Multi-Task Reinforcement Learning: Shaping and Feature Selection

Matthijs Snel and Shimon Whiteson

Intelligent Systems Lab Amsterdam (ISLA),
University of Amsterdam, 1090 GE Amsterdam, Netherlands
m.snel, s.a.whiteson@uva.nl

Abstract. Shaping functions can be used in multi-task reinforcement learning (RL) to incorporate knowledge from previously experienced source tasks to speed up learning on a new target task. Earlier work has not clearly motivated choices for the shaping function. This paper discusses and empirically compares several alternatives, and demonstrates that the most intuive one may not always be the best option. In addition, we extend previous work on identifying good representations for the value and shaping functions, and show that selecting the right representation results in improved generalization over tasks.

1 Introduction

Transfer learning approaches to reinforcement learning (RL) aim to improve performance on some *target tasks* by leveraging experience from some *source tasks*. Clearly, the target tasks must be related to the source tasks in order for transfer to be beneficial. In *multi-task reinforcement learning* (MTRL), this relationship is formalized with a *domain*, a distribution over tasks from which the source and target tasks are independently drawn [18]. For example, consider a domain consisting of a fixed goal location in a room with obstacles, where each obstacle configuration constitutes a different task. In this case, the agent could exploit the fact that the goal is always in the same place to learn new tasks more quickly.

Various approaches to MTRL exist, such as using the source tasks to form a prior for Bayesian RL [7, 21] or transfering state abstractions [19] (see [18] for an overview of transfer learning approaches for RL). In this paper, we consider a different approach in which the source tasks are used to learn a *shaping function* that guides learning in the target tasks. Shaping functions [10] augment a task's base reward function, which is often sparse, with additional artificial rewards. By capturing prior knowledge about the task, manually designed shaping functions have proven successful at speeding learning in single-task RL (e.g. [1, 3, 20]). In MTRL, a shaping function can be learned automatically from the source tasks. In the above example, an agent that observes that the goal location is fixed in the source tasks could learn a shaping function that rewards approaching this location.

This paper makes two contributions. First, it investigates several strategies for learning shaping functions. Earlier work learned a shaping function based on an approximation of the optimal value function V^* or Q^* for each source task [14, 6, 17]. While these approaches are intuitive, they have never been explicitly motivated. We discuss and compare several alternatives, and demonstrate that approximating the optimal value function across tasks may not always be the best option.

Second, this paper also considers what representation the shaping function should employ. Konidaris and Barto observed that the best state features to use for shaping can be different from those used to represent the value function for each task. However, their experiments relied on manually selecting these features. Snel and Whiteson proposed a definition of the relevance of a feature for shaping and provided a method that uses this definition to automatically construct a shaping function representation. However, while their experiments validate their definition, they do not demonstrate that automatically finding the right shaping features can improve performance. We extend this work by casting the problem of learning a shaping function as a regression problem to which feature selection methods from supervised learning can be applied. Using this approach, we demonstrate that effective feature selection improves generalization from the source tasks, which in turn improves the shaping function's ability to speed target task learning.

2 Background and Notation

An MDP is a tuple $m = \langle \mathcal{S}_m, \mathcal{A}_m, \mathcal{B}_m, P_m, R_m, \gamma \rangle$ with set of states $\mathcal{S}_m$, set of actions $\mathcal{A}_m$, and set of admissable action pairs $\mathcal{B}_m \subseteq \mathcal{S}_m \times \mathcal{A}_m$. A transition from s to s' given a has probability $P_m^{sas'}$ and results in expected reward $R_m^{sas'}$. The expected γ-discounted return in m when taking a in s under policy π_m is $Q_m^{\pi}(s,a)$, and under the optimal policy π_m^* it is $Q_m^*(s,a)$.

