Multi-Task Evolutionary Shaping Without Pre-Specified Representations

Matthijs Snel
Intelligent Autonomous Systems Group
University of Amsterdam
Science Park 107, Amsterdam, The Netherlands
m.snel@uva.nl

Shimon Whiteson
Intelligent Autonomous Systems Group
University of Amsterdam
Science Park 107, Amsterdam, The Netherlands
s.a.whiteson@uva.nl

ABSTRACT

Shaping functions can be used in multi-task reinforcement learning (RL) to incorporate knowledge from previously experienced tasks to speed up learning on a new task. So far, researchers have pre-specified a separate representation for shaping and value functions in multi-task settings. However, no work has made precise what distinguishes these representations, or what makes a good representation for either function. This paper shows two alternative methods by which an evolutionary algorithm can find a shaping function in multi-task RL without pre-specifying a separate representation. The second method, which uses an indirect fitness measure, is demonstrated to achieve similar performance to the first against a significantly lower computational cost. In addition, we define a formal categorisation of representations that makes precise what makes a good representation for shaping and value functions. We validate the categorisation with an evolutionary feature selection method and show that this method chooses the representations that our definitions predict are suitable.

Categories and Subject Descriptors

H.1 [**Systems and Information Theory**]: Value of information

Keywords

reinforcement learning, genetic algorithms, feature selection, shaping

1. INTRODUCTION

Reinforcement learning (RL) is a well-established field for learning control policies for intelligent agents. An RL *agent* engages in a sequential decision process in which it interacts with an *environment* by choosing an *action* based on sensory input, or *state*, it receives from the environment. For each action, the agent receives a real-valued reward from the environment's *reward function*. It uses this to update its *value function*, which for each state-action pair estimates the expected cumulative reward, or *return*, after picking that action in that state. The agent's goal is to maximise return by constructing a *policy* – a mapping from states to actions – that picks the highest valued actions.

RL has until recently mainly been applied to single-task settings. However, in real life, agents should be able to cope with multiple different, but related tasks. Imagine a mobile robot that should be able to find a charge station. Each new environment that it is placed in – for example, different buildings – comprises a separate task, that nonetheless shares structure with the other tasks. We call a distribution over tasks a *domain*. Thus, a charge station domain could be one composed of different buildings with randomly placed charge stations. Naïve agents might try to learn each task from scratch. A better approach is to apply knowledge from previously experienced tasks to some new task from the domain.

Shaping functions [12] capture prior knowledge about a task and speed up learning by providing the agent with additional informative "guideposts" that reward for desired actions. In a multi-task setting, this prior knowledge can be learned from previously experienced tasks in the domain. A shaping function is thus a good candidate to transfer knowledge and speed up learning across similar tasks.

Konidaris and Barto [6] were the first to take this approach. In their work, an agent in a multi-task setting uses knowledge from experienced tasks to learn an estimator of state value across tasks, which is then used as a shaping function that significantly accelerates learning on new tasks. A drawback of their method is that it requires the pre-specification of a separate sensory representation for the shaping function, which is based on those sensory features that are "consistently present and retain the same semantics across tasks" [6]. In other words, the representation consists of the features on which similarities across tasks are based. Other work that focuses on discovering a shaping function across tasks [15] similarly uses a separate representation. However, no work has made precise what distinguishes that representation from others, or what makes a good representation for a shaping function.

In this paper, we employ two alternative methods by which an evolutionary algorithm can find a shaping function in a multi-task RL setting without the pre-specification of a separate representation. In addition, we formalise what it means for a feature to retain the same semantics.

In particular, this paper makes the following main contributions. Firstly, we show that an evolutionary algorithm can find a shaping function in a multi-task setting without pre-specifying on which features the shaping function should be based. Secondly, we propose an alternative fitness measure that significantly speeds up the evolutionary algorithm, and analyse the cost / performance trade-off. Finally, we propose a formal definition of what it means for a feature to retain the same semantics across tasks, and therefore whether a feature can be used for shaping. Our experimental results show that the evolutionary algorithm finds the same features as our definitions predict.

