming discrete actions, if two

state-action pairs differ in the action, their similarity is zero.[3] Otherwise, we calculate:

$$g(s, s^d, \Sigma) = e^{\left(-\frac{1}{2}(s-s^d)^T \Sigma^{-1}(s-s^d)\right)}$$

Similarity is 1 when $s = s^d$, and trails to zero as s and s^d get further apart. The covariance matrix Σ is crucial in defining the sphere of influence of demonstrated state-action pairs, and needs to be tailored for each domain. This could be automated by including metric learning techniques [Taylor et al., 2011a]. In this work, we always normalize the state space (map state variables to $[0, 1]$), and use Σ of the form $\Sigma = \sigma I$, i.e. the n-dimensional identity matrix times a constant σ, with n the number of state variables.

To calculate the value of a given state-action pair, we look through the set of demonstrations, and find the sample with the same action that yields the highest similarity [Brys et al., 2015a]:

$$Q^D(s, a) = \max_{(s^d, a)} g(s, s^d, \Sigma)$$

In other words, the value function is piecewise Gaussian, a landscape with mountain ranges along the demonstrated trajectories, and flat plains where no demonstrations were provided.

5.4 Experiments

This experimental section will serve to investigate the following questions:

- How do the four knowledge injection techniques perform in the context of demonstrations?
- How do the policy-based HAT and the value function-based Gaussian technique compare?
- What is the effect of having little/ample demonstration data?
- What is the effect of different types/quality of demonstrators?

Our first experiments are performed in the Cart Pole environment, and later on validated in super Mario.[4]

[3] This metric can easily be extended to continuous actions by including the action as an extra dimension in the multi-variate Gaussian.

[4] We set the σ parameter to 0.2 in Cart Pole and 0.5 in Mario, based on a few preliminary experiments with various σ. Furthermore, probabilistic policy reuse (PPR) is parametrized with $\psi = 0.99^e$, where e is the episode number, in both Cart Pole and Mario, and dynamic shaping with respectively $\beta = 2\alpha$ and $\beta = 10\alpha$.

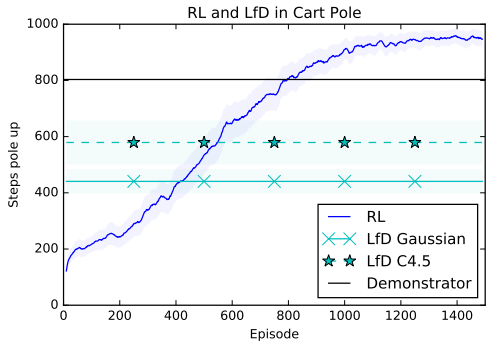
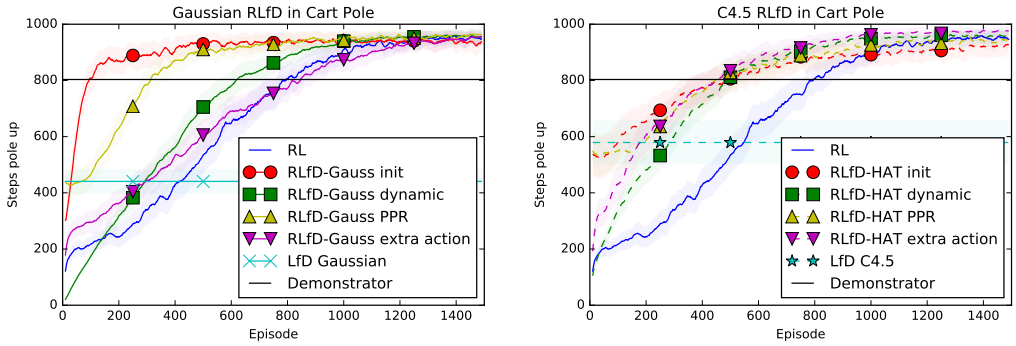


Figure 5.1: Comparing plain RL ($Q(\lambda)$-learning) and LfD (C4.5 and Gaussian) provided with a single demonstration. The LfD techniques are more sample efficient, but RL has the potential to eventually learn better policies than LfD techniques. Neither of the LfD techniques matches the performance of the demonstrator, while RL manages to outperform the demonstrator after about 800 episodes.

5.4.1 Initialization, Dynamic Shaping, PPR and Extra Action

In this first part, we validate our results with the four knowledge injection techniques from the previous chapter, by switching the context from policy transfer to learning from demonstration. We recorded 20 different demonstrations, each consisting of the state-action sequence of a single episode, provided by a high-performing RL agent that 5% of the time takes a random action instead of following its policy. The provided demonstrations have an average length of 803.3 ± 267.6 steps of balancing the pole. Figure 5.1 shows the comparison between only learning from rewards (RL – represented by our $Q(\lambda)$-learning agent), and only learning from demonstrations (LfD – represented by the C4.5 decision tree used in HAT and a policy derived from the Gaussian value function generated from demonstrations). In each of the 100 independent runs, the learning from demonstration techniques are provided only a single of the 20 demonstrations. Thus there are 5 runs per individual demonstration. Notice that neither of the LfD techniques manages to match the performance level of the demonstrator given only a single demonstration, while the RL baseline eventually outperforms the demonstrator. On the other hand, the LfD techniques are much more sample efficient than this RL agent: with only one episode worth of samples, they achieve a level of performance that the RL baseline only achieves after about $400 - 500$ episodes of learning.

In the following experiment, we look at the various possible combinations between RL and LfD, mediated through one of the four knowledge injection techniques we investigate.



(a) Gaussian (b) HAT

Figure 5.2: Comparing the performance of RL, LfD, and RLfD through the four knowledge injection techniques we investigate. All four injection techniques improve performance in both the HAT and Gaussian case, outperforming the plain RL and LfD techniques.

Algorithm	**Cumulative**	
Q-Learning	$1.0 \cdot 10^6 \pm 3 \cdot 10^4$	
	Gaussian	**HAT**
Initialization	$1.4 \cdot 10^6 \pm 1 \cdot 10^4$	$1.2 \cdot 10^6 \pm 5 \cdot 10^4$
Dynamic shaping	$1.1 \cdot 10^6 \pm 3 \cdot 10^4$	$1.2 \cdot 10^6 \pm 3 \cdot 10^4$
PPR	$1.3 \cdot 10^6 \pm 2 \cdot 10^4$	$1.2 \cdot 10^6 \pm 4 \cdot 10^4$
Extra action	$1.0 \cdot 10^6 \pm 4 \cdot 10^4$	$1.2 \cdot 10^6 \pm 3 \cdot 10^4$
LfD	$6.6 \cdot 10^5 \pm 7 \cdot 10^4$	$8.7 \cdot 10^5 \pm 1 \cdot 10^5$

Table 5.1: Performance of RL, LfD and RLfD techniques in Cart Pole, using either HAT (C4.5) or the Gaussian technique. All but the Gaussian extra action variant are significantly better than the baseline Q-Learning. All differences between HAT and Gaussian variants are also significant (e.g., Gaussian initialization is significantly better than HAT initialization).

Figure 5.2 and Table 5.1 present this combination on the one hand for the Gaussian technique, and on the other hand for HAT (using the C4.5 decision tree algorithm). This experiment provides further confirmation that all four injection techniques are valid options to provide heuristic knowledge to a temporal-difference reinforcement learning agent. All four techniques yield improvements over only using the reward signal in both the HAT and the Gaussian case. In the former case, the improvements are much larger with initialization and probabilistic policy reuse (PPR), while in the latter case, all techniques perform well, but not as good as with the Gaussian technique. This on the other hand also again shows that none of these techniques are always better than the others, and all can be considered valid options when designing a learning system.

Now given this basic conclusion that RLfD can work through the knowledge injection techniques we consider, we go deeper and analyse what effects can be observed for different demonstration datasets, differing in size, quality and demonstrator.

5.4.2 The Effect of Small/Large Amounts of Data

In the above experiments, the LfD and RLfD algorithms were fed only a single demonstration, consisting of 803.3 state-action samples on average. In this section, we investigate what the effect is of the size of this dataset (number of samples) on the performance of all techniques. To do that, we create new demonstration datasets of increasing sizes by sampling state-action pairs from the 20 demonstration trajectories (16066 samples in total) we had already generated for the previous experiment. For each size $\in \{1, 2, 5, 10, 20, 50, \ldots, 10000, 15000\}$, we create 20 new datasets, sampled (with replacement) from the combined 20 demonstration trajectories, i.e. from 16066 demonstrated samples.

In Figure 5.3, we plot the relationship between the demonstration dataset size and the cumulative performance of the pure LfD techniques over 1500 episodes (the C4.5 decision tree used by HAT and a policy derived from the Gaussian value function). The performance of plain RL is provided for comparison. For both LfD techniques, we notice an expected positive correlation between dataset size and performance. Larger datasets yield higher performance. If we look more closely, we see both techniques following a similar trend between dataset sizes 1 and 500. After that, the Gaussian technique plateaus, while HAT continues its seemingly linear trend (appears exponential on the logarithmic x-axis), indicating that the decision tree better leverages increasing amounts of data.

These results highlight an important difference between C4.5 and the Gaussian technique. The C4.5 decision tree takes in all demonstration samples and explicitly creates an approximation of the function that generated them. With more data, C4.5 can smooth out the inconsistencies present in the demonstrations (5% random moves), thus improving performance. This effect was observed by Sammut et al. [2002] when applying a C4.5 decision

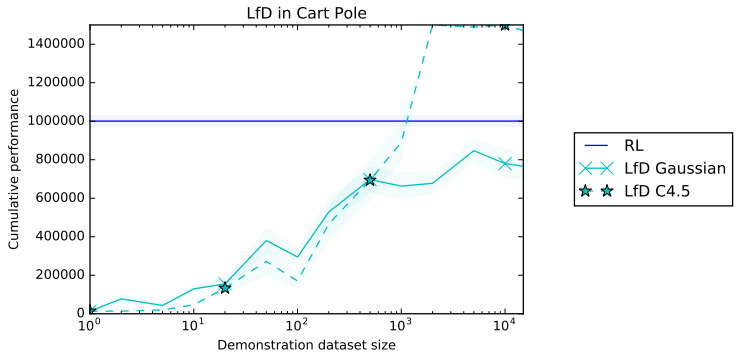
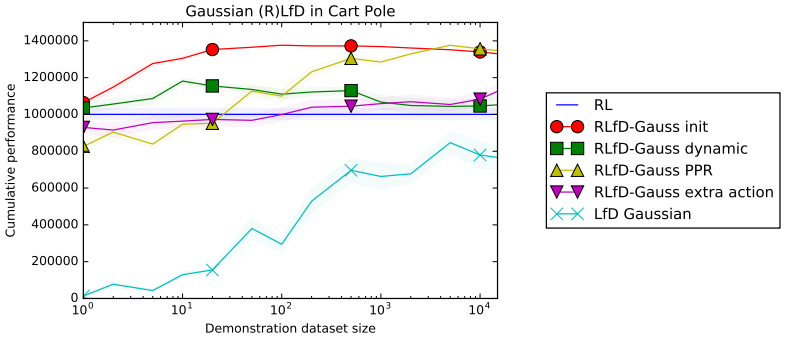


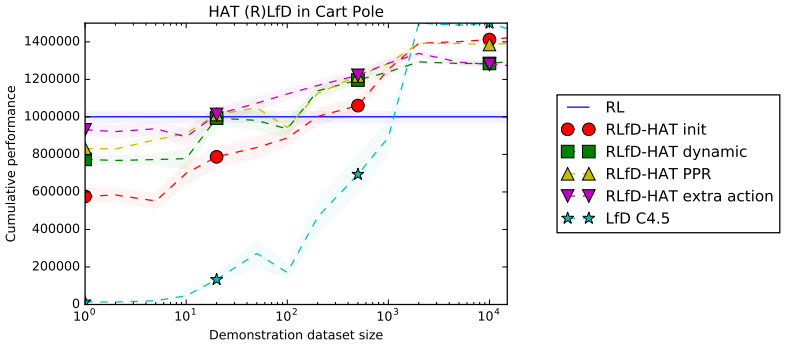
Figure 5.3: Comparing plain RL ($Q(\lambda)$-learning) and LfD (C4.5 and Gaussian) provided with demonstration datasets of increasing sizes. There is a clear correlation between dataset size and performance, which is better leveraged by the C4.5 decision tree than the Gaussian value function.

tree to flight trajectories supplied by human pilots. The many inconsistent and correcting actions taken by the pilots were "cleaned up" by the decision tree, resulting in much smoother control than that exhibited by the human demonstrators. The Gaussian value function on the other hand is similar to the 1-nearest neighbours classification algorithm, in that it stores and uses each provided sample individually. This lack of generalization seems to prevent the technique from better leveraging larger datasets that include noisy data.

Yet, while this is true for the pure LfD techniques, the conclusions are quite different when considering the RLfD techniques, see Figure 5.4. We see that even though the Gaussian technique by itself creates very ill-performing policies, the RL processes guided by the Gaussian technique perform much better. They even make surprisingly efficient use of very small datasets. When using Q-function initialization as injection technique for example, employing the Gaussian LfD technique with only 20 demonstrated samples yields a level of performance HAT only achieves with 2000 samples. A nearest neighbour style approach in this case makes much more efficient use of a limited set of samples than a decision tree. It provides the RL process with a few individual pointers as to what to do, which prove to be enough to quickly learn near-optimal policies. The decision tree on the other hand cannot learn a meaningful approximation of the demonstrator's policy with only few samples, thus not providing a useful bias. On the other hand, HAT is preferable for larger datasets as those allow the decision tree to generate better approximations and generalizations, which is reflected in all HAT variants performing very well with large datasets. In general, we observe a positive correlation between dataset size and performance, for both HAT and



(a) Gaussian



(b) HAT

Figure 5.4: Experimenting with different demonstration dataset sizes, ranging from 1 demonstrated sample to 15000, provided by a demonstrator whose demonstration performance is 803.3 ± 267.6. For both the Gaussian and the HAT variants, in general we observe performance increasing with demonstration data set size (significant for every technique except for the Gaussian dynamic shaping variant). The Gaussian variants are more effective than the HAT variants with smaller dataset sizes, while HAT manages to leverage large datasets better. Gaussian initialization achieves a speed of learning with 20 demonstration samples that HAT variants only achieve with 2000 samples. On the other hand, the C4.5 decision tree seems to generalize better, as evidenced by the much greater increase in performance it shows compared to the Gaussian technique.

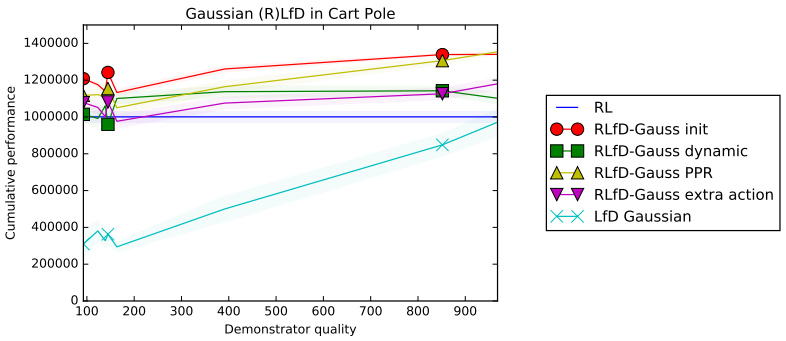
the Gaussian technique, as well as for all injection techniques, except for dynamic shaping with the Gaussian technique – this approach has a negative correlation with dataset size.

