Policy Transfer using Reward Shaping

Tim Brys
Vrije Universiteit Brussel
timbrys@vub.ac.be

Anna Harutyunyan
Vrije Universiteit Brussel
aharutyu@vub.ac.be

Matthew E. Taylor
Washington State University
taylorm@eecs.wsu.edu

Ann Nowé
Vrije Universiteit Brussel
anowe@vub.ac.be

ABSTRACT

Transfer learning has proven to be a wildly successful approach for speeding up reinforcement learning. Techniques often use low-level information obtained in the source task to achieve successful transfer in the target task. Yet, a most general transfer approach can only assume access to the output of the learning algorithm in the source task, i.e. the learned policy, enabling transfer irrespective of the learning algorithm used in the source task. We advance the state-of-the-art by using a reward shaping approach to policy transfer. One of the advantages in following such an approach, is that it firmly grounds policy transfer in an actively developing body of theoretical research on reward shaping. Experiments in Mountain Car, Cart Pole and Mario demonstrate the practical usefulness of the approach.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Miscellaneous

General Terms

Algorithms, Performance

Keywords

Reinforcement Learning; Transfer Learning; Reward Shaping

1. INTRODUCTION

Reinforcement learning is a paradigm that allows an agent to learn how to control a system in order to achieve specific goals. The agent is guided by reward/punishment received for the behaviour it exhibits, adjusting its behaviour in order to maximize the cumulative reward. In complex tasks, or tasks with sparse rewards, learning can be excruciatingly slow (as many learning algorithms take the *tabula rasa* approach), and the agent cannot do better than behaving randomly until feedback is received.

A lot of research in this domain is therefore dedicated to speeding up the learning process, relying on the incorporation of various pieces of external knowledge. Particularly the use of knowledge transfer [20], i.e. transferring knowledge learned in a previous (source) task, has received a lot of attention in recent years, thanks to the publication of a few seminal papers that propose simple and intuitive techniques, achieving impressive improvements in learning [7, 11, 19]. Yet, many of these techniques involve the use of low level information obtained in the source task, which may not be transferrable to or incompatible with the agent learning in the new task, as the algorithms used in source and target task may differ in many ways. In the most basic case, one can only assume access to the learned behaviour, or policy, in the source task, which is the output of any reinforcement learning algorithm (and other techniques, such as learning from demonstration). This has lead to *policy transfer*, i.e. the use of an old policy to speed up learning in a new task, without any reference to a low level algorithm-specific representation of that policy [6, 22].

In this paper, we investigate the use of reward shaping to achieve policy transfer. Reward shaping is another popular approach used to speed up reinforcement learning. Shaping means modifying the basic reward signal with some extra reward to bias the agent's exploration. Potential-based reward shaping is a form that is firmly grounded in theory [3, 13] and has had many empirical successes [2, 4, 5]. While many heuristics can be easily formulated as potential functions (e.g. height in Mountain Car), it is a lot harder to define behaviour-based heuristics in that form without losing information (e.g. go left when the pole is leaning left in Cart Pole). However, recent research in shaping has lead to the development of a technique that allows any reward function to be transformed into a potential-based shaping function, therefore allowing any non-potential-based shaping function to benefit from the theoretical guarantees of potential-based shaping [8]. We exploit these results to achieve policy transfer through shaping, providing a theoretical basis for this approach to transfer. The connection between reward shaping and transfer learning is not an unnatural one, as these techniques are not only similar in purpose, but often are so on a technical level too, as e.g. static potential-based reward shaping is shown to be equivalent, given the same experiences, to Q-value initialization [24], which is exactly what many transfer learning techniques do [19, 21].

In the following sections, we provide the reader first with preliminaries on reinforcement learning, reward shaping and transfer learning, followed by an exposition of the proposed approach to policy transfer. In the experimental section, we show how this approach compares to state-of-the-art in policy transfer on three benchmark problems.

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May, 4–8, 2015, Istanbul, Turkey.*

2. PRELIMINARIES

In this section we describe the body of research that is relevant to the work presented in this paper.

2.1 Reinforcement Learning

Reinforcement learning (RL) [17] is a paradigm that allows an agent to optimize its behaviour while operating in a given environment. It is rewarded or punished for the behaviour it exhibits, and its aim is to maximize the accumulated reward over time, which by definition amounts to solving the task. More formally, the environment is defined as a Markov Decision Process (MDP) $\langle S, A, T, \gamma, R\rangle$. $S = \{s_1, s_2, \ldots\}$ is the set of states the environment can be in, and $A = \{a_1, a_2, \ldots\}$ is the set of actions the learning agent can execute. Executing action a when the environment is in state s makes it transition to state s' with probability $T(s'|s, a)$, yielding $R(s, a, s')$ as reward for that transition. Finally, γ, the discounting factor, defines how important future rewards are. The goal is to learn a policy π that probabilistically maps states to actions in such a way that the expected discounted cumulative reward, the return, is maximized.