A *domain* d is a distribution $\Pr(m)$, where $m \in \mathcal{M}$, a set of MDPs. The domain has state set $\mathcal{S}_d = \bigcup_m \mathcal{S}_m$, action set $\mathcal{A}_d = \bigcup_m \mathcal{A}_m$ and admissable action pairs $\mathcal{B}_d = \bigcup_m \mathcal{B}_m$, allowing for MDPs with different state and action spaces.

This paper considers *potential-based* shaping functions of the form $F_{s'a'}^{sa} = \gamma\Phi(s',a') - \Phi(s,a)$, where $\Phi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is a *potential function* that assigns a measure of "desirability" of a state-action pair and s' and a' are the next state and action. Ng et al. [10] showed that such shaping functions preserve the optimal policy of the MDP and Wiewiora et al. [20] showed that using them is equivalent to initializing the agent's Q-table to Φ.

3 Approximating the Optimal Shaping Function

In this section, we formulate a definition of an optimal potential-based shaping function and propose four different ways of approximating it. We focus on a transfer learning scenario in which the agent aims to maximize the total reward accrued while learning on a set of target tasks; this measure also implicitly

captures the time to reach a good policy. Furthermore, for simplicity we assume a tabular learning algorithm L is employed on each target task. In this setting, an optimal shaping function is one based on a potential function that maximizes expected return across target tasks:

$$\Phi_L^* = \operatorname*{argmax}_{\Phi} \mathrm{E}\left[R_m|\Phi, L\right] = \operatorname*{argmax}_{\Phi} \sum_{m\in\mathcal{M}} \Pr(m)\mathrm{E}\left[\sum_{t=0}^{\infty}\gamma^t r_{t,m}|\Phi, L\right], \qquad (1)$$

where R_m is the return accrued in task m and $r_{t,m}$ is the immediate reward obtained on timestep t in task m. Note that the task should be seen as a hidden variable since the potential function does not take it as input. This means that the best potential function may perform poorly in some tasks; however, on average across tasks, it will do better than any other function.

Since shaping with Φ is equivalent to initializing the Q-table with it, solving (1) is equivalent to finding the best cross-task initialization of the Q-table. Unfortunately, there is no obvious way to compute such a solution efficiently, and search approaches quickly become impractical. Therefore, in the following sections we discuss four strategies for efficiently approximating Φ_L^*.

3.1 Initialization Closest to Q_m^*

Intuitively, a good initialization of the Q-function is the one that is closest in expectation to the optimal value function Q_m^* of the target task m the agent will face. Since choosing Φ is equivalent to choosing such an initialization, one way to approximate Φ_L^* is by minimizing the *expected mean squared error* (EMSE) across tasks:

$$EMSE(\Phi) = \sum_{m\in\mathcal{M}} \Pr(m) \sum_{s,a\in\mathcal{B}_m} \Pr(s,a|m)\Big[Q_m^*(s,a) - \Phi(s,a)\Big]^2, \qquad (2)$$

where $\Pr(s,a|m)$ is a task-dependent weighting over state-action pairs that determines how much each pair contributes to the error.

It is not immediately clear how to select $\Pr(s,a|m)$, though a minimal requirement is that $\Pr(s,a|m) = 0$ for all $(s,a) \notin \mathcal{B}_m$. The simplest option is to set $\Pr(s,a|m) = 1/|\mathcal{B}_m|$ for all $(s,a) \in \mathcal{B}_m$. Since the EMSE(Φ) averages over Q_m^*, another option is to use the distribution induced by π_m^*. However, this may be problematic as it represents only the distribution over (s,a) *after* learning. There may be state-action pairs that are never visited under π_m^* but that are visited during learning and for which $EMSE(\Phi)$ is thus important. The best choice of $\Pr(s,a|m)$ may thus be the distribution over (s,a) *during* learning. A disadvantage is that this requires applying L to all $m \in \mathcal{M}$.