5.4.3 The Effect of Demonstration Quality

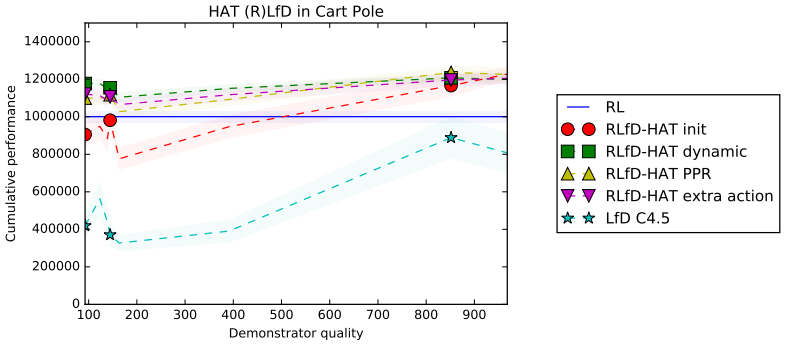
After investigating the robustness of algorithms against demonstration dataset size, we investigate their robustness against varying demonstration quality. We generate new sets of 20 demonstrations. As opposed to the previous demonstrations we worked with, this time each set of 20 is generated by a different RL agent of different ability (demonstrator performance $\in \{92.6,\ 123.2,\ 138.95,\ 144.5,\ 163.35,\ 391.2,\ 851.65,\ 968.2\}$). So for each level of demonstrator performance, we have 20 demonstrations. To account for the effect that the demonstration dataset size has on performance, we populate every set with exactly 1000 demonstrated samples. For every technique and demonstrator quality, we run 100 independent runs (as in every experiment in this thesis). In each of these runs, the techniques are provided only a single demonstration from the set of 20 for a given demonstrator quality.

Figure 5.5 shows the relationship between demonstration quality and the cumulative learning performance of each of the algorithmic variants we consider, for respectively the Gaussian technique and HAT. In both cases we can observe that all (R)LfD techniques benefit from increasing demonstration quality, an expected effect. Visually, this conclusion may appear weak or tentative, but it is corroborated by the Pearson correlation coefficients calculated between the demonstration quality and techniques' performance (average of 0.848 ± 0.068 over all techniques) as well as by the slopes of lines fitted to the data (average slope of 0.315 ± 0.143 for the demonstrator qualities normalized to $[0, 1]$). Although the correlation between demonstration quality and performance visually appears to be much less strong than the correlation between demonstration dataset size and performance, it is actually stronger. The average Pearson correlation coefficient when looking at demonstration dataset size is only 0.488 ± 0.221, compared to the 0.848 ± 0.068 for demonstration quality, and the average slope is only 0.002 ± 0.001 for the former compared to 0.315 ± 0.143 for the latter. The apparent discrepancy between the numbers and the figures is due to the log-scale on Figure 5.4 where we look at dataset size, and the linear scale on Figure 5.5 where we look at demonstration quality.

Taking the experiments in this section and the previous one together, we can make the simple conclusion that indeed *more* data by *better* demonstrators will quite likely yield better (R)LfD performance.



(a) Gaussian



(b) HAT

Figure 5.5: Experimenting with different levels of demonstration quality. Every (R)LfD technique improves its performance with increasing demonstration quality.

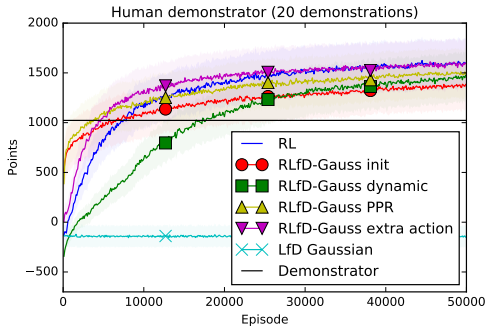
Algorithm	**Cumulative**	
Q-Learning	$6.6 \cdot 10^7 \pm 3 \cdot 10^5$	
	Gaussian	**HAT**
Initialization	$6.0 \cdot 10^7 \pm 3 \cdot 10^5$	$3.9 \cdot 10^7 \pm 2 \cdot 10^5$
Dynamic shaping	$5.1 \cdot 10^7 \pm 4 \cdot 10^5$	$7.2 \cdot 10^7 \pm 3 \cdot 10^5$
PPR	$6.6 \cdot 10^7 \pm 3 \cdot 10^5$	$5.6 \cdot 10^7 \pm 3 \cdot 10^5$
Extra action	$6.9 \cdot 10^7 \pm 3 \cdot 10^5$	$7.0 \cdot 10^7 \pm 3 \cdot 10^5$
LfD	$-7.1 \cdot 10^6 \pm 3 \cdot 10^4$	$-2.6 \cdot 10^7 \pm 3 \cdot 10^3$

Table 5.2: Performance of RL, LfD and RLfD techniques in Mario, using either HAT (C4.5) or the Gaussian technique to inject the human demonstration. Only the extra action and the HAT dynamic shaping variants are significantly better than the RL baseline.

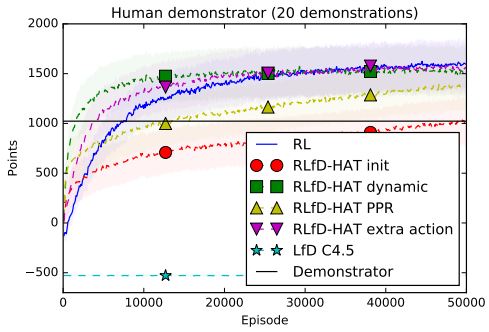
5.4.4 The Effect of Demonstrator Type

Above, we have considered RLfD in the Cart Pole problem with demonstrations provided by a reinforcement learning agent that has basically the same state representation as the student agent that receives the demonstrations – they observe the world in the same way. This commonality perhaps makes the student agent more likely to benefit from these demonstrations, while demonstrations provided by an agent with a very different perception of the world might be much less useful. To test this proposition, we turn to the super Mario problem. First, we recorded 20 human demonstrations, provided by the author of this thesis, who is reasonably well acquainted with the game. The demonstrations are recorded from the student agent's perspective (i.e., the student agent's state features, its view of the world, are registered when the human plays), even though the human demonstrator potentially operates based on very different state features than the observing reinforcement learning agent.

Then, we use these demonstrations in our RLfD setting to examine their effect on performance. Figure 5.6 and Table 5.2 show the results of an experiment comparing RL, LfD and RLfD using the human demonstrations. Plain RL outperforms the demonstrator after approximately 8000 episodes. The LfD techniques perform very badly, a far cry from the demonstrator's performance, quite likely due to the high dimensionality of the state space (27 dimensions) and the comparatively low number of samples covering that space (13560). Yet, again, despite the LfD policies being quite bad at executing the task, when combined with RL, they provide a good bias, in several cases allowing the RLfD agent to outperform plain RL overall, and in most cases to at least speed up learning significantly in the initial phase. The Gaussian technique outperforms plain RL overall when injected



(a) Gaussian



(b) HAT

Figure 5.6: Comparing RL, LfD and RLfD provided with 20 demonstrations by a human. Even though the LfD performance is very low, half of the RLfD manage to significantly outperform plain RL.

Demonstrator	Quality	Dataset size	Predictability (C4.5 accuracy)
Hand-coded	49.3	8019	32.0738
Human	1022.7	13560	60.7301
RL	1258.2	33983	34.037

Table 5.3: Different properties of the 20 demonstrations recorded for each of three different demonstrators. The demonstrator's level of performance, the demonstration dataset size, and the predictability of the demonstrator's actions, as measured by the cross-validated classification accuracy of a C4.5 decision tree trained on the dataset.

through the extra action method and HAT outperforms plain RL when combined with dynamic shaping or the extra action method. In light of these results and the above ones in this and the previous chapter, we stress again the importance of considering all four injection techniques when building a learning system that uses heuristic information. It remains unclear which factors favour one of the injection techniques over another, and each technique could potentially be beneficial or not.

Nonetheless, we see that even despite the obvious differences in demonstrator and student agent perception, the demonstrations proved useful for speeding up learning when injected through a suitable injection technique.

In the following experiment, we compare these demonstrations provided by a human, with demonstrations provided by a simple hand-coded agent,[5] and by a near-optimal RL agent that considers the same state-features as the student agent. These demonstrations are all very different, as can be seen in Table 5.3. Those provided by the **hand-coded agent** have no relationship between the state-features and the actions except that the action set is restricted to three of the twelve actions throughout the state space. This demonstrator therefore performs very badly overall in super Mario, since it usually can not achieve the high reward at the end of a level, although we believe that its behaviour can provide a useful bias. Furthermore, since this demonstrator is essentially randomly picking between three actions, its behaviour is no more predictable than random within that restricted set of three actions ($\frac{1}{3} \approx 33.3\%$). The demonstrations provided by the **human** are of much higher quality, but are provided from a different perspective (state features) than the student agent's perspective. Surprisingly, the human's demonstrations are very predictable, with a C4.5 decision tree achieving over 60% accuracy (fully random would be $\frac{1}{12} \approx 8.3\%$). Finally, the **RL demonstrator** is the best performing demonstrator, yet its demonstrations are far less predictable. Partially due to its $\epsilon = 0.05$ random actions, but more importantly, as we have seen in the previous chapter, due to the significant number of first-time state

[5] At every step, it randomly chooses between three options: going right, going right and jumping, and going right, jumping and pressing the speed/fire button.

visits, where even this good demonstrator can do no better than randomly selecting an action. The human does not suffer from this problem since he generalizes to unseen states.

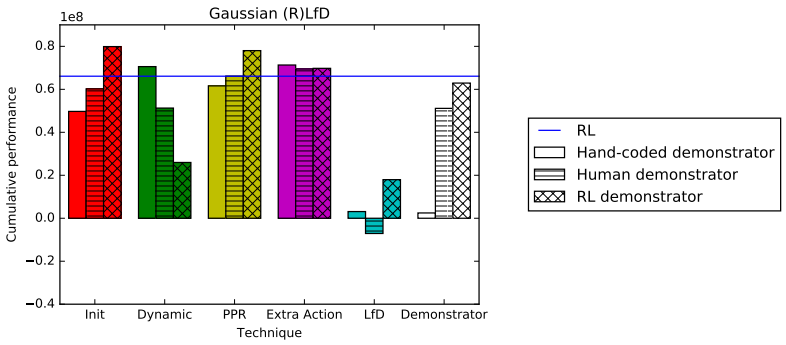
Based on these demonstration dataset properties and the conclusions drawn from the Cart Pole experiments in Sections 5.4.2 and 5.4.3, our hypothesis is that since the datasets by the hand-coded agent, the human and the RL agent in that order are of increasing size and quality, we predict that performance will be ranked accordingly.

Figure 5.7 shows a comparison of the cumulative performance observed for each technique when provided 20 demonstrations by either the hand-coded agent, the human, or the near-optimal RL agent. Plain RL's performance and that of the different demonstrators themselves is also provided for comparison. Indeed, as expected, the RL agent's demonstrations yield the best performance compared to the other two demonstrators' in general. Compared to the plain RL agent learning without demonstrations, using these demonstrations by a near-optimal RL agent yields better performance in every RLfD case except the Gaussian dynamic shaping one. When looking at each of the injection techniques across the different demonstrators, initialization and PPR their performance appears to be more affected by the properties of the different demonstration datasets (a larger difference in performance between using the hand-coded agent's or human's demonstrations and the RL agent's demonstrations), while the extra action method appears very robust to the differences in quality, size and demonstrator type, yielding approximately the same level of performance for each set of demonstrations. Dynamic shaping yields counter-intuitive results, appearing robust in the HAT case, but degrading performance with increasing dataset size and quality in the Gaussian case. More research on this technique is necessary to investigate its properties in practice.

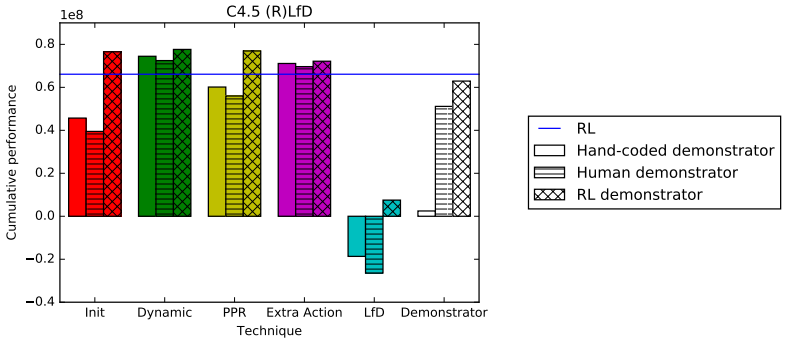
Considering the LfD techniques alone, it is interesting to note that the C4.5 decision tree achieved an impressive 60% classification accuracy on the human's demonstration data set while that decision tree yields the worst performance in terms of reward. A high classification accuracy of an LfD technique on a demonstration data set does not necessarily translate to good task performance with that dataset. Replicating a policy up to 95% accuracy is useless if that missing 5% keeps the agent from completing the task, or scoring high rewards. On the other hand, when combined with RL, this remaining 5% could be learned from exploration and observed rewards.

5.4.5 Policy Transfer vs Reinforcement Learning from Demonstration

As we noted in Chapter 3, demonstrations are a far cleaner and more applicable form of transfer of behaviour than explicit, full policy transfer. Indeed it is quite impossible for a human to transfer a full description of their policy for any household task to their



(a) Gaussian



(b) HAT

Figure 5.7: Experimenting with different levels of demonstration quality. In general, the RLfD agents benefit from higher quality demonstrations by an agent with the same perspective on the world.