2. RELATED WORK

Artificial shaping was introduced by Dorigo and Colombetti [1] in a robotics application, and in a more functional form by Matarić [11], who called shaping functions "progress estimators". In order to prevent divergence from the optimal policy such as in [14], Ng et al. [12] provided a rigorous theoretical foundation that showed that in order to retain the optimal policy, a shaping function should consist of a difference of potential functions over states.

Since then, there have been a number of successes in discovering shaping functions automatically for RL problems, both for single- and multi-task settings. In a single-task setting, Laud and DeJong [8] introduced "dynamic shaping", in which a shaping function based on the experimenters' rough intuition on a task gets refined while the agent interacts with the task. Other single-task research includes that of Marthi [10], in which a shaping function is discovered given a set of state- and temporal abstractions, and Grzes and Kudenko [4], in which learning the shaping function relies on an initial model of the task that is refined as the agent progresses on the task. Thus, these approaches all provide some form of prior knowledge. Elfwing et al. [2] evolve a shaping function that, when transferred to a real robot, resulted in better performance than when transferring Q-values.

All single-task work naturally involves cases in which the value- and shaping function are based on the same representation. A multi-task setting makes this impossible, since usually not all the features retain the same meaning across tasks. A shaping function in a multi-task setting is thus a powerful tool for speeding up learning since it allows for the *automatic* incorporation of prior knowledge (from previous tasks), and allows for a separation of representations, one about which knowledge can be retained across tasks (for the shaping function), and one about which knowledge has to be re-learned (for the value function).

Work on multi-task settings has so far focused on the automatic incorporation of knowledge from previous tasks. Konidaris and Barto [6] were the first to learn a shaping function automatically in a multi-task environment by learning an estimator of the value function. However, they pre-designed the representation on which the shaping function was based, just like in [15], where an "optimal reward function" equivalent to a shaping function is searched on a distribution of tasks, and [16], in which the performance of different shaping function representations is compared on a dynamic task.

Some work from the transfer learning community has focused on discovering or constructing relevant features for multi-task reinforcement learning, such as [3, 9], but these methods have focused only on value function representations, and do not exploit task-invariant knowledge in the domain. Although there are other transfer learning methods that do, their approach is not based on feature selection or construction (for an extensive overview of these and other transfer learning approaches, see [17]).

Our work differs from the previous in that it focuses on a multi-task setting in which different representations are relevant for the value and shaping function, and it does not pre-specify the representations for either function. In addition, it provides a categorisation of features that makes it clear which features should be used for both the value- *and* the shaping function, which could be used by regular feature selection methods.

3. BACKGROUND: SINGLE-TASK RL

This section provides a brief background on RL. In RL, a task is generally a **Markov decision process (MDP)**, which is a tuple $M = \langle S, A, \mathcal{P}, \gamma, \mathcal{R} \rangle$, where S is a set of **states**; A is a set of **actions**; $\mathcal{P}(s'|s,a)$ is the transition model that defines the probability of ending up in s' given state s and action a; $\gamma \in [0,1]$ is the **discount factor**; and $\mathcal{R}(r|s,a,s')$ is the **reward function** that defines a probability distribution over the immediate reward $r \in \mathbb{R}$ when taking action a in state s and transitioning to state s'. $R(s,a,s')$ denotes the expected reward in (s,a,s'), i.e., $R(s,a,s') = E[\mathcal{R}(r|s,a,s')]$.