Reinforcement learning algorithms either directly search the policy space to find a policy that maximizes the return, or estimate the expected returns and derive a policy from those. The learning algorithms used in this paper are of the second type, and more specifically temporal-difference (TD) learning algorithms. These estimate state (V) or state-action (Q) value functions, that represent the return expected while following some behaviour policy. Algorithms such as Q-learning incrementally update these estimates based on the rewards observed while the agent is interacting with the environment:

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

with α the learning rate, and δ the temporal-difference error:

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

Given certain conditions, such as the agent's exploration and learning rate going to zero, Q-learning is guaranteed to converge to the optimal values Q^*, from which the optimal policy π^* can easily be derived:

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

The basic form of this type of algorithm operates on a table that stores the values for every (s, a) pair. In continuous and/or very large state-spaces, the tabular approach is impractical or even impossible, and function approximation techniques are required. With linear function approximation techniques, such as the popular tile-coding [1], states are represented using a feature vector ϕ_s, and the Q-function is approximated using a set of weights θ:

$$Q(s, a) = \theta_a^T \phi(s)$$

The weight vector is updated using an update-rule similar to the one used in the tabular case:

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

2.2 Reward Shaping

Reward shaping provides the agent with an extra reward signal F that is added to the environment's reward R, making the agent learn on the composite signal $R_F = R + F$. The shaping reward F usually encodes some kind of heuristic knowledge, and is intended to complement the typically sparser signal R. Since the agent's goal is defined by the reward function (solving the task optimally means finding a policy that achieves the maximum accumulated reward in expectation), changing the reward signal may actually change the task. Ng et al. [13] proved that the only sound way to shape without changing the task is through potential-based shaping. That is, define a potential function Φ over the state space, and define F as the difference between the potential of states s' and s, given observed transition (s, a, s'):

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s)$$

This formulation preserves the total order over policies, and therefore the optimality of policies.

This result was extended to shaping over state-action pairs (s, a) by Wiewiora et al. [25]:

$$F(s, a, s', a') = \gamma\Phi(s', a') - \Phi(s, a)$$

This allows for the incorporation of more specific information, pertaining to both states and actions. Devlin et al. [3] extended Ng's potential-based reward shaping to dynamic potential-based shaping, allowing the shaping function to change over time:

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

Finally, Harutyunyan et al. [8] combine these two extensions into dynamic shaping over state-action pairs:

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

All of these extensions preserve the total order over policies and therefore do not change the task, given Ng's original assumptions.

Harutyunyan et al. developed this last extension to set the stage for their paper's main result: they 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 [15]:

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

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 $R^\dagger$, like the main Q-function will reflect useful information towards good policies before convergence to the optimal policy.

2.3 Transfer Learning

Transfer learning allows a learning agent to re-use knowledge obtained in a previous task, aiming to learn faster in the current task. Various approaches for transfer have been proposed, varying in the learning algorithms they work with, the type and representation of information that is transferred, allowed task differences, etc. [20].

Typically, transfer algorithms will be provided an inter-task mapping, to translate between the state and action spaces of the source and target task, in order to leverage the information transferred. Such 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:

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

and

$$\chi_A(a_{\text{target}}) = a_{\text{source}}$$

Conversely, ρ represents a mapping from source task to target task:

$$\rho_S(s_{\text{source}}) = s_{\text{target}}$$

and

$$\rho_A(a_{\text{source}}) = a_{\text{target}}$$

Note that $\chi_S(\rho_S(s_{\text{source}})) = s_{\text{source}}$ need not always be true (and similarly for χ_A and ρ_A), as both χ and ρ need not be injective.

In this paper, we focus on policy transfer, the most general case with respect to the knowledge transferred, assuming the only available knowledge from the source task is the output of the learning algorithm, i.e. the policy. The target algorithms we consider are any value-based algorithms that can benefit from reward shaping.

Fernandez and Veloso [6] propose Probabilistic Policy Reuse (PPR) to transfer a policy to a TD learner. 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 a previous policy. With probability ψ, an action is selected according to the old 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.[1] This adds a bias to the exploration of the agent, intended to guide it towards good policies in the new task. To use the old policy in the new task, the target task's state needs to be mapped to the source task's state, an action needs to be selected according to the source task's policy, and that action needs to be mapped to the target task:

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

There are other policy transfer algorithms besides PPR, but none, to the best of our knowledge, that transfer a full policy to TD-learners. Most such techniques will consider learning options in the source task and transfer those to the new task [10, 14]. Other popular transfer learning techniques transfer low-level information, such as Q or V values [21].