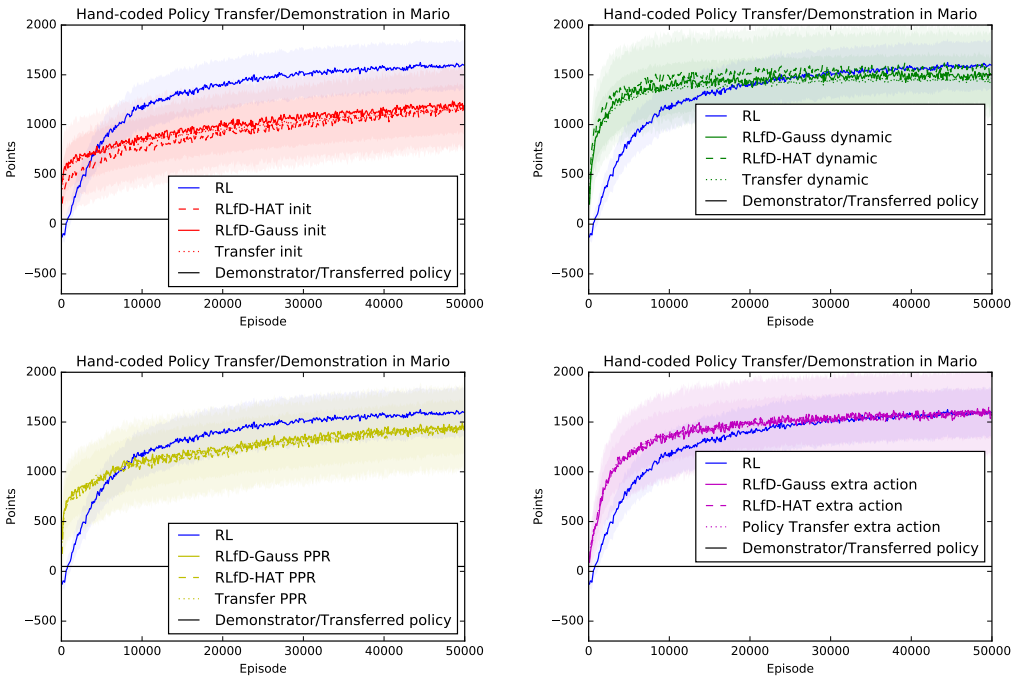


Figure 5.8: RLfD achieves performance similar to Transfer.

housekeeping robot, while they could easily give a few quick demonstrations of this policy. Yet it would seem that transferring the full policy (if possible) would be more beneficial than only transferring demonstrations, which are only samples from the policy. We put this to the test in the following experiment, either transferring the hand-coded policy discussed above, or using demonstrations provided by that policy. In Figure 5.8, we see that for each knowledge injection technique, performance is very similar when comparing policy transfer and demonstrations, despite the fact that in policy transfer the full policy is provided, and in the RLfD setting, only samples are provided. Of course, the results from this experiment should not be given too much weight, since the function (policy) generating this behaviour is very simple, and thus easy to approximate based on samples. With more complex policies, it is quite possible that policy transfer would indeed yield better results.

5.5 Summary

The research described in this chapter served to investigate the idea of Reinforcement Learning from Demonstration (RLfD). This involves the combination of the fields of Learning from Demonstration (LfD), where an agent must derive its own behaviour from demonstrations provided by another agent, and Reinforcement Learning (RL), where an agent must derive behaviour from rewards and interactions with the 'world'. Thus, in an RLfD setting, both demonstrations and rewards are available to the agent for learning. Using demonstrations as a heuristic bias for a process learning from rewards potentially leverages the advantages of both approaches. Using RL on the true reward signal, allows us to leverage the available theoretical guarantees for convergence and optimality, and thus we will still be learning to solve the actual task. The potential sub-optimality of demonstrations will furthermore not compromise this optimality, thanks to the theoretical guarantees associated with the knowledge injection techniques. And the problem of low sample efficiency in RL without prior knowledge is mitigated thanks to the exploratory bias introduced by the demonstrations.

In this chapter, we showed that indeed RLfD techniques (injecting the demonstrations through the knowledge injection techniques identified in Chapter 3) can outperform both plain LfD and RL in a variety of settings. We proposed a value-function based encoding of demonstrations as an alternative for existing policy-encodings of demonstrations. We analysed both types of approach, looking at the relation between performance and demonstration dataset size, quality, and demonstrator type. Interestingly, the value-function based approach, which is 1-Nearest Neighbour flavoured, yields impressive performance for small dataset sizes, while a decision tree-style approach performs better for large datasets. In general, we found that the intuition that more and better data yields better performance holds. Similarly, we found that agent performance was best when the demonstrator had a similar perception of the world.

In the next and final contribution chapter, we will investigate how one can incorporate multiple heuristics of unknown quality in the same reinforcement learning process.

6 | Ensembles of Shapings in Reinforcement Learning

> Where there is no guidance, a people falls, but in an abundance of counselors there is safety.
>
> King Solomon, *Proverbs 11:14*

In this final contribution chapter, we consider the question of how to handle multiple sources of information. How should a reinforcement learning agent that has several 'pieces' of heuristic information available, which all could potentially help in the learning process, combine these pieces of information such that they are maximally exploited? How can the agent do this without prior knowledge of how good each piece of information is?

This chapter develops a framework to achieve this within the context of reward shaping.[1] Thus we no longer consider the other injection techniques as we did in the previous chapters.

We start out with a general perspective to develop the ideas that are later on crystallized in the context of reinforcement learning and reward shaping.

[1] And therefore equivalently in the context of initialization, see Section 3.3.2

6.1 Introduction

When making decisions, especially decisions with a potentially high impact, it is considered good practice to solicit the advice of one or more third parties [Bonaccio and Dalal, 2006]. The idea is that when many parties agree, one can be more confident about the accuracy of the proposed decision. Conversely, if there is disagreement, the decision maker can at least take into account the fact that it is possible he does not know the best response, and take precautionary measures. This strategy is not only frequently used by humans, but is also implemented in automated decision makers. In the latter case, such approaches are most commonly referred to as ensemble techniques [Polikar, 2006]. A set of decision makers is combined into an ensemble (French for 'together'), and their advice or proposed decisions are aggregated into an actual decision using a specific strategy, such as majority voting. Ensembles derive their power from the diversity that is present in the set of their decision makers [Krogh et al., 1995], which allows the composite decision maker to outperform any single one of its components (clearly it would be pointless to combine many identical predictors). This diversity can originate from many sources: the decision makers can use different decision-making algorithms, they could have been trained on different inputs, be differently parameterized, etc. The reasoning is that due to their diversity, the different decision makers will make their mistakes on different inputs, and thus that the strategic combination of their advice will lead to a reduction of mistakes.

In this chapter, we argue for an as yet little researched way of generating this diversity. Instead of diversifying algorithm parameters or the training inputs fed into the algorithms, we create a diverse set of utility functions based on the original one. This process allows for the exploitation of structure found in the original utility function or the inclusion of domain knowledge, and yields several distinct ways to evaluate decisions for the ensemble to use. Although the idea of using diversity of evaluation in an ensemble could have general applicability, we develop it in the context of reinforcement learning, which is specifically amenable to the idea due to the distinct way an agent evaluates its behaviour (the accumulation of stochastic, delayed rewards over time, as opposed to an instantaneous deterministic objective function evaluation).

We first briefly discuss ensemble techniques and multi-objectivization, and then formally introduce multi-objective reinforcement learning. Following that, we look how the former ideas can be combined and integrated into reinforcement learning. We discuss and prove some theoretical properties of this approach, and experimentally show for three reinforcement learning problems that a significant boost in performance can be obtained.

6.2 Ensemble Techniques

As discussed before, a successful ensemble [Polikar, 2006] is built by providing two components: (1) a set of *diverse* decision makers, and (2) a strategy for the combination of their outputs such that the number of mistakes made is minimized. The choice of how to diversify the decision makers is crucial, as it is desirable that different decision makers are experts on different (overlapping) parts of the input space,[2] or put another way, that different decision makers make their mistakes on different inputs. We discuss two common ways of creating this required diversity: diversity of **input** and **algorithm**.

A lot of research on ensemble techniques has focused on classification and regression problems [Polikar, 2006]. Most successful techniques in those domains generate diversity by feeding different decision makers with different (possibly overlapping) subsets of the training data, i.e., **diversity of input**. Bagging [Breiman, 1996] is the most naive approach in this category, as it trains different classifiers on randomly sampled subsets of the training data, aggregating their output by majority voting. Boosting [Schapire, 1990] takes this idea one step further,[3] by intelligently selecting the subset of training data attributed to each classifier. It trains a first classifier with a random subset of the training data. Then, a second classifier is trained on data only half of which the first classifier can classify correctly. A third classifier is then trained on instances which the first two disagree on. This process can be applied recursively to build a strong ensemble. AdaBoost [Freund and Schapire, 1997] is an extremely popular and successful generalization of boosting, improving on boosting by including a more refined probabilistic training data sampling procedure, and weighted majority voting, based on classifiers' success during the training phase.

Diversity of algorithm refers to the actual algorithms used by decision makers being different, either inherently, or through different parameterization. Hansen and Salamon [1990] note that different neural network instances, trained on the full set of training data (in stark contrast to the 'diversity of input' approaches), will still be diverse due to different initializations of their weights and different sequencing of the training data. These networks end up in different local optima of the weights-space and therefore have different accuracy in different parts of the input space. In differential evolution literature, Mallipeddi et al. [2011] use ensembles of mutation strategies and parameters, positing that "different optimization problems require different mutation strategies with different parameter values" and "different mutation strategies with different parameter values may be better during different stages of the evolution." Their ensembles compared favourably

[2] Input is being used here as referring to such concepts as input features in classification, state in reinforcement learning, candidate solution in evolutionary computation, etc.

[3] Note that bagging and boosting have different goals: bagging aims to decrease variance in the models it produces, while boosting aims to decrease their bias [Bauer and Kohavi, 1999].

with the state-of-the art in differential evolution. Wiering and van Hasselt [2008] combine a variety of reinforcement learning algorithms with significantly different properties, showing how their ensemble requires less learning experiences to achieve equal or better performance than the single best algorithm.

These two ways to generate ensemble diversity have been researched extensively in different research areas and have yielded several powerful techniques. In this chapter, we argue for a third approach based on **diversity of evaluation**. In the following section, we describe *multi-objectivization*, the process we will use to achieve this diversity.

6.3 Multi-Objectivization

Multi-objectivization [Knowles et al., 2001] is the process of creating a variety of utility functions for a task, starting from a single one.[4] Formally, multi-objectivization takes single-objective problem p with utility function $u : X \to \mathbb{R}$, and outputs multi-objective problem $\hat{p}$ with utility function $\mathbf{u} = \{u_1, u_2, \ldots, u_n\} : X \to \mathbb{R}^n$, with $n > 1$. The idea is that $\hat{p}$ should be constructed in such a way that it is easier to solve than p. There are basically two approaches to create a diverse set of utility functions or 'objectives',[5] starting from a single-objective problem: either through **decomposition** of the original single objective, or through **addition** of extra objectives.

A prime example of multi-objectivization through **decomposition** is found in decision tree literature, where instead of using the total classification error to guide tree generation, tree performance is measured using the per-class misclassification, and trees are optimized using a multi-objective algorithm, which has been shown to lead to better trees in some cases [Fieldsend, 2009]. Another good example is the transformation of constrained optimization problems[6] into unconstrained ones, with the original objective being decomposed into an unconstrained objective, and several other objectives encoding the constraints [Coello Coello, 2000; Watanabe and Sakakibara, 2005; Saxena and Deb, 2007]. For example, Coello Coello [2000] gives empirical evidence that this form of constraint relaxation can outperform more standard penalty stratagems in Evolutionary Algorithms. This idea was also studied in reinforcement learning, where Raicevic [2006] assigned different reward signals to the sub-tasks resulting from a task decomposition.

[4] In this chapter, we are only concerned with problems that are originally single-objective. Actual multi-objective problems may also benefit from the approach described here. But, since optimality is defined differently in multi-objective problems, it is unclear whether applying multi-objectivization will have unwanted effects.

[5] Both utility function and objective will be used interchangeably throughout this chapter, and refer to the same concept.

[6] Constrained optimization problems are problems where one needs to optimize a given objective function with respect to a set of variables, given constraints on the values of these variables.

Multi-objectivization through the **addition** of objectives typically involves the incorporation of some heuristic information or expert knowledge on the problem. Jensen [2005] was one of the first to use what he calls 'helper' objectives next to the primary one. He investigated the job-shop scheduling and traveling salesman problems and found that additional objectives based on respectively time and distance-related intuitions help solve these problems faster. In genetic programming, where the goal is to evolve an accurate program to solve specific tasks, Bleuler et al. [2001] and de Jong et al. [2001] found that adopting program size as a second objective resulted in smaller *and* more accurate programs being evolved. This helper objective is a straightforward implementation of Occam's razor, and solves the common genetic programming problem of 'bloat', i.e., the tendency to evolve large overly-complex programs. The same principle of encoding minimization of model complexity for better generalization as an extra objective was succesfully used in decision tree research by Kim [2006].

In most of these examples, multi-objectivization is beneficial because the true utility function is not available; only approximations are. Consider the case of learning a decision tree to classify some data. We can not evaluate the true utility (accuracy) of a given decision tree, because that would involve testing it on all possibly relevant data in the universe. Therefore, one uses an approximation of this true utility, by measuring the tree's accuracy on the available training data, a subset of all possible data. Thus, it can be helpful to use heuristics, such as minimizing tree size, as further guidance that may improve the approximation of the true utility (or rather, create an order over decision trees using multiple objectives that is more like the true order over decision trees).

In temporal difference reinforcement learning, the true utility of an action is the expected, discounted, accumulated reward to be gained by taking an action and then following some behaviour. Since this utility is never directly given, but must be approximated based on observed rewards, reinforcement learning could benefit from the principle of multi-objectivization.

Before we describe our approach to multi-objectivization in reinforcement learning, we must give some preliminaries on multi-objective reinforcement learning itself.

6.4 Multi-Objective Reinforcement Learning

Multi-objective reinforcement learning [Roijers et al., 2013] (MORL) is a generalization of standard single-objective reinforcement learning, with the environment formulated as a multi-objective MDP, or MOMDP $\langle S, A, T, \gamma, \mathbf{R} \rangle$. The difference with the single-objective case is the reward function. Instead of returning a scalar value, it returns a vector of

scalars, one for each of the m objectives:

$$\mathbf{R}(s,a,s') = [R_1(s,a,s'), \ldots, R_m(s,a,s')]$$

Policies are in this case evaluated by their expected vector returns $\mathbf{Q}^\pi$:

$$\begin{aligned}\mathbf{Q}^\pi(s,a) &= [Q_1^\pi(s,a), \ldots, Q_m^\pi(s,a)] \\ &= \left[E\left\{\sum_{k=0}^{\infty} \gamma^k R_1(s_{t+k}, a_{t+k}, s_{t+k+1}) | s_t = s, a_t = a\right\}, \ldots, \right. \\ &\quad \left. E\left\{\sum_{k=0}^{\infty} \gamma^k R_m(s_{t+k}, a_{t+k}, s_{t+k+1}) | s_t = s, a_t = a\right\} \right]\end{aligned}$$

Since there are multiple (possibly conflicting) signals to optimize, there is typically no total order over policies. Policies may be incomparable, i.e., the first is better on one objective while the second is better according to another objective, and thus the notion of optimality has to be redefined. A policy π_1 is said to strictly Pareto dominate another policy π_2, i.e., $\pi_1 \succ \pi_2$, if for each objective, π_1 performs at least as well as π_2, and it performs strictly better on at least one objective. The set of non-dominated policies is referred to as the *Pareto optimal set* or *Pareto front*. The goal in multi-objective reinforcement learning, and multi-objective optimization in general, is either to find a Pareto optimal solution, or to approximate the whole set of Pareto optimal solutions.