Given a choice of $\Pr(s,a|m)$, we can find $\hat{\Phi}_m = \operatorname{argmin}_{\Phi} EMSE(\Phi)$ by setting the gradient of (2) to zero and solving, yielding:

$$\hat{\Phi}_*(s,a) = \sum_{m\in\mathcal{M}} \Pr(m|s,a)Q_m^*(s,a) \qquad \forall s,a \in \mathcal{B}_d. \qquad (3)$$

The notation $\hat{\Phi}$ indicates an approximation to Φ^*_L. The subscript indicates that this approximation averages over the optimal solution for each m. For (s, a) in $\mathcal{B}_d$ but not in $\mathcal{B}_m$ for a given m, we define $Q^*_m(s, a) \equiv 0$. In any case, these values are already excluded from the average by the weighting $\Pr(m|s, a)$.

While approaches similar to (3) have been used successfully in previous work [6, 14, 17], they are not guaranteed to be optimal. Since they minimize error with respect to the optimal value function that *results* from learning, they may be too optimistic *during* learning, which the shaping function is meant to guide. Consequently, they may cause the agent to over-explore the target task, to the detriment of total reward. Section 4 will provide experimental support for this claim.

3.2 Initialization Closest to $\tilde{Q}_m$

In some cases (e.g., when using function approximation or an on-policy algorithm with a soft policy), the agent's Q-function on a sournce or target task m will never reach Q^*_m, even after learning. When this occurs, it may be better to base the potential function on an average over $\tilde{Q}_m$, the value function to which the learning algorithm converges. The derivation is the same as for the previous section, yielding:

$$\hat{\Phi}_{\tilde{Q}}(s) = \sum_{m \in \mathcal{M}} \Pr(m|s, a)\tilde{Q}_m(s, a) \qquad \forall s, a \in \mathcal{B}_d. \tag{4}$$

3.3 Best Fixed Cross-Task Policy

While $\hat{\Phi}_{\tilde{Q}}$ is less optimistic than $\hat{\Phi}_*$, it may still be too optimistic since it is also based on minimizing error with respect to a value function *after* learning. To address this issue, we can instead base the potential function on the optimal *cross-task value function*. A cross-task value function describes the expected return for a fixed *cross-task policy*, i.e., one that assigns the same probability to a given state-action pair regardless of what task it is used in.

We define μ^* to be the policy that maximizes expected total reward given the initial state distribution $\Pr(S_0 = s)$:

$$V^\mu_d = \sum_s \Pr(S_0 = s) \sum_a \mu(s, a) Q^\mu_d(s, a) \tag{5}$$

$$Q^\mu_d(s, a) = \sum_{m \in \mathcal{M}} \Pr(m|s, a) Q^\mu_m(s, a), \tag{6}$$

where V^μ_d is the domain-wide value of μ, $\Pr(m|s, a)$ is a weighting term like in section 3.1, and $Q^\mu_m(s, a)$ is the value of pair (s, a) in m under μ. Unfortunately, it seems that no Bellman-like equation can be derived from (6), so we are left with a dependency on $Q^\mu_m(s, a)$. One consequence of this is that we are restricted to policy search methods for trying to find μ^*.

Using these definitions, we can define a potential function based on the optimal cross-task value function:

$$\hat{\Phi}_{\mu^*}(s,a) = Q_d^{\mu^*}(s,a) \qquad \forall s,a \in \mathcal{B}_d. \tag{7}$$

The key distinction between this potential function and both $\hat{\Phi}_{\bar{Q}}$ and $\hat{\Phi}_*(s,a)$ is that it averages expected return of a single fixed policy. In contrast, in $\hat{\Phi}_{\bar{Q}}(s)$ and $\hat{\Phi}_*(s,a)$ averages are computed over different policies for each task.