[1]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 [23].

3. POLICY TRANSFER USING REWARD SHAPING

Given these preliminaries, we go on to describe our approach to policy transfer. To achieve policy transfer using reward shaping (PTS), we use the reward shaping technique developed by Harutyunyan et al. [8], which turns an arbitrary reward function into a potential-based shaping function, see Preliminaries. As the authors note, the approach is especially well-suited for the incorporation of behaviour-based knowledge, which is much harder to directly describe as a potential function without losing some information. Since a policy *is* behaviour-based knowledge, we build on this technique to realize policy transfer.

In order to use the technique, we need to define a reward function R^π in the new task that captures the policy π transferred from the source task. The idea is to reward the learning agent for taking action a in state s, proportionally to the probability of the mapped state-action pair $(\chi_S(s), \chi_A(a))$ in the transferred policy:

$$R^\pi(s, a, s') = \pi(\chi_S(s), \chi_A(a)) \tag{1}$$

Even though the formulation works for stochastic as well as deterministic policies, in this paper, we only focus on the latter. Therefore, R^π will always be either 0 or 1.

The negation of this reward function is then learned in a secondary value function Φ^π, whose values are used to shape the main reward R:

$$\begin{aligned} R_F(s, a, s', a') &= R(s, a, s') + F^\pi(s, a, t, s', a', t') \\ F^\pi(s, a, t, s', a', t') &= \gamma\Phi^\pi(s', a', t') - \Phi^\pi(s, a, t) \end{aligned}$$

Since this shaping is learned in parallel to the main learning, it is advisable to use a higher learning rate for the secondary value function, so that this information becomes available faster [8].

Note that a simpler approach to policy transfer using shaping could be taken, using a static potential function:

$$\Phi(s, a) = \pi(\chi_S(s), \chi_A(, a)) \tag{2}$$

The problem here, as argued in [8], is that when using this approach, a transition between two state-action pairs (s, a) with both high potential yields a very small shaping reward (as $F(s, a, s', a') = \gamma\Phi(s', a') - \Phi(s, a)$), while with the more elaborate technique, the actual shaping reward F would be high, which is desirable. We have evaluated the simpler approach empirically, and it proved to be inferior to the technique explained in this section. We omit these results from the experimental section for clarity of exposition.

In approaching policy transfer from a reward shaping perspective, we are able to ground this approach in the theory that has developed around potential-based reward shaping, leveraging the associated convergence guarantees for sound policy transfer.

4. EXPERIMENTS

To demonstrate the practical use of policy transfer using reward shaping as outlined in this paper, we evaluate the technique on two common reinforcement learning benchmarks, and a more complex domain: Mountain Car, Cart Pole and Mario. We compare with PPR, and evaluate the strengths and weaknesses of both techniques. Experiments in every domain are averaged over 100 trials, and the source task learning is rerun for every transfer trial. Whenever

we say methods perform differently or similarly, this is supported by the Student's t-test ($p = 0.05$).

4.1 Mountain Car 3D

In the standard Mountain Car task [16], the agent is in control of a car and needs to drive this car up a hill. Yet, the car is underpowered, and therefore cannot drive up the hill in one go. The agent needs to learn to build up momentum driving the car up and down two opposite hills, until enough potential energy has built up to drive to the top of the hill where the goal is located. The state space is described by the position of the car and its velocity $(x, \dot{x})$, while the actions available to the agent involve applying either negative, positive, or no force to the car ($A = \{Left, Right, Neutral\}$). The agent receives a per-step reward of -1, encouraging it to find the goal as quickly as possible. The 3D version of the task [18] (or 4D as some would have it) is very similar, except that the terrain is three dimensional, and the car can apply force along two axes, although only one at a time ($A = \{East, West, North, South, Neutral\}$). The state space consists of the car's position and velocity in two dimensions $(x, \dot{x}, y, \dot{y})$.

To learn this task, we use Q(λ)-learning and SARSA(λ) with tile-coding in the 2D and 3D tasks respectively (each chosen because it yields the best performing base-line agent in its respective task). In the 2D task, parameters are $\alpha = \frac{0.1}{14}$, $\gamma = 1$, $\epsilon = 0.1$, $\lambda = 0.95$, with 14 10×10 tilings. In the 3D task, parameters are $\alpha = \frac{0.2}{14}$, $\gamma = 1$, $\epsilon = 0.05$, $\lambda = 0.95$, with 14 $10 \times 10 \times 10 \times 10$ tilings. Weights are initialized pessimistically to $-\frac{250}{14}$, which yields better performance in both base learner and agents using transfer.