With a multi-objective variant of Q-learning, Q-values for each objective can be learned in parallel, stored as Q-vectors [Gábor et al., 1998; Van Moffaert et al., 2013]:

$$\hat{\mathbf{Q}}(s,a) \leftarrow \hat{\mathbf{Q}}(s,a) + \alpha\boldsymbol{\delta}$$

$$\boldsymbol{\delta}_i = \mathbf{R}_i(s,a,s') + \gamma \max_{a'} \hat{\mathbf{Q}}_i(s',a') - \hat{\mathbf{Q}}_i(s,a)$$

The most common approach to derive a policy from these estimates is to calculate a linear scalarization, or weighted sum based on the estimated Q-vectors and a weight vector w [Vamplew et al., 2010; Roijers et al., 2013; Van Moffaert et al., 2013]:

$$\pi(s) = \arg\max_a w^T \hat{\mathbf{Q}}(s,a)$$

The weight vector determines which trade-off solutions are preferred, although setting these weights *a priori* to achieve a particular trade-off is hard and non-intuitive [Das and Dennis, 1997], often requiring significant amounts of parameter tuning.

6.5 Multi-Objectivization in Reinforcement Learning

In this section, we describe how multi-objectivization can be achieved in a reinforcement learning context. First, we describe CMOMDPs, a subclass of multi-objective MDPs that contain multi-objectivized MDPs. Then, we describe how an MDP can be transformed into a CMOMDP through multi-objectivization using reward shaping. In the section after this one, we will describe how ensemble techniques can be used to solve such a CMOMDP.

6.5.1 CMOMDP

Recall that multi-objective MDPs (MOMDPs) require the simultaneous optimization of multiple feedback signals. As conflicts may exist between objectives, there is in general a need to identify (a set of) trade-off policies. The set of optimal, i.e., non-dominated, incomparable policies is called the Pareto-front. Assuming all objectives are to be maximized, define the set $\mathcal{S}_p^*$ to contain all policies π of MOMDP p that are within ϵ_o of optimality for at least one objective o (with respect to the expected discounted cumulative reward $\mathbf{Q}^\pi$):

$$\pi \in \mathcal{S}_p^* \iff \exists o \in \mathcal{O}_p, \forall \pi' \in \Pi_p : \mathbf{Q}_o^\pi + \epsilon_o \geq \mathbf{Q}_o^{\pi'}$$

$\epsilon_o \geq 0$ defines the largest difference in utility of objective o that the system designer is indifferent about, $\mathcal{O}_p$ is the set of objectives in p and Π_p is the set of possible policies for p. $\mathcal{S}_p^*$ will include at least the extrema of the Pareto-front of p.

We identify MOMDPs with correlated objectives (CMOMDP) as a specific sub-class of MOMDPs, defined to contain those MOMDPs p whose set $\mathcal{S}_p^*$ (and by extension whose Pareto-front) is so small that trade-offs between solutions on the Pareto-front are negligible, i.e. there are no noteworthy conflicts between the objectives. By consequence, the system designer does not care about which of the very similar optimal policies is found, but rather how fast it is found (and perhaps how well it is approximated). Formally:

$$p \in \text{ CMOMDP} \iff \forall o \in \mathcal{O}_p : \max_{\pi \in \mathcal{S}_p^*}(\mathbf{Q}_o^\pi) - \min_{\pi' \in \mathcal{S}_p^*}(\mathbf{Q}_o^{\pi'}) \leq \epsilon_o$$

Thus, whether a multi-objective problem is contained in this class is to a certain extent subjective, depending on the system designer's preferences (ϵ_o). Such problems can be seen as providing multiple sources of information or feedback for the same basic single-objective problem, and intelligently combining such objectives may yield faster and better optimization. See Figure 6.1 for a visual comparison between an MOMDP and a CMOMDP.

An example of a CMOMDP is the traffic light control problem in our previous work [Brys et al., 2014c]. Popular metrics to quantify agents' behaviour in that setting are the average delay experienced by cars and the throughput of the system. In practice, optimizing either

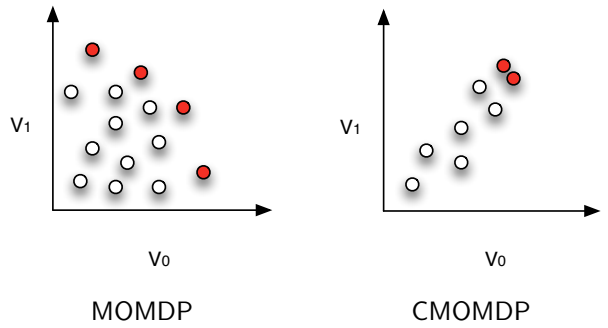


Figure 6.1: Representative examples of an MOMDP and a CMOMDP. Dots represent policies, and red dots are optimal policies. In the CMOMDP, the optimal policies are so similar that the system designer/user does not care about which one is found, whereas in the MOMDP, it matters which trade-off is found.

one of these results in optimizing the other, and combining these signals using a simple weighted sum was shown to result in faster learning.

6.5.2 Multi-objectivization

The relevance of this problem class may seem limited, as one does not encounter many such problems in the literature. But, we describe below how any single-objective MDP can be transformed (or multi-objectivized) into a CMOMDP using multiple reward shaping functions.

We reiterate that multi-objectivization [Knowles et al., 2001] is the process of turning a single-objective problem into a multi-objective problem in order to improve solving of the single-objective problem. Our goal is to turn a single objective RL problem into a CMOMDP, i.e., a multi-objective problem without (significant) conflicts between the objectives, in order to better solve the single objective RL problem. We desire furthermore the property that finding an optimal policy in the CMOMDP equates to finding a (near-) optimal policy in the original single-objective problem.

MDPs can be multi-objectivized by copying the reward function multiple times, which results in a CMOMDP with a single Pareto-optimal point. Of course, this modification in itself can not improve learning (unless other sources of diversity are present). But, these copies of the basic reward signal can be diversified by enriching them with heuristic knowledge using the potential-based reward shaping paradigm. Since potential-based reward

shaping is guaranteed to not alter the optimality of solutions [Ng et al., 1999], adding a different potential-based reward shaping to each of the copies of the reward signals keeps the problem a CMOMDP with a single Pareto optimal point. Yet, each of the different shaping functions provides a different evaluation of the behaviour during the learning process.

More formally: to turn MDP M into CMOMDP M′ using m reward shaping functions F_i, the reward vector $\mathbf{R}$ of M′ is constructed as follows:

$$\begin{aligned}\mathbf{R}(s,a,s') = \quad & [R(s,a,s') + F_1(s,a,s'), \ldots, \\ & R(s,a,s') + F_m(s,a,s')]\end{aligned} \tag{6.1}$$

where R is the reward function of M. Thus, we copy the base reward of M m times, and add a different shaping function to each.

We prove that this formulation preserves the total ordering, and thus also the optimality, of policies between M and M′, provided the shapings are potential-based. That is, that multi-objectivization by reward shaping does not introduce conflicts.

Theorem 1. *Let M be a given (finite, discrete) MDP, $M = (S, A, T, \gamma, R)$. We say that the MOMDP M′ is a shaping-based multi-objectivization of M, iff M′ $= (S, A, T, \gamma, \mathbf{R})$ with*

$$\begin{aligned}\mathbf{R}(s,a,s') = \quad & [R(s,a,s') + F_1(s,a,s'), \ldots, \\ & R(s,a,s') + F_m(s,a,s')]\end{aligned}$$

If all shaping functions $F_i,\ i = 1, \ldots, m$ are potential-based, as defined in Equation 3.1, we have the following properties:

- *Any policy π^* which is an optimal policy for M, is a Pareto optimal policy for M′.*
- *No other Pareto optimal policies for M′ exist, i.e., if π is not an optimal policy for M, π is not Pareto optimal in M′.*

Proof: The proof follows from the results by Ng et al. [1999]. There, Ng et al. proved that if a policy is optimal for an MDP with reward function R, it is also optimal for the shaped MDP with rewards $R + F$ (and vice versa), provided that F is a potential-based shaping function. So, any policy π^* that is optimal for MDP M will also be optimal for each of the shaped rewards $R + F_i$. Since π^* maximises the returns for all objectives, no policy which Pareto dominates π^* can exist (since such a policy would have to perform strictly better on at least one objective) and π^* must be part of the Pareto front for M′. Now suppose a policy π exists, which is part of the Pareto front of M′, but which is not optimal in M. Since π is suboptimal in M, according to Ng et al. [1999] it must also be suboptimal for each of the $R + F_i$ objectives. However, this means that any policy π', that is optimal in M,[7] will achieve a strictly higher return for all objectives. Thus, π' Pareto dominates π and π

[7] At least one such optimal policy must exist, see e.g., [Puterman, 2014].

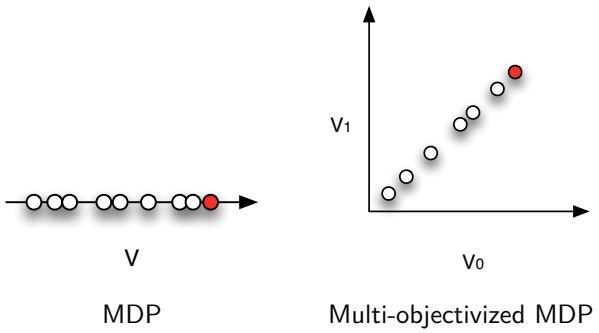


Figure 6.2: Representative examples of an MDP and multi-objectivized MDP. Dots represent policies, and red dots are optimal policies. Note that in the multi-objectivized MDP, the value functions of policies are fully correlated between the different 'objectives'. Recall furthermore that this correlation is not necessarily true for the immediate reward + shaping.

cannot be part of the Pareto optimal set for M′, which contradicts our original assumption.

Corollary. *Since all optimal policies of M′ (and thus also of M) achieve the highest expected return for each objective in M′, the Pareto front of M′ consists of a single point. Moreover, since Ng et al. [1999] actually prove that the total order of policies is preserved when using potential-based shaping, and not just optimality, MOMDP M′ also has a total order over all policies. These all lie on a single line in the multi-objective value-function space, see Figure 6.2.*

One may wonder what the purpose of this multi-objectivization is if all policies lie on the same line in the multi-objective space, i.e., if the objectives are all fully correlated. Why do we need extra objectives if they are correlated with all other objectives? The answer to that lies in the important distinction between return and reward. Rewards are generated at each time step for the state transition that occurred. Return is the discounted accumulation of these rewards. The returns (which measure the quality of a policy) of the objectives generated in the multi-objectivization we propose here are fully correlated, while the immediate rewards are not. These latter may differ wildly, and this heterogeneity in immediate rewards, combined with a guarantee of correlation of the total return, is the power of this proposed multi-objectivization.

Whether a multi-objectivization will actually be successful in making a problem easier to solve depends on the formulation of the shaping functions themselves. The best potential function[8] is V^* itself, the optimal value function of the problem.[9] While it is unlikely that the system designer can define a single potential function that equates V^* throughout the search space (which would amount to solving the problem), it is more likely that they can define several potential functions that correlate well with V^* in different parts of the state space, i.e., several rules of thumb for different situations,[10] or rules of thumb that weakly correlate with V^* throughout the state space. If these shapings are used to multi-objectivize an MDP, one can then attempt to strategically combine these signals, for which we will use ensemble techniques described in the next section.

6.6 Ensemble Techniques in Reinforcement Learning

Ensemble techniques have been introduced in Reinforcement Learning by Wiering and van Hasselt [2008]. Their approach involved several distinct reinforcement learning algorithms (agents) learning in parallel from the same experiences. The ensemble policy from which these experiences are generated is derived from a voting mechanism based on each agent's action preferences. Others have investigated the robustness of ensembles of neural network reinforcement learners, showing how these ensembles alleviate the problem of parameter tuning and are more stable [Hans and Udluft, 2010; Faußer and Schwenker, 2011] or how to select the best subset of ensemble agents for learning [Faußer and Schwenker, 2015]. In our multi-objectivization setting, the diversity required for an ensemble will not primarily come from the algorithms or their parameters, which may all be the same, but from the reward signal, which is enriched with a different shaping signal for each ensemble agent.

The different ensemble strategies we will describe below can all be formalized as calculating a weighted (w_i) sum of each ensemble agent's preferences (p_i), where the actual

[8] Recall that potential function is the term used in reward shaping literature to refer to the value function constructed to represent some knowledge. It is no different from the value functions we have been using throughout this thesis to represent knowledge, but we use the term potential function in this chapter to be consistent with the reward shaping context.

[9] $V^*(s) = \max_a Q^*(s, a)$.

[10] E.g., shaping using kinetic (speed) or potential energy (height) in the Mountain Car domain [Singh and Sutton, 1996], a problem where an underpowered car needs to build up momentum to climb a hill. These are opposite forces in this domain, as the car trades speed for height and vice versa, yet each is useful in a different situation: the car needs to focus on gaining speed when it can no longer gain height, and focus on gaining height when speed is already high.

choice for weight and preference functions will define the different strategies. The greedy ensemble policy then is:

$$\pi(s) = \arg\max_a \sum_i^n w_i(s) p_i(s,a) \tag{6.2}$$

The preference function $p_i(s,a)$ defines how high agent i values action a in state s. In the simplest case, $p_i(s,a) = Q_i(s,a)$, i.e., an agent's preference function is its estimated Q-function. The weight function $w_i(s)$ defines the relative importance of an agent in state s, i.e., how much an agent contributes to the ensemble decision relative to the other agents. Without loss of generality, we constrain w_i to: $\forall s : \sum_i^n w_i(s) = 1$. Without prior knowledge, it is advisable to set each agent's contribution to $w_i(s) = \frac{1}{n}$, independent of state.

Algorithm 1 describes the pseudocode for a greedy reward-function based ensemble of RL agents. At each step of the process, state s is observed, and action a is chosen given s, based on Equation 6.2. Action a is executed, next state s' is observed, and each agent i observes reward R_i, which in the shaping case amounts to $R_i = R + F_i$. These are used to update each learning agent.[11] Equation 6.2 gives a greedy policy; an ϵ-greedy, or soft-max policy can be defined analogously.

Algorithm 1 Greedy Ensemble of RL agents

```
1: procedure GREEDY-ENSEMBLE
2:     initialize s
3:     for each step of episode do
4:         a ← arg max_a Σ_i^n w_i(s) p_i(s,a)
5:         take action a, observe r, s'
6:         for each learner i do
7:             update agent i with (s, a, R_i(s,a,s'), s')
8:         end for
9:         s ← s'
10:    end for
11: end procedure
```

Algorithm 2 illustrates a specific implementation of this ensemble of RL agents:[12] a greedy ensemble of standard Q-learners using the linear ensemble technique. The agents' preference function is their estimated Q-function, and all agents have equal contribution.

[11] For on-policy learners, the ensemble needs to choose next action a' as well, before updating the learners.

[12] Note that we can refer to such an algorithm as an ensemble of RL agents or a single agent learning multiple value-functions or policies.

Note that each agent i learns independently, in that there is no mixing of values. Only at the action selection stage is there 'mixing' between the agents.