3.4 Averaging MDP

A potential drawback of the previous approaches is that solutions to all tasks must be computed, or a policy search must be done to find μ^*. If the task models are available, then we could take the average of these models according to $\Pr(m)$, and compute the solution to this *averaging MDP*, defined as

$$\bar{R}^{sas'} = \sum_{m \in \mathcal{M}} \Pr(m|s,a) R_m^{sas'} \tag{8}$$

$$\bar{P}^{sas'} = \sum_{m \in \mathcal{M}} \Pr(m|s,a) P_m^{sas'}. \tag{9}$$

It might not be immediately clear what the difference, if any, between the solution to the averaging MDP and for example equation 3 is. As section 4 will show, it turns out that they are not necessarily the same, although there exist tasks for which they are. Furthermore, the solution to the averaging MDP does usually not correspond to μ^*, since the latter may not be deterministic (analogous to a reactive policy for a POMDP).

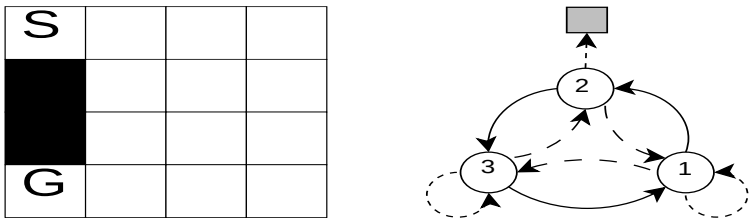
The potential function equals the solution to the averaging MDP:

$$\hat{\Phi}_{\text{avg}}(s,a) = \bar{Q}^*(s,a) \qquad \forall s,a \in \mathcal{B}_d. \tag{10}$$

Finally, it should be clear that in this case we should use $\Pr(s,a|m) = 1/|\mathcal{B}_m|$.

4 Shaping Function Evaluation

This section empirically compares the four different potential functions proposed in the previous sections to a baseline agent that does not use shaping. For comparison purposes, we assume we have perfect knowledge of the domain and compute each potential function using all tasks in the domain [1]. Evaluation occurs using a sample of tasks from the same domain. This enables us to compare the potential functions' maximum potential, untainted by sampling error. In the next sections, we discuss and evaluate cases in which only a sample of tasks from the domain is available.



(a) Example cliff task (b) Example task from sect. 6

Fig. 1: Example tasks from the domains used for evaluation in section 4 (a) and 6 (b). In (b), line types solid, dashed, and dotted represents actions 1, 2, and 3; grey square is terminal.

4.1 Domain

To illustrate a scenario in which $\hat{\Phi}_*$, the average over optimal value functions, is not the optimal potential function, we define a *cliff domain* based on the episodic cliff-walking grid world from Sutton and Barto [16]. One task from this domain is shown in Fig. 1a. The agent starts in **S** and needs to reach the goal location **G**, while avoiding stepping into the cliff represented by the black area.

The domain consists of all permutations with the goal and start state in opposite corners of the same row or column with a cliff between them (8 tasks in total). Each task is a 4x4 grid world with deterministic actions N, E, S, W, states (x, y), and a -1 step penalty. Falling down the cliff results in -1000 reward and teleportation to the start state. The distribution over tasks is uniform.

4.2 Method

We compute each potential function according to the definitions given in section 3. Since we cannot compute the cross-task policy μ^* exactly, we use an evolutionary algorithm (EA) to approximate it.

To illustrate how performance of a given Φ depends on the learning algorithm, we use two standard RL algorithms, Sarsa(0) and Q(0). Since for Q-Learning, $\hat{\Phi}_{\bar{Q}}(s) = \hat{\Phi}_*(s)$, we use Sarsa's solution for $\hat{\Phi}_{\bar{Q}}(s)$ for both algorithms. Both algorithms use an ϵ-greedy policy with $\epsilon = 0.1$, $\gamma = 1$, and the learning rate $\alpha = 1$ for Q-Learning and $\alpha = 0.4$ for Sarsa, maximizing the performance of both algorithms for the given ϵ.