In PPR, to take an action from the 2D Mountain Car policy, first the 3D state must be mapped to the 2D state, an action must be selected according to the transferred policy, and then that 2D action must be mapped onto a 3D action. Both of these mappings are ambiguous:

$$\chi_S((x, \dot{x}, y, \dot{y})) = \begin{cases} (x, \dot{x}) & \text{probability } 0.5 \\ (y, \dot{y}) & \text{probability } 0.5 \end{cases}$$

$$\rho_A(\text{Left}) = \begin{cases} \text{West} & \text{probability } 0.5 \\ \text{South} & \text{probability } 0.5 \end{cases}$$

$$\rho_A(\text{Right}) = \begin{cases} \text{East} & \text{probability } 0.5 \\ \text{North} & \text{probability } 0.5 \end{cases}$$

$$\rho_A(\text{Neutral}) = \text{Neutral}$$

This ambiguity prevents one to use the old policy to its full potential with PPR. ψ, the parameter controlling how frequently the old policy is used, is initially set to 1, and we tuned ψ to decay by 0.99 after every episode.

Unlike PPR, PTS can use the old policy completely, as the values obtained from both possible state mappings can be combined, and actions are mapped from 3D to 2D, which is an unambiguous mapping, as opposed to the mapping from 2D to 3D:

$$\chi_A(\text{West}) = \text{Left}$$

$$\chi_A(\text{South}) = \text{Left}$$

$$\chi_A(\text{East}) = \text{Right}$$

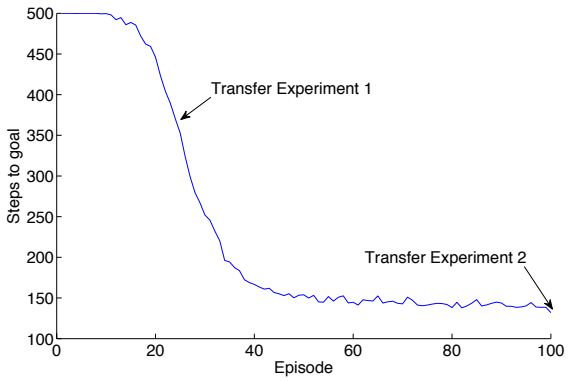

Figure 1: Basic Q(λ)-learner learning Mountain Car 2D. Convergence happens after about 50 episodes.

$$\chi_A(\text{North}) = \text{Right}$$

$$\chi_A(\text{Neutral}) = \text{Neutral}$$

The secondary reward function used for shaping is then:

$$R^\pi(s, a, s') = \frac{\pi(\chi_{S,1}(s), \chi_A(a)) + \pi(\chi_{S,2}(s), \chi_A(a))}{2}$$

with $\chi_{S,1}$ and $\chi_{S,2}$ indicating the two possible state mappings. The learning rate for the secondary value function, used for shaping, is $\beta = \frac{0.5}{14}$.

Figure 1 shows the basic Q-learner learning the 2D Mountain Car task. We perform two transfer experiments, either transferring after 25 or 100 episodes of learning in the 2D Mountain Car task. This allows us to investigate how these techniques perform when transferring either a suboptimal or near-optimal policy.

Figure 2 shows how both techniques perform when using a transferred policy that is suboptimal. Whereas PPR performs worse compared to the base learner, PTS results in faster learning, despite the transferred policy being far from optimal. When the transferred policy is near-optimal in the source task (Figure 3), PPR manages a big jumpstart in performance, as it immediately starts using the transferred policy. With the shaping approach on the other hand, a number of experiences are required before the transferred knowledge becomes apparent in the learning process. Despite the jumpstart, PPR cannot maintain this level of performance and quickly degrades before improving again (we discuss this phenomenon at the end of the paper), while PTS again achieves a statistically significant improvement in learning.

Since PTS and PPR are complementary on a technical level (the former modifies the reward signal, the latter affects action selection), we can investigate how they perform combined. During initial episodes, PTS+PPR performs the same as PPR, as with high initial ψ, action selection is fully controlled by PPR. As ψ decays, PTS+PPR's performance does not drop as much as with PPR alone.