Algorithm 2 Greedy linear ensemble of Q-learners

1: **procedure** GREEDY-Q-ENSEMBLE
2: initialize s
3: initialize each learner
4: **for** each step of episode **do**
5: $a \leftarrow \arg\max_a \sum_i^n \frac{1}{n} Q_i(s,a)$
6: take action a, observe r, s'
7: **for** each learner i **do**
8: $Q_i(s,a) \leftarrow Q_i(s,a) + \alpha \left[R_i(s,a,s') + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a)\right]$
9: **end for**
10: $s \leftarrow s'$
11: **end for**
12: **end procedure**

Below, we describe four ensemble strategies, each defined by a different implementation of the weight and preference functions.

6.6.1 Linear

The most straightforward ensemble strategy (also often used in multi-objective reinforcement learning in general), is to calculate a linear combination of the different estimated Q-functions: $p_i(s,a) = Q_i(s,a)$, as in Algorithm 2, line 5.

In this ensemble, the weight function serves two purposes:

1. to weigh the relative importance of the different ensemble agents
2. to compensate for the differences in magnitude between the different agents' Q-functions

Thus, we decompose the weight function into two weight functions: $w_{i,ri}$ and $w_{i,sc}$, respectively used for the two goals described above. These are then combined in this way: $w_i(s) = \frac{w_{i,ri}(s) w_{i,sc}(s)}{\sum_i^n w_{i,sc}(s)}$. Unless prior information is available, which could come from a tuning phase prior to learning, or from expert knowledge on the efficacy of each reward shaping function,[13] the relative importance weight function is set to $w_{i,ri}(s) = \frac{1}{n}$, subject

[13] Marivate and Littman [2013] for example learn these weights in a limited setting.

to $\sum_i^n w_{i,ri}(s) = 1$. Although the agents are learning on the same base reward signal, their shaping signals may have wildly varying scalings and therefore result in estimated Q-functions of varying scalings. This can be compensated for using the scaling weights $w_{i,sc}$, but, without prior knowledge, we set these weights to: $w_{i,sc}(s) = 1$.

Importantly, in contrast to what we claimed in prior work [Brys et al., 2014a], this ensemble is not equivalent to a single learner learning with a single composite shaping (a reward shaping function $F(s, a, s') = \sum_i^n w_i(s)F_i(s, a, s')$), due to the non-linearity of the max operator employed [Roijers et al., 2013]. A single composite shaping is a much more computationally and memory efficient combination of shapings, since it does not require the storage of several value functions, but may result in the loss of information, as any dimensionality reduction technique risks.[14] We will also compare with this approach in the experimental section.

6.6.2 Majority Voting

Intuitively, in majority voting [Wiering and van Hasselt, 2008],[15] each agent casts a single vote, choosing its preferred action. Given equal weights, the greedy ensemble action is the one that received the most votes. Formally, an agent's preference function is this:

$$p_i(s, a) = \begin{cases} 1 & \text{if } a = \arg\max_b Q_i(s, b) \\ 0 & \text{else.} \end{cases}$$

In this case, the weight function only serves to weigh the relative importance of agents, since with voting, all agents' preference functions have the same codomain: $(0, 1)$. Again, unless prior information is available, we set the weight function to: $w_i(s) = \frac{1}{n}$.

6.6.3 Rank Voting

Rank voting [Wiering and van Hasselt, 2008] is a more fine-grained voting mechanism, where, as in majority voting, the lowest ranked action gets a score of 0, and the most preferred action a score of 1. In contrast, the in-between actions get a score proportional to their rank:

$$p_i(s, a) = \frac{\#\text{actions} - \text{rank(s,a)}}{n - 1}$$

[14] For example, if nine out of ten ensemble agents have attribute the highest expected return to action a, but the tenth believes this action will yield a very large negative return, in a linear combination, the fact that the majority agreed on action a will be lost, and another action will be deemed 'best' due to the effect of a single of the ensemble agents.

[15] In more recent work, van Hasselt preferred the term plurality voting [van Hasselt, 2011].

where $rank(s, a)$ outputs the index of (s, a) from $[1, \#\text{actions}])$ in the sorted list according to some quality criterion. Commonly, one will sort according to $Q(s, a)$.

The weight function is again used to weigh the relative importance of agents.

6.6.4 Confidence-based

Adaptive objective selection is a technique we previously introduced specifically to solve CMOMDPs [Brys et al., 2014b]. It builds on work on multi-objectivization in evolutionary computation where Jensen [2005] proposes to base every optimization decision on feedback from only a single of the several objectives [Jensen, 2005]. This is possible since each of the objectives alone can be used to solve the original MDP.

Specifically, at every step, this ensemble technique tries to measure how confident each of the ensemble agents is about its estimates for the current state, and uses the most confident agent's estimates alone to make an action selection decision. Thus, the preference function is simply the Q-function, as in the linear ensemble. The difference with that ensemble lies in the weight function $w_i(s)$, which incorporates the measure of confidence:

$$w_i(s) = \begin{cases} 1 & \text{if } i = \arg\max_{\text{agent}} \text{confidence}(s, \text{agent}) \\ 0 & \text{else.} \end{cases}$$

We define confidence as an estimation of the likelihood that the estimates are correct. Higher-variance reward distributions will make any estimate of the average reward less confident, and always selecting the agent whose estimates are most likely to be correct will maximize the likelihood of correctly ranking the action set. This is loosely inspired by the use of confidence intervals in UCB [Auer et al., 2002].

To measure confidence in estimates, we model every (s, a)-pair as a distribution, and not just a mean (Q-value). This allows us to determine how well each agent can differentiate between the actions based on common statistical tests. See Figure 6.3 for an illustration of this concept. Below, we describe an efficient way to represent these distributions and measure confidence for the case of tile-coding function approximation. For a discussion on how to represent these distributions in other cases, we refer the reader to our previous work [Brys et al., 2014b].

Tile-coding provides a very natural way to represent (s, a)-pairs as distributions, without requiring the storage of extra information. For a (s, a)-pair, agent i can simply take the weights in $\theta_{i,a}$ activated by s as samples representing the distribution of that pair. Then, we can estimate confidence by applying a paired statistical test to the samples of every action (or of the estimated best and worst actions). Such tests calculate a p-value which indicates how likely it is to observe the given estimates under the hypothesis that they come from the same distribution. The smaller the p-value, the more likely it is the distributions

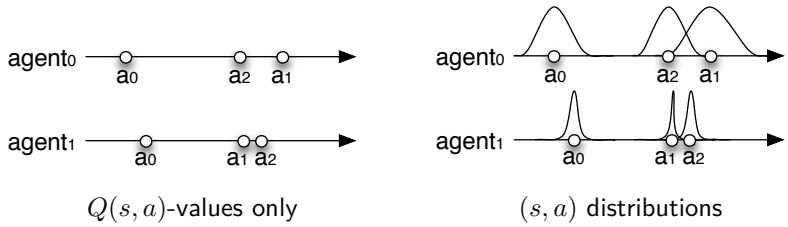


Figure 6.3: Showing two agents' estimates (top and bottom respectively). Determining which agent can be most confident about its estimates is impossible based on the estimated Q-values alone (left). Extra information is necessary (right). In this example, the second agent's estimates are deemed to be more reliable, as the actions' distributions are more significantly different/show less overlap.

are different, and that the agent can differentiate correctly between the actions. We can use a paired test, such as the paired Student's t-test or the Wilcoxon signed-rank test, because the weights for different actions will come from the same tiles in the same tilings, although stored in different weight vectors.

This confidence-based ensemble has several interesting properties. It makes its decisions a function of the state-space, which can account for different shapings being more or less reliable in different parts of the state space. Furthermore, it uses the different agents' estimates in a scale-invariant way. That is, its workings do not depend on the relative scalings of the shapings, since all statistical tests proposed are scale-invariant, and thus no parameters are introduced. This is a significant improvement over the simpler linear ensemble, which usually requires weight tuning, if only to align the magnitudes of the different shapings. Do note however that, despite using concepts such as distributions and statistical tests, there are no theoretical underpinnings to this confidence ensemble; it is no more than a heuristic.

6.7 Empirical Validation

We empirically demonstrate the usefulness of the proposed approach in the three domains we have been considering in this thesis: cart pole, the pursuit domain and Mario. All experiments are averaged over 100 trials for statistical significance, evaluated using the Student's t-test with $p = 0.05$. In the first two domains, we compared the following

approaches, mainly by measuring their cumulative performance (an indication of how fast they learn):[16]

1. vanilla $Q(\lambda)$-learning with pessimistic initialization
2. vanilla $Q(\lambda)$-learning with optimistic initialization
3. $Q(\lambda)$-learning shaped with a single shaping
4. $Q(\lambda)$-learning shaped with a composite shaping, i.e., a linear combination of all the shapings ($\Phi(s) = \sum_i^n \frac{1}{n}\Phi_i(s)$)
5. an ensemble of $Q(\lambda)$-learners, each shaped with a single of the shapings, combined with one of the four ensemble techniques
   (a) linear
   (b) majority voting
   (c) rank voting
   (d) confidence

We compare with optimistic initialization to analyse the effect of undirected uniform exploration, versus the more directed biased exploration that the shapings provide, which we hypothesize will be more effective. The shaping variants are always pessimistically initialized, so that exploration is almost exclusively driven by the shaping, besides the exploration induced by the action selection mechanism.

Furthermore, we perform two sets of experiments. One with normalized shapings, i.e., shapings with their potential function $\in [0, 1]$, and one with non-normalized shapings, to show how ensemble techniques are affected by the differences in magnitude between the shapings.

Lastly, we analyse the diversity induced by the different shaping signals.

6.7.1 Cart Pole

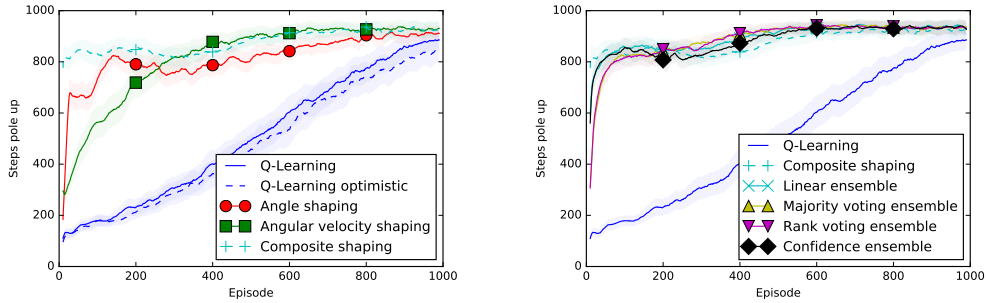
We transform Cart Pole into a CMOMDP by multi-objectivizing the problem using two potential-based shaping functions [Harutyunyan et al., 2015a]:

[16] In Mario, we only compared vanilla $Q(\lambda)$-learning with initialization set to 0, a composite shaping and linear and rank voting ensembles. Majority voting was left out to save time in favour of rank voting that always performed at least as good in previous experiments. The confidence ensemble was left out since in Mario we did not use tile-coding function approximation.

Angle encourages the agent to keep the angle of the pole close to 0, i.e., upright. Its potential function is defined as $\Phi_A(s) = -|\theta|$ (radians).

Angular velocity encourages the agent to keep the velocity of the pole low, since a fast moving pole is harder to control. Its potential function is defined as $\Phi_{AV}(s) = -|\dot{\theta}|$ (radians).

These shapings are scaled by 100 for optimal effect.[17] The angle is normalized by dividing by 0.20944 (the largest angle of the pole before failure in radians), and the angular velocity is normalized by dividing by 6, the maximum angular velocity.



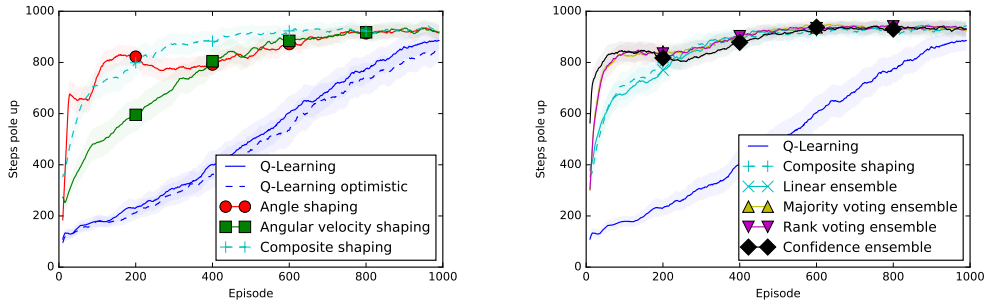
Single shapings Ensembles

Figure 6.4: Cart Pole - normalized shapings. The composite shaping outperforms the individual shapings, while the ensembles slightly outperform the composite shaping.

Figure 6.4 and Table 6.1 summarize the first experiment in Cart Pole with normalized shapings. As can be expected, both the angle and angular velocity shapings help the agent learn much faster than without this prior knowledge. Unguided, uniform exploration through optimistic initialization on the other hand results in slightly worse performance compared to the baseline vanilla $Q(\lambda)$, as the agent is encouraged to explore failure states too. Furthermore, a simple composite shaping (the average of the angle and angular velocity shapings) outperforms either of the individual shapings alone, while the ensembles slightly, but significantly, outperform this composite shaping. Arguably, in this case the composite shaping is preferable due to the lower computational and memory complexity, even though its sample efficiency is slightly worse than the ensembles.

[17] The shaping tuning problem has been addressed by Harutyunyan et al. [2015a], where the authors propose to include the same shaping with a number of different scalings in the ensemble. That is beyond the scope of this thesis.

In the case of non-normalized shapings (Figure 6.5 and again Table 6.1), the angular velocity shapings's performance drops significantly due to its much larger magnitude relative to the reward function. Consequently, the composite shaping and linear ensemble's performance drop too, since they are sensitive to the relative magnitudes of the different signals they are composed of. Importantly, the performance of the voting and confidence ensembles is unaffected, thanks to their scale-invariance. This suggests that the parameterless voting and confidence ensembles yield more robust combinations of shapings.



Single shapings Ensembles

Figure 6.5: Cart Pole - non-normalized shapings. The composite shaping and linear ensemble's performance is lower compared to the normalized shapings case, due to their sensitivity to the scaling of the signals they incorporate. The voting and confidence ensembles are scale-invariant and yield therefore similar performance to the normalized shaping case.

6.7.2 Pursuit Domain

We formulate a CMOMDP by multi-objectivizing the problem using three potential-based shaping functions:[18]

Proximity encourages a predator to move closer to the prey. Its potential function is defined as $\Phi_P(s) = -d(pred, prey)$, with d the Manhattan distance.

Angle encourages the predators to move to different sides of the prey, trapping it. It is defined to maximize the angle in radians between them and the prey to π: $\Phi_A(s) =$

[18] It has been proven that potential-based shaping in multi-agent RL does not alter the Nash Equilibria [Devlin and Kudenko, 2011].