We also run an additional set of experiments in which the agent is given a "cliff sensor" that indicates the direction of the cliff (N, E, S, W) if the agent is standing right next to it. Note that the addition of this sensor makes no difference for learning a single task: it does not change the optimal policy, nor does the number of states per task increase. However, the number of states in

[1] With perfect knowledge of the domain, a better approach would be to store the optimal policies for each task and select the appropriate one while interacting with the target task. However, our current approach serves to demonstrate the theoretical advantages of each potential function type.

the domain does increase: one result of adding the sensor is that tasks no longer have identical state spaces.

For each potential function, we report the mean total reward incurred by sampling a task from the domain, running the agent for 500 episodes, and repeating this 100 times.

4.3 Results

	Q-Learning	**Sarsa**
$\hat{\Phi}_{\mu^*}$	-19.77 ± 2.43	-512 ± 130
$\hat{\Phi}_{\text{avg}}$	-5.83 ± 0.13	-4.48 ± 0.11
No shaping	-5.86 ± 0.12	-3.86 ± 0.10
$\hat{\Phi}_*$	-5.13 ± 0.17	-3.96 ± 0.11
$\hat{\Phi}_{\tilde{Q}}$	-4.74 ± 0.19	-3.93 ± 0.11

(a) Without sensor

	Q-Learning	**Sarsa**
No shaping	-5.85 ± 0.13	-3.96 ± 0.11
$\hat{\Phi}_*$	-5.44 ± 0.12	-3.67 ± 0.12
$\hat{\Phi}_{\tilde{Q}}$	-4.75 ± 0.17	-3.37 ± 0.13

(b) With sensor

Table 1: Mean total reward for various shaping configurations and learning algorithms on the cliff domain. All numbers $\times 10^4$. Intervals represent the 95% confidence interval.

Table 1a shows the performance for the scenario without cliff sensor. On this domain, $\hat{\Phi}_d^{\mu}$ performs very poorly; one reason for this may be that the EA did not find μ^*, but a more likely one is that due to the structure of the domain, even μ^* would incur a very low return for each state and thus lead to a very pessimistic potential function.

As expected, Sarsa outperforms Q-Learning on this domain due to its on-policy nature: because Q-Learning learns the optimal policy directly, it tends to take the path right next to the cliff and is thus more likely to fall in. For both algorithms, $\hat{\Phi}_{\text{avg}}$ does not outperform the baseline agent without shaping. For Q-Learning, $\hat{\Phi}_*$ and $\hat{\Phi}_{\tilde{Q}}(s)$ do better than the baseline, with the latter doing significantly better. Inspection of the learning curves shows that for $\hat{\Phi}_{\tilde{Q}}(s)$, Q-Learning initially incurs high average reward because it is driven away from the cliff by the potential function (remember that this is the average *Sarsa* solution to the domain), but thereafter a regress occurs as it returns to its risky behavior. For Sarsa, there is no significant difference between the baseline and either potential function. This is slightly surprising, since an initial gain would be expected.

The situation changes when the cliff sensor is added (we did not retest the two potential functions that did worse than the baseline), see table 1b. Although the sensor does not help speed learning within a given task, across tasks it provides consistent information. Whenever the grid sensor provides a signal, the agent should not step in that direction. This information is reflected in the average of the value functions and thus in the potential function (technically, what happens is that the state-action space $\mathcal{B}_d$ is enlarged, and fewer state-action pairs are

shared between tasks). Under these circumstances, both $\hat{\Phi}_*$ and $\hat{\Phi}_{\tilde{Q}}$ significantly outperform baseline Sarsa, with the latter, again, doing best. The picture for Q-Learning remains largely the same.

5 Shaping Function Representations

We now turn to the case where only a sample of N tasks is available, and our goal is to maximize the expected total return while learning on a number of target tasks, sampled from the same domain. That is, the goal is to find the Φ_L^* that satisfies (1), where $\mathcal{M}$ is the set of possible target tasks.