4.2 Cart Pole

Cart Pole [12] is a task in which the agent controls a cart with a pole on top. The goal is to keep the pole balanced for as long as possible. The agent can move the cart either left or right within a given interval in a single dimension. The

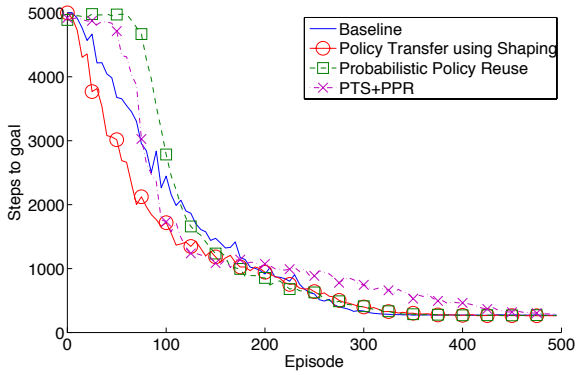

Figure 2: Policy transfer to Mountain Car 3D after having learned for 25 episodes in Mountain Car 2D, i.e. before convergence to the optimal policy. Nonetheless, PTS manages to improve performance over the base learner, while PPR cannot.

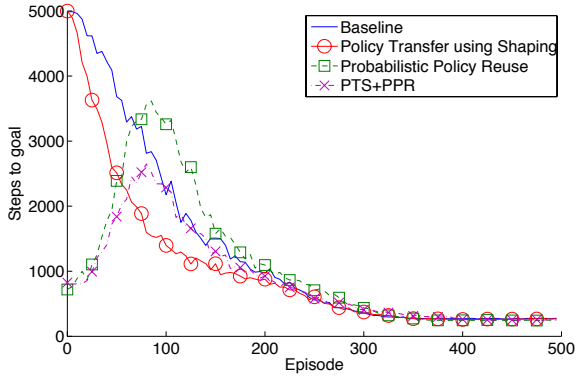

Figure 3: Policy transfer to Mountain Car 3D after having learned for 100 episodes in Mountain Car 2D, i.e. after convergence to a (near-) optimal policy. While PPR benefits from a jumpstart in performance, its performance deteriorates before improving again. PTS on the other hand exhibits more consistent learning, although it does not benefit from a jumpstart in performance.

state space consists of the position of the cart, its velocity, the angle of the pole and its angular velocity $(x, \dot{x}, \theta, \dot{\theta})$. We consider two versions of the task. The source task is the standard Cart Pole task, while we coin the target task the Heavy Cart Pole task, as we increase the weight of the pole from 0.1 to 1.0, making the task much harder. Since these two tasks only differ in transition function, no state or action mappings are required for transfer.

To learn the task, we use $Q(\lambda)$-learning with tile-coding. In the source task, we shape the learning with a static reward shaping function encouraging the angle of the pole to be 0, i.e. up $(\Phi(s) = 100(\pi - \bar{\theta}))$, for faster learning; we add π to make the potential function positive, i.e. optimistic. Parameters are $\alpha = \frac{0.05}{32}$, $\gamma = 1$, $\epsilon = 0.05$, $\lambda = 0.95$, with 32 $10\times10\times10\times10$ tilings for both tasks. Weights are initialized optimistically to 0, except with PPR. For PPR, we need to initialize the weights pessimistically to $-\frac{1}{32}$ to avoid a

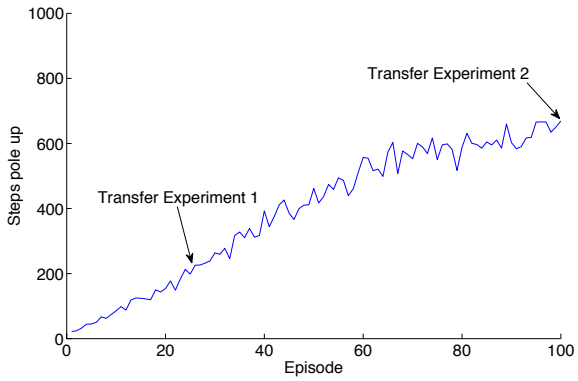

Figure 4: Basic $Q(\lambda)$-learner learning standard Cart Pole. It does not converge to the optimal policy within 100 episodes.

performance drop similar to the one seen in Mountain Car (we address this issue at the end of the paper). In PPR, ψ is initially set to 1, and decayed by 0.99 after every episode. In PTS, the secondary reward function is as described in Equation 1, and its value function is learned using the same parameters as the base learner, except with learning rate $\beta = \frac{0.5}{32}$.

We again perform two transfer experiments, one after learning for 25 episodes in the source task, and one after learning 100 episodes in the source task. In neither case will the agent have converged on the optimal policy, but the policy is much better after 100 episodes than after 25 episodes, as shown by the base-learner performance in the standard Cart Pole task, plotted in Figure 4.