Algorithm	Cumulative performance	
Q-Learning	$5.0 \cdot 10^5 \pm 3 \cdot 10^4$	
Q-Learning optimistic	$4.7 \cdot 10^5 \pm 3 \cdot 10^4$	
	Normalized	**Non-normalized**
Angle shaping	$8.1 \cdot 10^5 \pm 7 \cdot 10^3$	$8.3 \cdot 10^5 \pm 6 \cdot 10^3$
Angular velocity shaping	$8.2 \cdot 10^5 \pm 2 \cdot 10^4$	$7.7 \cdot 10^5 \pm 2 \cdot 10^4$
Composite shaping	$8.8 \cdot 10^5 \pm 3 \cdot 10^3$	$8.5 \cdot 10^5 \pm 1 \cdot 10^4$
Linear ensemble	$\mathbf{8.9 \cdot 10^5 \pm 2 \cdot 10^3}$	$8.6 \cdot 10^5 \pm 1 \cdot 10^4$
Majority voting ensemble	$8.8 \cdot 10^5 \pm 3 \cdot 10^3$	$\mathbf{8.8 \cdot 10^5 \pm 3 \cdot 10^3}$
Rank voting ensemble	$8.8 \cdot 10^5 \pm 3 \cdot 10^3$	$\mathbf{8.8 \cdot 10^5 \pm 3 \cdot 10^3}$
Confidence ensemble	$8.8 \cdot 10^5 \pm 3 \cdot 10^3$	$\mathbf{8.8 \cdot 10^5 \pm 3 \cdot 10^3}$

Table 6.1: Cart Pole - normalized. Averages of 100 trials of 1000 episodes each, measuring the number of steps the pole was up before falling. Those results not significantly different from the best are indicated in bold.

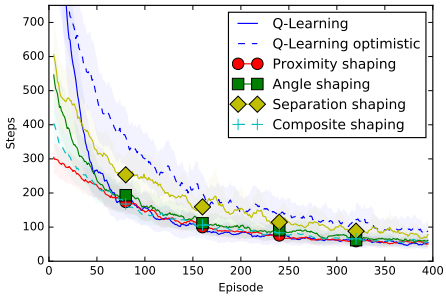
$\arccos(\frac{a \cdot b}{|a||b|})$, with a and b vectors pointing from the prey to the two predators respectively.

Separation encourages the predators to move away from each other. Its potential function is defined as $\Phi_S(s, a) = d(pred_1, pred_2)$ with d again the Manhattan distance.

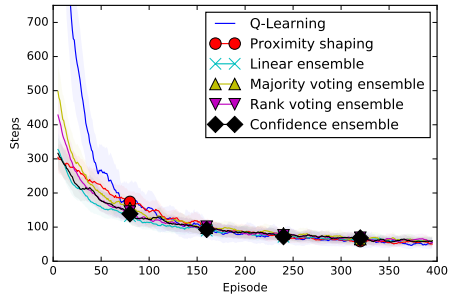
Proximity and Separation are normalized by dividing by $2 \times size$, with $size = 20$ both the width and height of the world; Angle is normalized by dividing by π. Furthermore, Proximity is implemented as $2 \times size - d(pred, prey)$, so that all shaping functions are positive, as theory indicates potentials should be when $\gamma < 1$ and there is no step-reward [Grześ and Kudenko, 2009].

The results of the first experiment (with normalized shapings) are shown in Figure 6.6 and Table 6.2. While the proximity and angle shapings improve performance compared to the baseline, the separation shaping appears not to be that useful and slightly degrades performance. Optimistic initialization results in much worse performance, due to the excessive exploration of unseen states far away from the prey. The composite shaping yields performance in-between the other shapings', not as good as the proximity shaping alone, showing that in this case the simple combination of signals is *not* better than the best of its constituting parts, unlike in the Cart Pole domain. The ensembles on the other hand outperform the composite shaping, and all but the majority voting one outperform the proximity shaping's performance.

When one cannot normalize the shapings' magnitudes in this domain (because the size of the world is not known for example, or infinite), the performance yielded by the individual

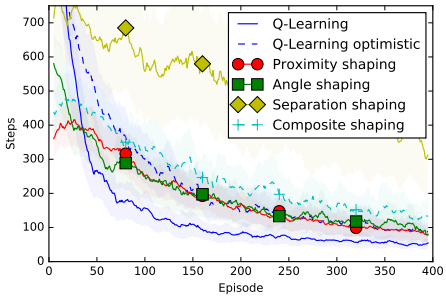


Single shapings

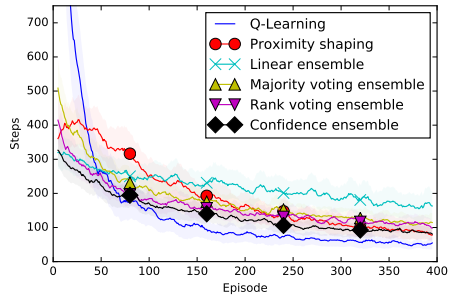


Ensembles

Figure 6.6: Pursuit domain - normalized shapings. The ensembles yield similar final performance and better cumulative performance than the individual or composite shapings.



Single shapings



Ensembles

Figure 6.7: Pursuit domain - non-normalized shapings. The individual and composite shapings, as well as the linear ensemble are most affected by the large differences in magnitude.

and composite shapings, as well as the linear ensemble is much worse, while the voting and confidence ensembles are again least affected due to their scale-invariance (Figure 6.7 and again Table 6.2). The fact that the performance of the ensembles still degrades is because they are scale-invariant across the different reward signals, but not within the different reward signals. That is, they are invariant to differences in scalings between the different reward+shaping signals, but they are not invariant to differences in scaling

Algorithm	**Cumulative performance**	
Q-Learning	$7.4 \cdot 10^4 \pm 3 \cdot 10^3$	
Q-Learning optimistic	$1.0 \cdot 10^5 \pm 2 \cdot 10^3$	
	Normalized	**Non-normalized**
Proximity shaping	$4.6 \cdot 10^4 \pm 6 \cdot 10^2$	$7.8 \cdot 10^4 \pm 1 \cdot 10^3$
Angle shaping	$5.6 \cdot 10^4 \pm 1 \cdot 10^3$	$8.2 \cdot 10^4 \pm 3 \cdot 10^3$
Separation shaping	$7.5 \cdot 10^4 \pm 2 \cdot 10^3$	$2.1 \cdot 10^5 \pm 9 \cdot 10^3$
Composite shaping	$4.9 \cdot 10^4 \pm 7 \cdot 10^2$	$1.0 \cdot 10^5 \pm 2 \cdot 10^3$
Linear ensemble	$\mathbf{4.1 \cdot 10^4 \pm 4 \cdot 10^2}$	$8.9 \cdot 10^4 \pm 1 \cdot 10^3$
Majority voting ensemble	$4.9 \cdot 10^4 \pm 6 \cdot 10^2$	$7.5 \cdot 10^4 \pm 7 \cdot 10^2$
Rank voting ensemble	$4.6 \cdot 10^4 \pm 4 \cdot 10^2$	$6.6 \cdot 10^4 \pm 7 \cdot 10^2$
Confidence ensemble	$4.3 \cdot 10^4 \pm 6 \cdot 10^2$	$\mathbf{5.7 \cdot 10^4 \pm 7 \cdot 10^2}$

Table 6.2: Pursuit domain. Averages of 100 trials of 400 episodes each, measuring the number of steps needed to catch the prey. Those results not significantly different from the best are indicated in bold. The ensembles yield better cumulative performance than the individual or composite shapings.

between the reward and shaping signals themselves, which is a different problem addressed elsewhere [Harutyunyan et al., 2015a].

6.7.3 Mario

Finally, we explore the ensembles and multi-objectivization using results from the previous chapters in Mario. We formulate the CMOMDP by creating shapings using five potential functions.[19]

Policy Transfer the policy transfer value function from Chapter 4 as a potential function.

Gaussian demonstrations the Gaussian value function generated from all 20 demonstrations provided by the RL demonstrator from Chapter 5 as a potential function.

HAT demonstrations as the previous, but processed by HAT.

Right a potential function encouraging Mario to go right ($\Phi(s) = s_4$), as finishing the level involves a significant amount of going right

[19] As a technical detail, in the Mario experiments of this section, we actually initialized the ensemble's Q-functions with the potential functions instead of shaping the reward functions. This is theoretically equivalent, see Section 3.3.2.

Up a potential function encouraging Mario going up ($\Phi(s) = s_5$), since being 'up' increases Mario's chances of dropping on enemies to kill them, or simply jumping over them.

The first three already yield values between 0 and 1; for the latter two, we normalize their output to be between 0 and 1.

Figure 6.8 and Table 6.3 show the results of an experiment comparing the Q-learning baseline with a composite shaping and a linear and rank voting ensemble. In this case, we observe the composite shaping performing best, with the two ensembles following close behind. Since we were considering signals of equal magnitude, this is not necessarily surprising, and it shows that a naive and computationally and memory efficient combination of heuristics can definitely be preferable.

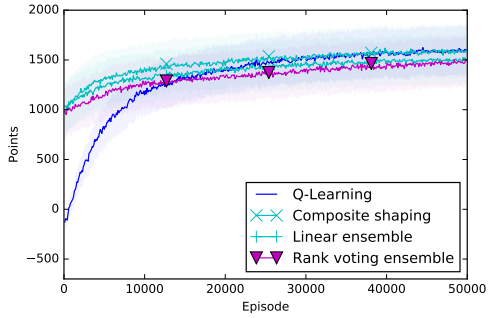


Figure 6.8: Mario. This domain serves as a counter example. The naive combination of shapings yields better performance than an ensemble.

Algorithm	**Cumulative performance**
Q-Learning	$6.6 \cdot 10^7 \pm 3 \cdot 10^5$
Composite shaping	$\mathbf{7.4 \cdot 10^7 \pm 7 \cdot 10^5}$
Linear ensemble	$7.0 \cdot 10^7 \pm 6 \cdot 10^5$
Rank voting ensemble	$6.7 \cdot 10^7 \pm 3 \cdot 10^5$

Table 6.3: Mario. Averages of 100 trials of 50.000 episodes each, measuring the points accumulated in the game. Those results not significantly different from the best are indicated in bold.

6.7.4 Diversity of shapings

Since our ensembles of shapings perform well in the experiments conducted, we can assume that the different shapings induced the diversity required for ensemble systems to work. In this section, we show that this diversity is indeed present and show how it evolves during the learning process. We measure diversity at a given time step of the learning process by calculating the Pearson correlation coefficient between Q-values. Specifically, in a given state, for each pair of agents in an ensemble, we calculate the Pearson correlation coefficient between the two agents' Q-values in that state. For every visited state-action pair, we average over all pairs of agents in the ensemble. A coefficient of 1 indicates full correlation (i.e., the agents fully agree on the relative quality of actions in that state), a coefficient of -1 indicates full anti-correlation (i.e., the agents have completely opposite ideas on the relative quality of actions in that state), and 0 indicates uncorrelated estimates (i.e., the agents disagree on the quality of actions, but in an inconsistent way). We hypothesize that full correlation (at least initially) and full anti-correlation are not useful for an ensemble, since with the former, there is no diversity, and with the latter, there is no agreement. We hypothesize that good trade-offs between diversity and agreement will yield a correlation coefficient around 0.

Figure 6.9 shows this analysis for the different ensembles used in Cart Pole, the Pursuit domain, with normalized shapings and in Mario. Observe that in all three domains, early in the learning process, the estimated Q-values of the different ensemble agents are indeed uncorrelated, or even slightly anti-correlated. As learning progresses, the ensemble agents' estimates become more and more correlated, as they converge to their optimal value functions. Here we see the interplay between rewards and returns in the learning process. Immediate rewards are uncorrelated or even anti-correlated due to the different shapings sometimes disagreeing on what are 'good' states, and thus the initial estimates are also uncorrelated. As learning progresses, and rewards are propagated through the value function representation, the estimates converge to the actual expected returns, which *are* correlated, independent of the amount of anti-correlation that may exist between the shapings themselves. The initial diversity makes the ensemble learn faster, and the later lack of diversity is a result of the guarantee that the ensemble learns the optimal policy.

6.8 Summary

This chapter introduced a novel perspective on diversity in ensemble techniques, where the necessary diversity is not derived from the differences in the algorithms themselves or their input data, but rather from the way they evaluate their own performance, i.e., diversity of evaluation. This allows for the use of expert knowledge to complement an

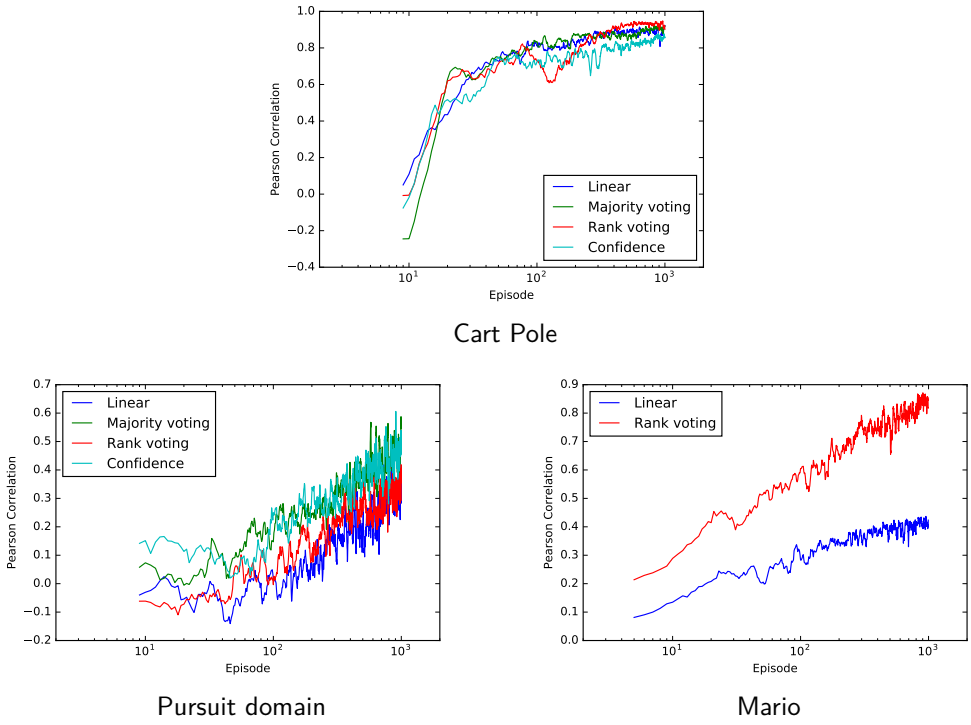


Figure 6.9: Pearson correlation coefficient between the Q-values of the different ensemble agents over time. We can observe the diversity induced by the different reward shaping signals early on, which diminishes as the learners converge to their optimal value functions (which are fully correlated).

otherwise uninformative objective function (think for example of the objective function in classification, which is usually only approximate due to the sheer size and unavailability of all relevant data). Yet, certain guarantees are required for this approach to not compromise optimality of solutions, since changing the objective function typically means changing the problem (and thus the solutions to the problem).

Therefore, we developed a theoretical framework in the context of reinforcement learning that provably provides these guarantees, thus allowing the safe incorporation of many

heuristics in a reinforcement learning process. It leverages the strong theory that underpins potential-based reward shaping and shows promising results in the experiments we conducted.