One option is to compute a $\hat{\Phi}$ that exactly matches the sample data, by letting the set $\mathcal{M}$ of tasks that we average over (see the definitions in section 3) be the set of sample tasks. However, it is not desirable to do so since such an overfit $\hat{\Phi}$ will usually generalize poorly to unseen tasks and state-action pairs. Generalization over state-action pairs requires a function approximator (note that if the state-action space is the same for every task and sufficiently small, this is not necessary). Generalization over tasks requires the identification of state-action pair properties that result in a more consistent (low-variance) estimator $\hat{\Phi}$ across tasks. For factored state spaces, this amounts to doing feature selection; in the general case, it amounts to state abstraction (e.g., [8]).

While the connection with state abstraction methods is interesting, space constraints preclude an in-depth discussion. Therefore, we will focus on factored state spaces, since it is more intuitive and most problems fall in this category. For example, in the cliff domain, which has factored states, the addition of a cliff sensor increases the impact of the potential functions on performance. This is because, *whichever task the agent is in*, the correlation between the cliff sensor feature and expected return is consistent: whenever there is a cliff to the north, large negative return follows when stepping north. The expected return associated with position, on the other hand, varies greatly per task. Thus, the cliff sensor seems like a good feature to use for $\hat{\Phi}$, while position does not.

Both feature selection (FS) and state abstraction methods have been applied to single-task RL, with especially FS seeing a recent surge in interest [11, 12, 4, 9]. Although similar methods have also been applied to transfer learning, most of these seek to reduce the state space of the target task, or find good representations for value functions or policies, by identifying features or abstractions that are useful *within* tasks [5, 19, 15].

None of these approaches applies the insight that features that are not useful for transferring a value function or policy might be useful in some other way, for example for learning a shaping function. Although previous work on shaping in MTRL exists that does apply this insight [6, 13], it circumvents the feature selection problem by manually designing a representation for the value and shaping function. Thus, these papers do not make clear what makes a feature set useful for the shaping function. Snel and Whiteson [14] use an information-theoretic metric to define *task-relevant features* as features that provide information on value within tasks, and *domain-relevant* features to be those that provide information on value across tasks. The latter are the ones that should be used for

the shaping function. While experiments validate their definitions, the authors do not demonstrate that automatically finding the right features for shaping can improve generalization performance. Furthermore, their definition might not be the best for a regression setting, since their metric does not take error magnitude into account.

This paper casts the problem as a supervised regression problem to which supervised learning and feature selection algorithms can be applied to identify the relative importance of features. Various learning algorithms are suitable, but for this paper we use a random forest [2] of regression trees. This suits our current purposes nicely since it is a method with a proven track record that is easy to interpret in terms of the original features, and offers ways to estimate relative feature importance in terms of the impact on squared error of the estimate. In the next section, we show how this method can be applied to find task- and domain-relevant features and to find a potential function that generalizes well over tasks.

6 Evaluation of Representations

Consider the episodic domain of which an example task is shown in figure 1b, with a step penalty of -1. In addition to feature x_1, which corresponds to the state numbers shown, the agent perceives two additional features x_2 and x_3. Feature x_2 is the inverse of the square of the shortest distance to the goal, i.e. in the figure, the states would be $(1, 0.25), (2, 1), (3, 0.25)$. Feature x_3 is a binary feature that indicates whether the task is one in which it is undesirable to stay in one place (1) or not (0); if 1, the agent receives a -10 penalty for self-transitions. x_3 is constant within a task, but may change from one task to the next. The goal may be at either of the three states, and the effect of actions 1 and 2 (see figure caption) may be reversed. Action 3 always results in either a self-transition or a goal-transition (when the goal is next to the current state). This amounts to 12 possible tasks in total.

The domain is kept small in order to facilitate computing all potential function types exactly, and to illustrate the main points of the paper. Furthermore, the features are chosen such that they represent three categories: the x_1 feature is task relevant, only providing information within a task (because its relation to return changes in each task); x_3 is useless within a task, but useful to distinguish classes of tasks, and therefore domain relevant. The x_2 feature, finally, is both task and domain relevant. The next subsection will justify and illustrate these claims experimentally.