Figure 5 shows the results for transfer after 25 learning episodes in the standard Cart Pole task. PPR again manages a good jumpstart in learning because it immediately starts using the transferred policy directly, while PTS needs learning experiences before the transferred knowledge affects the performance. Yet, PTS benefits much more from the transfer, despite the transferred policy's lack of quality, resulting in much better learning than PPR. The results for PPR are much better after 100 learning episodes in the source task (Figure 6), resulting in a jumpstart, and good learning from there on. PTS matches the performance of PPR and their combination only after some 600 episodes. Across the two experiments, PTS+PPR offers the best alternative.

4.3 Mario

The Mario benchmark problem [9] is a public reimplementation of the original Super Mario Bros® game. It involves an agent (Mario) that navigates a 2D level, collecting points for finishing the level, finding coins, getting hurt (negative points), etc. The goal is to collect as many points as possible. An episode is ended when time runs out, Mario dies, or when he finishes the level. The state space in Mario is fairly complex, as Mario observes the locations and types of enemies on the screen, he observes all information pertaining to himself, e.g. 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 (a brick, a coin, a mushroom, an enemy, etc.). Mario can take 12 distinct 'super' actions, each being a combination of one action

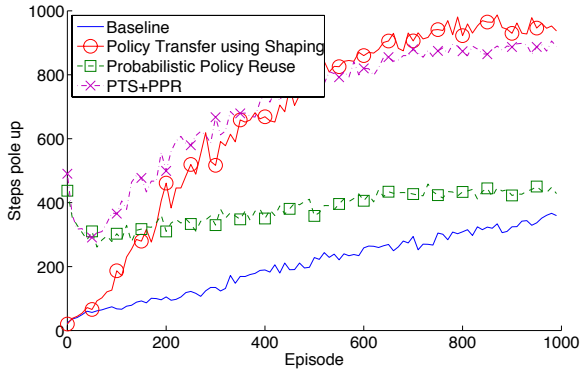

Figure 5: Policy transfer to Heavy Cart Pole, after having learned for 25 episodes in standard Cart Pole. Despite the low quality of the transferred policy, both PPR and PTS manage to leverage the transferred knowledge, although PTS is much more robust to the (lack of) quality of the transferred knowledge.

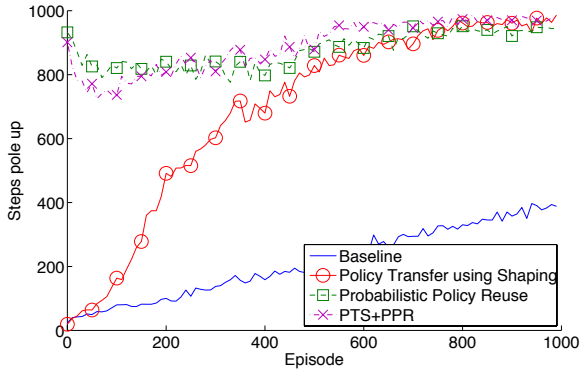

Figure 6: Policy transfer to Heavy Cart Pole, after having learned for 100 episodes in standard Cart Pole. PPR outperforms PTS in this case, as it manages to maintain the performance achieved after its jumpstart, while it takes PTS much longer to get to that level of performance.

from these three sets: {left, right, no direction}, {jump, do not jump} and {run, do not run}.

One part of the Mario game that can have a significant impact on performance is the presence of enemies. They are the source of most of the negative points Mario collects, by hurting, or worse, killing him. The latter ends the episode, preventing Mario from collecting more points. Therefore, a simple approach to transfer is to let Mario first figure out how to navigate and collect points in a level without enemies, and then transferring this information to a learning process in a level with enemies.

In these experiments, we use tabular $Q(\lambda)$-learning. The state-space in both source and target task consists of four boolean variables (indicating whether Mario is able to jump, on the ground, which direction he is facing, and whether he is able to shoot fireballs). In the target task, there are two additional variables indicating the relative position (x and

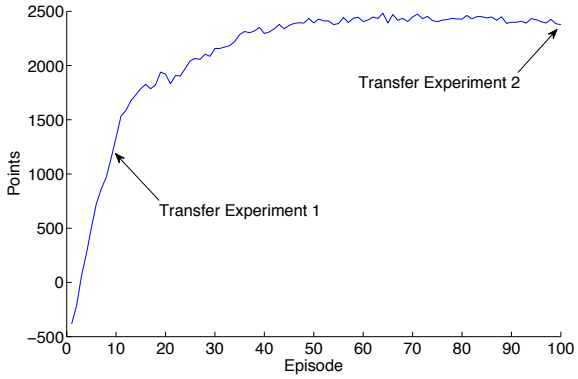

Figure 7: Basic $Q(\lambda)$-learner learning standard Mario without enemies. It converges after about 50 episodes.

y) of the closest enemy within a 21×21 grid surrounding Mario, making for a total of $2^4 \times (21^2 + 1) = 7072$ states (one extra for when there are no enemies in sight). Since we omit a lot of relevant state information, the agent operates in a non-Markovian state-space. Learning parameters are $\alpha = 0.01$, $\gamma = 0.9$, $\epsilon = 0.05$, and $\lambda = 0.5$. In PPR, ψ is initially set to 1, and decayed by 0.95 after every episode. In PTS, the secondary reward function is as described in Equation 1, and is learned using the same parameters as the base learner, except for its learning rate being $\beta = 0.05$.