7 | Conclusions

> Now all has been heard, here is the conclusion of the whole matter.
>
> King Solomon, *Ecclesiastes 12:13*

This chapter summarizes the contributions laid out in this thesis, and proposes some directions for future research.

7.1 Contributions

The research described in this thesis deals with the situation where a reinforcement learning agent is provided with prior or external heuristic knowledge. Such knowledge could be used by the agent to explore its environment/task in a focused way, as opposed to a typically uniform exploration when no information is available. Such focused exploration in turn could yield much faster learning, in terms of number of interactions with the environment, which are usually costly – i.e., an increased sample efficiency.

Our first contribution in this context is an overview of what components could be present in such a reinforcement learning system with heuristic information, specifically in the context of temporal-difference learning, a popular form of reinforcement learning (see Chapter 3). We discussed several types of information and the sources that could provide this kind of information, describing amongst others demonstrations, teacher advice and behaviour transfer. We discussed two ways of encoding such information, as value functions and policies, and provided a way to translate between these two encodings.

Finally and most importantly, we discuss four common ways of injecting knowledge encoded either as a value function or policy into a temporal difference process, describing their theoretical properties: Q-value initialization, (dynamic) reward shaping, probabilistic policy reuse (PPR), and the extra action method. From a small sampling from the literature, we showed how different algorithms are built up using different components in this framework.

Our second contribution is the experimental validation of different instantiations of this framework (see Chapters 4 and 5). First in policy transfer, and second in a learning from demonstration setting. The main conclusion we can draw from this research is that in both settings, all four knowledge injection techniques are plausible options to use, as each of these can yield improved learning versus not injecting heuristic knowledge. Besides simply validating and comparing the performance of the different knowledge injection techniques in these settings, we analysed their robustness by varying several parameters of these settings. In **policy transfer**, we have looked at transferring policies of low and high quality (there is a clear positive correlation between the quality of the transferred policy and the learning agent's performance), at transfer in the context of a multi-agent system (transferring a policy to all agents is better than to a single agent), and at transferring deterministic and stochastic policies (stochastic policies provide a broader, more nuanced bias than deterministic policies, which is often beneficial), spread over three different benchmark problems.

The **Reinforcement Learning from Demonstration** setting combines demonstrations and rewards as sources of information for learning. We have argued that the fields of RL and Learning from Demonstration (LfD) can benefit from each other, with the rewards in RL providing a ground truth for learning, with potential guarantees of optimality for what is learned, and the demonstrations providing a useful bias for the otherwise slow (low sample efficiency) RL process. Since the demonstrations are only treated as a heuristic, and not as the 'truth,' as usually is done in a pure LfD setting where no other information is available, it is not problematic if the demonstrations are not of the highest quality, or if there are not enough demonstrations. We have shown this experimentally, by investigating demonstration datasets of varying sizes (showing an expected positive correlation between dataset size and learning performance), by looking at demonstration dataset quality (showing again an expected positive correlation between dataset quality, or demonstrator performance, and learning performance), and by looking at different demonstrator types and their effect on performance (the demonstrator with the same perception of the world as the student yielded the best performance). These experiments have confirmed the intuitions that indeed *more* and *better* demonstrations by demonstrators that have a perception of the world *similar* to the student agent's yield better performance. Furthermore,

we developed a value-function based encoding of demonstrations as opposed to the existing policy encoding provided by classification LfD algorithms. This encoding has proven to be preferable in settings where a limited number of demonstration samples is available.

An important take-away from this line of research (policy transfer and learning from demonstration) is that it remains unclear which factors favour one of the knowledge injection techniques yielding better performance over the others. Each technique outperforms the others in some settings. Therefore, when building a reinforcement learning system where one wants to inject such heuristic information, each of the injection techniques must be considered. We propose to consider these in the following order: PPR, initialization, extra action and dynamic shaping. PPR first since it has yielded the most consistently good performance over all experiments, followed by initialization. The extra action method also has performed consistently good, but due to its smaller bias, its effect on performance has typically been smaller. Finally, dynamic shaping is the most complex to implement, and has yielded mixed results, sometimes very good, but sometimes unstable, and therefore we consider it as a last option.

Our final and most important contribution of this thesis is the development of the ensembles of shapings framework (see Chapter 6). Building on work from the ensemble literature and from multi-objectivization literature, we propose a general machine learning approach, instantiated in reinforcement learning, that provides a systematic way of incorporating multiple pieces of heuristic knowledge in a learning process, in order to guide its learning without compromising convergence or optimality guarantees. In essence, the technique involves creating several reward signals, each augmented with a shaping function encoding a different piece of knowledge, learning a Q-function for each signal, and making ensemble decisions based on the different Q-functions.[1] We prove that this approach preserves convergence guarantees, and experimentally show that combining various pieces of knowledge in this way can outperform a more naive way of combining (a linear scalarization of the Q-functions encoding the knowledge), because it keeps the different pieces of knowledge of knowledge separate, combines them in a scale-invariant way, and allows for different combinations in different parts of the state space.

7.2 Future outlook

In general. All the techniques and results discussed in this thesis should be validated in more complex settings, in combination with more complex function approximators (i.e., deep neural networks), to prove their usefulness at the cutting edge of reinforcement

[1] Alternatively, given the equivalence between static reward shaping and initialization, it could be seen as different function approximators learning from the same reward signal but each approximator differently initialized.

learning.[2] We do not foresee any reasons why the research discussed in this thesis could not also be useful there, but we will not claim that indeed it does, until it has been shown to be so.

Knowledge injection. The knowledge injection techniques we have investigated were discussed and experimented with in the context of temporal difference learning methods. While temporal difference learning is an important and popular class of reinforcement learning algorithms, other classes, such as policy gradient algorithms are equally important, thanks to their many theoretical and empirical successes. An investigation of the different ways of biasing a policy gradient algorithm in a theoretically sound way would be useful for the field, along with an empirical validation of the techniques identified. Perhaps it will be possible to identify variants of the four techniques we discussed that work with policy gradient algorithms. AlphaGo is an example of a policy gradient algorithm initialized with values learned through a learning from demonstration technique.

An important issue when using techniques and algorithms, is their requirement for parameter tuning. Parameter tuning typically adds a large amount of overhead or (i.e. lowers sample efficiency). An important avenue for future research is looking into how parameters for probabilistic policy reuse and dynamic shaping could be automatically set, by linking them to other parameters or through on-line analysis of the agent's performance. If for example it can be determined that the policy being reused is not very good, policy reuse's reuse parameter ψ could be decayed much faster, to quickly remove the bad bias. Similarly with dynamic shaping, if it can be determined that the shaping function being learned is simple and deterministic, the learning parameter β could be increased significantly to speed up learning.

Policy Transfer. When transferring a policy, it is interesting to further investigate how to modulate the bias introduced by this policy, by introducing further stochasticity in the policy. We investigated this by first looking at transferring greedy deterministic policies, and later more random stochastic policies. We found in the Pursuit domain, that there is a sweet spot between fully random and fully greedy policies that yielded the best performance. It should be investigated (not only in the context of policy transfer, but in any heuristic setting where the knowledge is encoded as a policy) how one can find this sweet spot that provides enough focus on good information, but leaves enough randomness so that the environment is explored sufficiently too (a type of exploration-exploitation trade-off).

Reinforcement Learning from Demonstration. In this Reinforcement Learning from Demonstration (RLfD) setting, we have looked at two Learning from Demonstration (LfD) techniques to process and encode the provided demonstrations before injecting them into

[2] We have not done this ourselves, since our research was performed when the deep revolution in reinforcement learning was only beginning.

the reinforcement learning process that should benefit from these demonstrations. One of these techniques is a very straightforward application of a decision tree to build a policy, the other was the generation of a piecewise Gaussian value function. Many more LfD techniques exist that could be useful as the pre-processing and encoding component in an RLfD setting. Further investigations into RLfD should consider more complex LfD algorithms, especially as the tasks to be solved are of much higher complexity, in which case the sample efficiency of LfD would be even more important than in the simpler settings we have considered in this thesis.

Ensembles of shapings. In a general context, future work involves the investigation of our ensemble ideas in other machine learning fields, as well as optimization literature, where the idea of multi-objectivization has had some successes, but without strong theoretical guarantees.

In a reinforcement learning context, there remain several avenues for future research. Intra-signal scale-invariance, i.e., invariance to the relative magnitudes of reward and shaping signals is a vital avenue for research, since the magnitude of shapings still requires tuning in our ensemble approach. Harutyunyan et al. [2015a] recently started investigating this problem and provide a simple ensemble solution to this problem, which should be further investigated.

The ensembles' robustness against bad shapings is also an important issue. How is performance of the ensemble techniques related to the proportion of 'bad' shapings in the ensemble (do they degrade gracefully)? Can we build ensemble techniques that actively monitor the quality and usefulness of each heuristic and alters it's combination strategy accordingly?

A more challenging, but very interesting question is: how can we automatically generate shaping functions to be used in an ensemble? It should not be hard to generate many combinations of state features, and many thousands of value functions can be tractably learned in parallel [Sutton et al., 2011]. The difficulty will lie in building a shaping generator that satisfies conditions similar to that of a weak learner (a learner that generates classifiers that are merely better than random), as defined by Schapire [1990], in this case providing shapings that yield slightly better performance than without shaping. Lastly, it remains unclear how many of the ensemble algorithms such as AdaBoost that are very successful in supervised learning could be used in reinforcement learning.

List of Publications

International Journals

1. Tim Brys, Anna Harutyunyan, Peter Vrancx, Matthew E. Taylor and Ann Nowé (accepted). *Multi-objectivization and Ensembles of Shapings in Reinforcement Learning*. Neurocomputing.

2. Tim Brys, Tong T. Pham and Matthew E. Taylor (2014). *Distributed learning and multi-objectivity in traffic light control*. Connection Science, 26(1), pages 65-83. Taylor & Francis.

Book Chapters

1. Tim Brys, Yann-Michaël De Hauwere, Ann Nowé and Peter Vrancx (2012). *Local coordination in online distributed constraint optimization problems*. Multi-Agent Systems, Selected and Revised Papers of EUMAS-11, LNAI, Volume 7541, pages 31-47. Springer Berlin / Heidelberg. ISBN 978-3-642-34799-3.

Invited Surveys

1. Ann Nowé and Tim Brys (2016). *A Gentle Introduction to Reinforcement Learning*. In Proceedings of the Tenth International Conference on Scalable Uncertainty Management (SUM), pages 18-32.

Proceedings of Conferences with International Referees

1. Halit Bener Suay, Tim Brys, Sonia Chernova, Matthew E. Taylor (2016). *Learning from Demonstration for Shaping through Inverse Reinforcement Learning*. In Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pages 429-437.

2. Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E. Taylor and Ann Nowé (2015). *Reinforcement Learning from Demonstration through Shaping*. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pages 3352-3358.

3. Tim Brys, Anna Harutyunyan, Matthew E. Taylor and Ann Nowé (2015). *Policy Transfer using Reward Shaping*. In Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pages 181-188.

4. Kristof Van Moffaert, Tim Brys and Ann Nowé (2015). *Risk-Sensitivity Through Multi-Objective Reinforcement Learning*. In Proceedings of the IEEE Congress on Evolutionary Computation (IEEE CEC).

5. Silvio Rodrigues, Tim Brys, Rodrigo Teixeira Pinto, Ann Nowé and Pavol Bauer (2015). *Online Distributed Voltage Control of an Offshore MTdc Network using Reinforcement Learning*. In Proceedings of the IEEE Congress on Evolutionary Computation (IEEE CEC).

6. Ivomar Brito Soares, Yann-Michaël De Hauwere, Kris Januarius, Tim Brys, Thierry Salvant and Ann Nowé (2015). *Departure MANagement with a Reinforcement Learning Approach: Respecting CFMU Slots*. In Proceedings of the IEEE International Conference on Intelligent Transportation Systems (ITSC).

7. Tim Brys, Ann Nowé, Daniel Kudenko and Matthew E. Taylor (2014). *Combining Multiple Correlated Reward and Shaping Signals by Measuring Confidence*. In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI), pages 1687-1693.

8. Steven Adriaensen, Tim Brys and Ann Nowé (2014). *Fair-Share ILS: A Simple State-of-the-art Iterated Local Search Hyperheuristic*. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pages 1303-1310.

9. Steven Adriaensen, Tim Brys and Ann Nowé (2014). *Designing Reusable Metaheuristic Methods: A Semi-automated Approach*. In Proceedings of the IEEE Congress on Evolutionary Computation (IEEE CEC), pages 2969-2976.

10. Tim Brys, Anna Harutyunyan, Peter Vrancx, Matthew E. Taylor, Daniel Kudenko and Ann Nowé (2014). *Multi-Objectivization of Reinforcement Learning Problems by Reward Shaping*. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), pages 2315-2322.

11. Kristof Van Moffaert, Tim Brys, Arjun Chandra, Lukas Esterle, Peter Lewis and Ann Nowé (2014). *A Novel Adaptive Weight Selection Algorithm for Multi-Objective Multi-Agent Reinforcement Learning*. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), pages 2306-2314.

12. Tim Brys, Kristof Van Moffaert, Kevin Van Vaerenbergh and Ann Nowé (2013). *On the behaviour of scalarization methods for the engagement of a wet clutch*. In Proceedings of the 12th International Conference on Machine Learning and Applications (ICMLA), pages 258-263.

13. Tim Brys, Madalina M. Drugan and Ann Nowé (2013). *Meta-evolutionary algorithms and recombination operators for satisfiability solving in fuzzy logics*. In Proceedings of the IEEE Congress on Evolutionary Computation (IEEE CEC), pages 1060-1067.

14. Tim Brys, Madalina M. Drugan, Peter A.N. Bosman, Martine De Cock and Ann Nowé (2013). *Solving satisfiability in fuzzy logics by mixing CMA-ES*. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pages 1125-1132. **Best Paper Award** in IGEC/ESEP/BIO Track.

15. Tim Brys, Madalina M. Drugan, Peter A.N. Bosman, Martine De Cock and Ann Nowé (2013). *Local search and restart strategies for satisfiability solving in fuzzy logics*. In Proceedings of the IEEE Symposium Series on Computational Intelligence (IEEE SSCI), pages 52-59. **Nominated for Best Paper Award**.

16. Tim Brys, Yann-Michaël De Hauwere, Martine De Cock and Ann Nowé (2012). *Solving satisfiability in fuzzy logics with evolution strategies*. In Proceedings of the 31st Annual North American Fuzzy Information Processing Society Meeting (NAFIPS), pages 1-6. **Best Student Paper Award**.

Bibliography

Abbeel, P., A. Coates, M. Quigley, and A. Y. Ng
2007. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19:1.

Abbeel, P. and A. Y. Ng
2005. Exploration and apprenticeship learning in reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, Pp. 1–8. ACM.

Albus, J. S.
1981. *Brains, behavior, and robotics*. Byte books Peterborough, NH.

Amazon
2016. Amazon prime air. http://www.amazon.com/b?node=8037720011. Accessed 20/4/2016.