6.1 Feature Relevance

To compute feature relevance, we choose $\hat{\Phi}_*$, the average over optimal value functions, as the target potential function (experiments showed that for this domain, it does not matter which potential function we choose for this). Thus, we first solve N sample tasks, then use the $N|\mathcal{S}_m||\mathcal{A}_m| = 9N$ state-action-value examples to train a random forest as estimator of $\hat{\Phi}_*$.

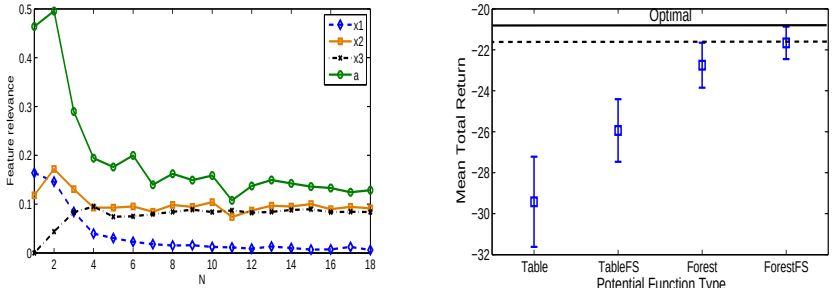


Fig. 2: Left: Feature relevance per sample size N, where a indicates action. Right: Comparison of shaping functions without and with feature selection. Error bars indicate 95% confidence interval. Horizontal solid (dashed) line is mean (95% confidence interval) of a potential function with complete domain knowledge.

Fig. 2a shows how the estimated importance of each feature changes with increasing sample size. For each tree in the forest, feature importance is the weighted (by tree node probability) sum in changes in error for each split on the feature. A change is computed as the difference between the error of the parent node and the total error of the two children. This roughly means that the more a feature contributes to reduction of the squared error in prediction of optimal value, the more important it is. Each feature score then is averaged over all trees in the forest. Final score is computed by repeating each measurement 10 times for each sample size and taking the mean.

The changes in feature relevance with increasing sample size clearly demonstrate the difference between task and domain relevance. For a single task ($N = 1$), x_3 is irrelevant since it is constant within a task. The action is most relevant, followed by x_1 and x_2. As sample size increases, x_2 and x_3 gain in importance, while x_1 loses importance. This pattern makes sense: while x_1 has strong correlation with return in a single task, this correlation decreases for increasing sample size (when all tasks in the domain are sampled perfectly uniformly, it is 0). Thus x_1 is task relevant, but not domain relevant. Feature x_2 is both task *and* domain relevant: it is relevant for $N = 1$, but remains so as sample size increases. This makes sense since decreasing distance to the goal leads to higher expected return both within and across tasks. x_3 is only domain relevant: it can be used to distinguish classes of tasks, but is not useful within a single task. Finally, the importance of the action suggests a shaping function of the form $\Phi(s, a)$; if actions did not provide any information, we might also use a potential over just states, $\Phi(s)$.

6.2 Generalization

To asses the impact on generalization, we next compare four different potential types: a table-based one without FS (*table*) and with FS (*tableFS*), and a random forest without (*forest*) and with (*forestFS*) FS. Before doing so, we computed all potential types of section 3, assuming full knowledge of the domain. This should

provide an upper bound on the performance of the potential functions computed based on samples. In this domain, it turns out there is no significant difference between the potential types, so we use $\hat{\Phi}_*$ in all experiments.

Performance of each potential function is assessed by computing the potential function on 4 samples (25% of the domain size), sampling a task from the domain and recording the total reward incurred by a Q-Learning agent run for 10 episodes. Final performance is the average over 100 such trials. Fig. 2b displays the results. The horizontal line indicates performance of the shaping function computed assuming complete domain knowledge, with the dashed line indicating the 95% confidence interval. The table-based function without FS does worst; performance is improved by deselecting the x_1 feature as per fig. 2a, indicating that leaving this task-relevant feature out improves generalization across tasks. The random forests significantly outperform the table-based functions, while the performance difference between them is not significant. The advantage of the forests over the tables is, as noted before, their capacity to also generalize to unseen state-action pairs. The fact that FS does not have a great impact on random forest performance stems from the fact that, by averaging over many independently grown trees, this method has some inherent protection against overfitting.