The state-mapping from target to source task involves selecting only the four boolean variables, discarding the two state variables pertaining to enemies.

$$\chi_S(s_{target}) = s_{target,\{1..4\}}$$

The action spaces in source and target task are the same. Therefore no mapping is required.

Figure 7 shows the performance of the base learner in the Mario task without enemies (level generated with seed 0 and difficulty 0). We transfer to the same level, but with enemies, either after 10 or 100 learning episodes in the source task.

Results for transfer after learning only 10 episodes in the enemy-free level are depicted in Figure 8. PPR, PTS, and PTS+PPR all speed up learning, performing similarly, i.e. without statistically significant differences. Figure 9 shows the results for transferring the policy learned after 100 episodes. Transferring the better policy, PPR achieves a much better jumpstart, but PTS actually manages to converge to significantly better policies, which is possible due to the non-Markovian nature of the state-space.

5. DISCUSSION

Both Policy Transfer using Reward Shaping and Probabilistic Policy Reuse try to bias the agent's exploration in the target task using the transferred policy to achieve faster learning, but they do it in very dissimilar ways. PPR uses the transferred policy directly in the target task, while PTS rather uses the transferred policy to shape the state-action values being learned, which in turn biases exploration. The main advantage of PPR, is that using the transferred policy directly from the start can result in big jumpstarts in initial

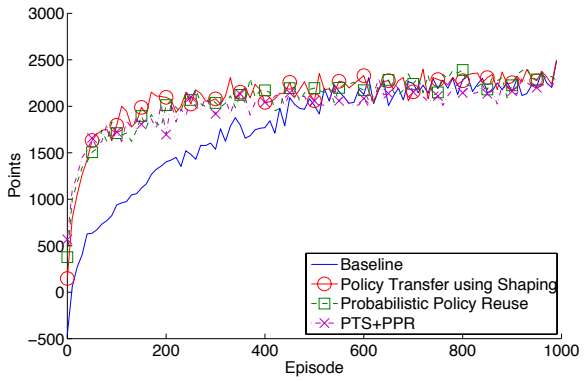

Figure 8: Policy transfer in Mario, after having learned for 10 episodes in the same level without enemies. PPR, PTS and their combination all improve performance, showing no statistically significant differences.

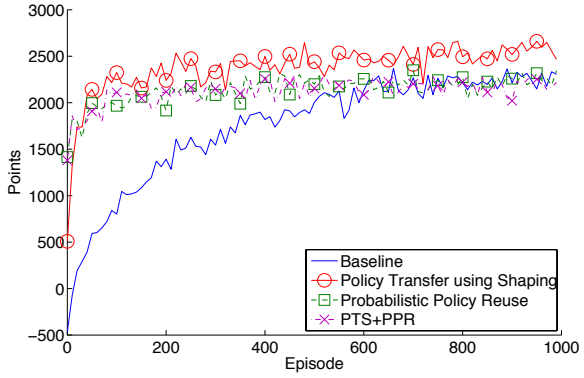

Figure 9: Policy transfer in Mario, after having learned for 100 episodes in the same level without enemies. PPR achieves a much greater jumpstart than PTS, but PTS quickly achieves the same level of performance, and actually converges to better policies than the base learner, PPR, or their combination.

performance. PTS on the other hand can only see the effect of the old policy after a state transition has been observed, and not before as PPR can.[2]

On the other hand, the shaping approach can easily exploit ambiguous state and action mappings, by combining the values obtained for each, while PPR is limited to taking a single action.[3] With respect to the quality of the transferred policy, PTS appeared to be more robust in our experiments, outperforming the base learner in every case, despite the suboptimality of transferred policies. PPR suffered more from the lack of quality in the transferred policy. A likely reason for this is that PPR always uses either only information from the old or the new task, while PTS always uses a combination of both, as its decisions will always be based on both the new environment's reward and the transferred policy's shaping.