Ammar, H. B., E. Eaton, M. E. Taylor, D. C. Mocanu, K. Driessens, G. Weiss, and K. Tuyls
2014. An automated measure of mdp similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Argall, B. D., S. Chernova, M. Veloso, and B. Browning
2009. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

Atkeson, C. G. and J. C. Santamaria
1997. A comparison of direct and model-based reinforcement learning. In *International Conference on Robotics and Automation*. Citeseer.

Atkeson, C. G. and S. Schaal
1997. Robot learning from demonstration. In *International Conference on Machine Learning*, Pp. 12–20.

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

Baird, L.
1995. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the twelfth international conference on machine learning*, Pp. 30–37.

Bauer, E. and R. Kohavi
1999. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1-2):105–139.

Benda, M., V. Jagannathan, and R. Dodhiawala
1986. On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, WA, USA.

Bertsekas, D. P.
1995. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA.

Bhattacharya, R. and E. C. Waymire
2007. *A basic course in probability theory*. Springer Science & Business Media.

Bleuler, S., M. Brack, L. Thiele, and E. Zitzler
2001. Multiobjective genetic programming: Reducing bloat using spea2. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 1, Pp. 536–543. IEEE.

Bonaccio, S. and R. S. Dalal
2006. Advice taking and decision-making: An integrative literature review, and implications for the organizational sciences. *Organizational Behavior and Human Decision Processes*, 101(2):127–151.

Bou Ammar, H., D. C. Mocanu, M. E. Taylor, K. Driessens, K. Tuyls, and G. Weiss
2013. *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part II*, chapter Automatically Mapped Transfer between Reinforcement Learning Tasks via Three-Way Restricted Boltzmann Machines, Pp. 449–464. Berlin, Heidelberg: Springer Berlin Heidelberg.

Bou Ammar, H., K. Tuyls, M. E. Taylor, K. Driessens, and G. Weiss
2012. Reinforcement learning transfer via sparse coding. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, Pp. 383–390. International Foundation for Autonomous Agents and Multiagent Systems.

Breiman, L.
1996. Bagging predictors. *Machine learning*, 24(2):123–140.

Browne, C. B., E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton
2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Brys, T., A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé
2015a. Reinforcement learning from demonstration through shaping. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Pp. 3352–3358.

Brys, T., A. Harutyunyan, M. E. Taylor, and A. Nowé
2015b. Policy transfer using reward shaping. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, Pp. 181–188. IFAAMAS.

Brys, T., A. Harutyunyan, P. Vrancx, M. E. Taylor, D. Kudenko, and A. Nowé
2014a. Multi-objectivization of reinforcement learning problems by reward shaping. In *International Joint Conference on Neural Networks (IJCNN)*, Pp. 2315–2322. IEEE.

Brys, T., A. Nowé, D. Kudenko, and M. E. Taylor
2014b. Combining multiple correlated reward and shaping signals by measuring confidence. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, Pp. 1687–1693.

Brys, T., T. T. Pham, and M. E. Taylor
2014c. Distributed learning and multi-objectivity in traffic light control. *Connection Science*, 26(1):1–19.

Caruana, R.
1995. Learning many related tasks at the same time with backpropagation. *Advances in neural information processing systems*, Pp. 657–664.

Coello Coello, C. A.
2000. Treating constraints as objectives for single-objective evolutionary optimization. *Engineering Optimization*, 32(3):275–308.

Crick, C., S. Osentoski, G. Jay, and O. C. Jenkins
2011. Human and robot perception in large-scale learning from demonstration. In *Proceedings of the 6th international conference on Human-robot interaction*, Pp. 339–346. ACM.

Das, I. and J. E. Dennis
1997. A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. *Structural optimization*, 14(1):63–69.

de Jong, E., R. Watson, and J. Pollack
2001. Reducing bloat and promoting diversity using multi-objective methods. Pp. 11–18.

Degris, T., P. M. Pilarski, and R. S. Sutton
2012. Model-free reinforcement learning with continuous action in practice. In *2012 American Control Conference (ACC)*, Pp. 2177–2182. IEEE.

Devlin, S. and D. Kudenko
2011. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, Pp. 225–232. IFAAMAS.

Devlin, S. and D. Kudenko
2012. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, Pp. 433–440. International Foundation for Autonomous Agents and Multiagent Systems.

Devlin, S., D. Kudenko, and M. Grześ
2011. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(02):251–278.

Efthymiadis, K. and D. Kudenko
2013. Using plan-based reward shaping to learn strategies in Starcraft: Broodwar. In *IEEE Conference on Computational Intelligence in Games (CIG)*, Pp. 1–8. IEEE.

Fachantidis, A., I. Partalas, M. E. Taylor, and I. Vlahavas
2015. Transfer learning with probabilistic mapping selection. *Adaptive Behavior*, 23(1):3–19.

Faußer, S. and F. Schwenker
2011. Ensemble methods for reinforcement learning with function approximation. In *Multiple Classifier Systems*, Pp. 56–65. Springer.

Faußer, S. and F. Schwenker
2015. Selective neural network ensembles in reinforcement learning: Taking the advantage of many agents. *Neurocomputing*, 169:350–357.

Fernández, F. and M. Veloso
2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, Pp. 720–727. ACM.

Fieldsend, J. E.
2009. Optimizing decision trees using multi-objective particle swarm optimization. In *Swarm Intelligence for Multi-objective Problems in Data Mining*, Pp. 93–114. Springer.

Freund, Y. and R. E. Schapire
1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Gábor, Z., Z. Kalmár, and C. Szepesvári
1998. Multi-criteria reinforcement learning. In *ICML*, volume 98, Pp. 197–205.

Google
2016. Google self-driving car project. https://www.google.com/selfdrivingcar/. Accessed 20/4/2016.

Grześ, M. and D. Kudenko
2009. Theoretical and empirical analysis of reward shaping in reinforcement learning. In *International Conference on Machine Learning and Applications*, Pp. 337–344. IEEE.

Hans, A. and S. Udluft
2010. Ensembles of neural networks for robust reinforcement learning. In *Ninth International Conference on Machine Learning and Applications (ICMLA)*, Pp. 401–406. IEEE.

Hansen, L. K. and P. Salamon
1990. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001.

Harutyunyan, A., T. Brys, P. Vrancx, and A. Nowé
2015a. Multi-scale reward shaping via an off-policy ensemble. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, Pp. 1641–1642. IFAAMAS.

Harutyunyan, A., S. Devlin, P. Vrancx, and A. Nowé
2015b. Expressing arbitrary reward functions as potential-based advice. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Jensen, M. T.
2005. Helper-objectives: Using multi-objective evolutionary algorithms for single-objective optimisation. *Journal of Mathematical Modelling and Algorithms*, 3(4):323–347.

Kaelbling, L. P., M. L. Littman, and A. W. Moore
1996. Reinforcement learning: A survey. *Journal of artificial intelligence research*, Pp. 237–285.

Karakovskiy, S. and J. Togelius
2012. The Mario AI benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67.

Kim, D.
2006. Minimizing structural risk on decision tree classification. In *Multi-Objective Machine Learning*, Pp. 241–260. Springer.

Kim, H. J., M. I. Jordan, S. Sastry, and A. Y. Ng
2004. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems 16*, S. Thrun, L. K. Saul, and B. Schölkopf, eds., Pp. 799–806. MIT Press.

Klopf, A. H.
1972. Brain function and adaptive systems: a heterostatic theory. Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA.

Knowles, J. D., R. A. Watson, and D. W. Corne
2001. Reducing local optima in single-objective problems by multi-objectivization. In *Evolutionary Multi-Criterion Optimization*, Pp. 269–283. Springer.

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

Konidaris, G. and A. Barto
2006. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, Pp. 489–496. ACM.

Konidaris, G. and A. G. Barto
2007. Building portable options: Skill transfer in reinforcement learning. In *International Joint Conference on Artificial Intelligence*, Pp. 895–900.

Krogh, A., J. Vedelsby, et al.
1995. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, 7:231–238.

Lazaric, A.
2008. *Knowledge transfer in reinforcement learning*. PhD thesis, Politecnico di Milano.

Liao, Y., K. Yi, and Z. Yang
2012. Cs229 final report reinforcement learning to play mario. Technical report, Stanford University, USA.

Loftin, R., B. Peng, J. MacGlashan, M. L. Littman, M. E. Taylor, J. Huang, and D. L. Roberts
2016. Learning behaviors via human-delivered discrete feedback: modeling implicit feedback strategies to speed up learning. *Autonomous Agents and Multi-Agent Systems*, 30(1):30–59.

Mallipeddi, R., P. N. Suganthan, Q.-K. Pan, and M. F. Tasgetiren
2011. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11(2):1679–1696.

Marivate, V. and M. Littman
2013. An ensemble of linearly combined reinforcement-learning agents. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*.

Michie, D. and R. Chambers
1968. Boxes: An experiment in adaptive control. *Machine intelligence*, 2(2):137–152.

Minsky, M.
1961. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al.
2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Ng, A. Y., D. Harada, and S. Russell
1999. Policy invariance under reward transformations: Theory and application to reward

shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, volume 99, Pp. 278–287.

Nicolescu, M. N. and M. J. Mataric
2003. Natural methods for robot task learning: Instructive demonstrations, generalization and practice. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, Pp. 241–248. ACM.

Polikar, R.
2006. Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE*, 6(3):21–45.

Puterman, M. L.
2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Raicevic, P.
2006. Parallel reinforcement learning using multiple reward signals. *Neurocomputing*, 69(16):2171–2179.

Ravindran, B. and A. G. Barto
2003. Relativized options: Choosing the right transformation. In *International Conference on Machine Learning*, Pp. 608–615.

Roijers, D. M., P. Vamplew, S. Whiteson, and R. Dazeley
2013. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113.

Rummery, G. A. and M. Niranjan
1994. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering.

Russell, S.
1998. Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, Pp. 101–103. ACM.

Russell, S. and P. Norvig
1995. Artificial intelligence: A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27.

Sammut, C., S. Hurst, D. Kedzier, and D. Michie
2002. Learning to fly. *Imitation in animals and artifacts*, P. 171.

Saxena, D. K. and K. Deb
2007. Trading on infeasibility by exploiting constraint,Äôs criticality through multi-objectivization: A system design perspective. In *IEEE Congress on Evolutionary Computation*, Pp. 919–926. IEEE.

Schaal, S.
1997. Learning from demonstration. *Advances in neural information processing systems*, 9:1040–1046.

Schapire, R. E.
1990. The strength of weak learnability. *Machine learning*, 5(2):197–227.

Selfridge, O. G., R. S. Sutton, and A. G. Barto
1985. Training and tracking in robotics. In *International Joint Conference on Artificial Intelligence*, Pp. 670–672. Citeseer.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al.
2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Singh, S., T. Jaakkola, M. L. Littman, and C. Szepesvári
2000. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308.

Singh, S. P. and R. S. Sutton
1996. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158.

Skinner, B. F.
1938. *The behavior of organisms: An experimental analysis*. Appleton-Century.

Skinner, B. F.
1951. *Science and human behavior*. Simon and Schuster.

Smart, W. D. and L. P. Kaelbling
2000. Practical reinforcement learning in continuous spaces. In *International Conference on Machine Learning*, Pp. 903–910. Citeseer.

Smart, W. D. and L. P. Kaelbling
2002. Effective reinforcement learning for mobile robots. In *IEEE International Conference on Robotics and Automation*, volume 4, Pp. 3404–3410. IEEE.

Song, J., Y. Gao, H. Wang, and B. An
2016. Measuring the distance between finite markov decision processes. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, Pp. 468–476. International Foundation for Autonomous Agents and Multiagent Systems.

Stone, P. and M. Veloso
2000. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383.

Suay, H. B., T. Brys, S. Chernova, and M. E. Taylor
2016. Learning from demonstration for shaping through inverse reinforcement learning. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Pp. 429–437.

Sutton, R.
2016. The future of AI. https://www.youtube.com/watch?v=pD-FWetbvN8. Accessed 28/6/2016.

Sutton, R. and A. Barto
1998. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press.

Sutton, R. S., D. A. McAllester, S. P. Singh, Y. Mansour, et al.
1999. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, Pp. 1057–1063.

Sutton, R. S., J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup
2011. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, Pp. 761–768. IFAAMAS.

Taylor, M. E.
2008. *Autonomous inter-task transfer in reinforcement learning domains*. ProQuest.

Taylor, M. E., N. Carboni, A. Fachantidis, I. Vlahavas, and L. Torrey
2014. Reinforcement learning agents providing advice in complex video games. *Connection Science*, 26(1):45–63.

Taylor, M. E., B. Kulis, and F. Sha
2011a. Metric learning for reinforcement learning agents. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 22

Taylor, M. E. and P. Stone
2007. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th International Conference on Machine Learning*, Pp. 879–886. ACM.

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

Taylor, M. E., P. Stone, and Y. Liu
2007. Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167.

Taylor, M. E., H. B. Suay, and S. Chernova
2011b. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems*, Pp. 617–624.

Thorndike, E. L.
1911. *Animal intelligence: Experimental studies*. Macmillan.

Thorndike, E. L. and R. S. Woodworth
1901. The influence of improvement in one mental function upon the efficiency of other functions. ii. the estimation of magnitudes. *Psychological Review*, 8:247–261.

Thrun, S.
1996. Is learning the n-th thing any easier than learning the first? *Advances in neural information processing systems*, Pp. 640–646.

Thrun, S. and L. Pratt
2012. *Learning to learn*. Springer Science & Business Media.

Torrey, L. and M. E. Taylor
2012. Help an agent out: Student/teacher learning in sequential decision tasks. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS-12)*.

Tsitsiklis, J. N.
1994. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202.

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

van Hasselt, H. P.
2011. *Insights in reinforcement learning*. Hado van Hasselt.

van Lent, M. and J. E. Laird
2001. Learning procedural knowledge through observation. In *Proceedings of the 1st international conference on Knowledge capture*, Pp. 179–186. ACM.

Van Moffaert, K., M. M. Drugan, and A. Nowé
2013. Scalarized Multi-Objective Reinforcement Learning: Novel Design Techniques. In *2013 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. IEEE.

Watanabe, S. and K. Sakakibara
2005. Multi-objective approaches in a single-objective optimization environment. In *IEEE Congress on Evolutionary Computation*, volume 2, Pp. 1714–1721. IEEE.

Watkins, C. J. C. H.
1989. *Learning from delayed rewards.* PhD thesis, University of Cambridge.

Whiteson, S., B. Tanner, M. E. Taylor, and P. Stone
2011. Protecting against evaluation overfitting in empirical reinforcement learning. In *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, Pp. 120–127. IEEE.

Wiering, M. and M. Van Otterlo
2012. Reinforcement learning. *Adaptation, Learning, and Optimization*, 12.

Wiering, M. A. and H. van Hasselt
2008. Ensemble algorithms in reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(4):930–936.

Wiewiora, E.
2003. Potential-based shaping and Q-value initialization are equivalent. *Journal of Artificial Intelligence Research (JAIR)*, 19:205–208.

Wiewiora, E., G. Cottrell, and C. Elkan
2003. Principled methods for advising reinforcement learning agents. In *International Conference on Machine Learning*, Pp. 792–799.