7 Conclusion

This paper makes two contributions to the multi-task RL (MTRL) literature. First, while shaping functions have been used in MTRL before, this paper is the first to provide an extensive discussion and empirical comparison of different types of shaping functions that could be useful to improve target task performance. While some previous work on shaping in MTRL has used the average over optimal value functions of the source tasks as potential function [14, 6, 17], our results demonstrate that this is not always the best option: the best potential function highly depends on the domain and learning algorithm under consideration.

Second, by casting the problem of finding a shaping function as a supervised regression problem, we can automatically detect which features are relevant for the shaping function, by measuring each feature's influence on the squared error in prediction of cross-task value. We further demonstrate that feature selection improves generalization from the source tasks, which in turn improves the shaping function's ability to speed target task learning. Future work should extend these results to domains with larger feature and state spaces, and assess the impact of shaping functions on more advanced learning algorithms.

Bibliography

[1] J. Asmuth, M. Littman, and R. Zinkov. Potential-based shaping in model-based reinforcement learning. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 604–609. The AAAI Press, 2008.

[2] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[3] S. Elfwing, E. Uchibe, K. Doya, and H. Christensen. Co-evolution of shaping: Rewards and meta-parameters in reinforcement learning. *Adaptive Behavior*, 16(6):400–412, 2008.

[4] H. Hachiya and M. Sugiyama. Feature selection for reinforcement learning: Evaluating implicit state-reward dependency via conditional mutual information. In *ECML/PKDD*, pages 474–489, 2010.

[5] N. K. Jong and P. Stone. State abstraction discovery from irrelevant state variables. In *IJCAI-05*, 2005.

[6] G. Konidaris and A. Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proc. 23rd International Conference on Machine Learning*, pages 489–496, 2006.

[7] A. Lazaric and M. Ghavamzadeh. Bayesian multi-task reinforcement learning. In *ICML*, pages 599–606, 2010.

[8] L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for mdps. In *Aritificial Intelligence and Mathematics*, 2006.

[9] S. Mahadevan. Representation discovery in sequential decision making. In *AAAI*, 2010.

[10] A. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proc. 16th International Conference on Machine Learning*, 1999.

[11] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *ICML*, pages 752–759, 2008.

[12] M. Petrik, G. Taylor, R. Parr, and S. Zilberstein. Feature selection using regularization in approximate linear programs for markov decision processes. In *ICML*, pages 871–878, 2010.

[13] S. Singh, R. Lewis, and A. Barto. Where do rewards come from? In *Proc. 31st Annual Conference of the Cognitive Science Society*, pages 2601–2606, 2009.

[14] M. Snel and S. Whiteson. Multi-task evolutionary shaping without pre-specified representations. In *Genetic and Evolutionary Computation Conference (GECCO'10)*, 2010.

[15] J. Sorg and S. Singh. Transfer via soft homomorphisms. In *Proc. 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 741–748, 2009.

[16] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[17] F. Tanaka and M. Yamamura. Multitask reinforcement learning on the distribution of mdps. In *Proc. 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2003)*, pages 1108–113, 2003.

[18] M. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[19] T. J. Walsh, L. Li, and M. L. Littman. Transferring state abstractions between mdps. In *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[20] E. Wiewiora, G. Cottrell, and C. Elkan. Principled methods for advising reinforcement learning agents. In *Proc. 20th International Conference on Machine Learning*, pages 792–799, 2003.

[21] A. Wilson, A. Fern, S. Ray, and P. Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *ICML*, pages 1015–1022, 2007.