Furthermore, PTS is firmly grounded in potential-based reward shaping theory, providing convergence guarantees for PTS, previously proven in that literature, without requiring special measures. PPR on the other hand needs to decrease the use of the transferred policy to preserve convergence guarantees.

Lastly, we need to address the drop in performance observed in Mountain Car for PPR (and in Cart Pole when not initializing pessimistically, results not included). For PPR to be effective, it needs to be able to affect the Q-values in such a way that its trajectories become positively reinforced. With Mountain Car, there is a step-reward of -1, which actually results in PPR's trajectories being negatively reinforced, except close to the goal, at least initially. In Cart Pole, initializing optimistically with 0, and receiving step rewards of 0, results in the old policy being unable to affect the Q-values in any way (positive or negative), and thus when the learner progressively relies less on the transferred policy, it will have learned little from those experiences. On the other hand, initializing the Q-values in Cart Pole to -1, will lead to every visited state-action pair's Q-value being positively reinforced due to the step-reward of 0, allowing PPR to carve a path in the Q-values.

Since PTS and PPR are compatible on a technical level, with PTS operating on the reward signal and PPR on the action selection mechanism, we have evaluated their combination too. PTS appears to be useful to improve the performance of PPR, as in our experiments PTS+PPR always outperformed or matched PPR performance. On the other hand, it is not true that PTS+PPR always outperforms PTS alone. As ψ is typically initialized to 1 (both in this paper and other works [6, 7, 23]), initial performance is completely determined by PPR. As ψ decays, PTS manages to affect learning more, but can not always overcome the initial bias set by PPR (as exemplified by the results in Mario, a non-Markovian environment), as opposed to when using PTS alone. Therefore, PTS appears to be a way to improve PPR, but not the reverse.

6. CONCLUSIONS AND FUTURE WORK

We presented a novel approach to policy transfer, encoding the transferred policy as a dynamic potential-based reward shaping function, benefiting from all the theory behind reward shaping. An experimental comparison between Policy Transfer using Reward Shaping (PTS) and Probabilistic Policy Reuse (PPR) has shown the weaknesses and strengths of both in several domains. Because PPR uses the transferred policy directly during action selection, it can achieve a big jumpstart in performance, but can not always maintain this level of performance as its effect on the value function being learned depends on factors such as the environment's step-reward and the value function's initialization. PTS needs more learning experiences before the effect of the transferred policy becomes apparent, but the technique appears much more robust against lack of quality of the transferred policy, and does not suffer from issues similar to PPR's. The combination of the two techniques is a way to improve PPR performance, while PTS alone may be a better option in some cases.

[2]We have experimented with initializing the Q-function with the static potential function (Eq. 2), aiming to achieve a similar jumpstart in PTS. This improved initial performance, but not dramatically so.

[3]One could approach this problem in an ensemble fashion, combining the different mappings through some voting mechanism, but that is beyond the scope of this article.

In a follow-up study, we are interested in looking at multi-task transfer. Recent developments in reward shaping have shown how learning with different shapings in parallel, and combining their estimates using ensemble techniques can speed up learning a lot [2]. In a transfer context, we can transfer multiple policies using PTS, learn estimates in parallel, and achieve multi-task transfer using an ensemble technique.

Furthermore, we want to look at transfer between agents using different learning algorithms (including humans), where policy transfer makes the most sense.

Acknowledgement

Tim Brys is funded by a Ph.D grant of the Research Foundation Flanders (FWO). Anna Harutyunyan is supported by the IWT-SBO project MIRAD (grant nr. 120057). This work was supported in part by NSF IIS-1149917.

7. REFERENCES

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

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

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

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

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

[6] F. Fernández, J. García, and M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, 2010.

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

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

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

[10] G. Konidaris and A. G. Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pages 895–900, 2007.

[11] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 544–551. ACM, 2008.

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

[13] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, volume 99, pages 278–287, 1999.

[14] B. Ravindran and A. G. Barto. Relativized options: Choosing the right transformation. In *ICML*, pages 608–615, 2003.

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

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

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

[18] M. E. Taylor, G. Kuhlmann, and P. Stone. Autonomous transfer for reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 283–290. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[19] M. E. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 53–59. ACM, 2005.

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

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

[22] M. E. Taylor, S. Whiteson, and P. Stone. Transfer via inter-task mappings in policy search reinforcement learning. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 156–163. ACM, 2007.

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

[24] E. Wiewiora. Potential-based shaping and Q-value initialization are equivalent. *J. Artif. Intell. Res. (JAIR)*, 19:205–208, 2003.

[25] E. Wiewiora, G. Cottrell, and C. Elkan. Principled methods for advising reinforcement learning agents. In *ICML*, pages 792–799, 2003.