A **policy** over a set of states S is a function $\pi : S \times A \mapsto [0,1]$, i.e. $\pi(s,a)$ gives the probability of taking action a in state s. Given such a policy, the value of taking action a in state s under that policy is the expected return (expected cumulative discounted reward) received when starting in s, taking action a, and following π thereafter, and is defined by the **action value function**, or **Q-function**, $Q^\pi(s,a)$:

$$Q^\pi(s,a) = \sum_{s'} \mathcal{P}(s'|s,a) \left[R(s,a,s') + \sum_{a'} \pi(s',a') \gamma Q^\pi(s',a') \right]$$

All optimal policies share the same **optimal Q-function**

$$Q^*(s,a) = \sum_{s'} \mathcal{P}(s'|s,a) \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right] \tag{1}$$

An often-used algorithm to learn Q-values is **Q-learning** [18], which uses the following update rule to update the Q-value for a given state-action pair:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

where α is the learning rate and r is the immediate reward received after taking a in s. Frequently, a learning agent is evaluated on the MDP for a number of **episodes**. In each episode, the agent interacts with the MDP until an "absorbing" state is encountered, which is often a goal state that the agent needs to reach.

When learning using a **shaping reward function** F in addition to the reward function R provided by the MDP, the agent finds an optimal policy for MDP $M = \langle S, A, T, \gamma, R \rangle$ by learning on a transformed MDP $M' = \langle S, A, T, \gamma, R' \rangle$, where $R' = R + F$. Ng et al. [12] showed that for an optimal policy in M' to also be optimal in M, F must be a difference of potentials: $F(s,a,s') = \gamma \Phi(s') - \Phi(s)$, where Φ is a potential function over states. Wiewiora [19] showed that using Φ for shaping is equivalent to using it to initialise the Q-function for the original MDP M.

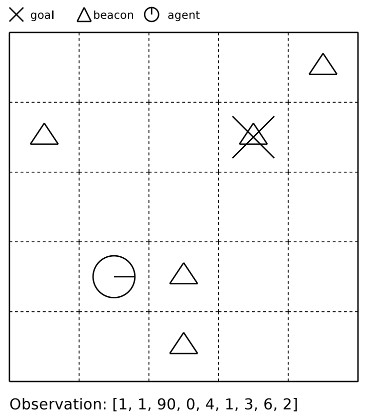


Figure 1: Simulation environment (real size 10x10).

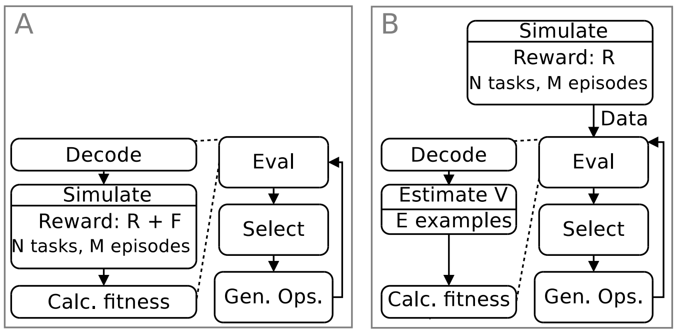


Figure 2: Methods: simulation-based fitness (a) and regression-based fitness (b). R is the base reward function of the MDP, whereas $F = \gamma\Phi(s') - \Phi(s)$ is the shaping function based on the potential function $\Phi(\cdot)$. Note that the regression-based method does not explicitly evaluate the effect of the shaping function on return.

4. A MULTI-TASK RL MODEL

To investigate the automatic discovery of shaping functions in a multi-task setting, the problem domain should be composed of a series of multiple different, but related tasks. In addition, states should be represented such that it is possible to distinguish separate features that correspond to various sensor readings. This section describes how we model this setting.

A task t is an MDP as defined previously. A *domain* D is characterized by a fixed distribution over tasks that defines the probability of a given task t to occur. The agent faces a series of tasks sampled from the domain. We extend the definition of the transition- and reward function to make explicit the dependency on the task: $\mathcal{P}(s'|s,a,t)$ is the transition model that, given task t, defines the probability of ending up in s' given state s and action a; and $R(s,a,s',t)$ is the expected reward in (s,a,s') given t: $R(s,a,s',t) = E[\mathcal{R}(r|s,a,s',t)]$. For simplicity, we assume that the distribution over tasks is such that all tasks share the same action space. However, the reward- and transition function may change between tasks.

We consider a particular implementation of this model in which a single task consists of a deterministic, non-toroidal, 10x10 grid world and a goal. The agent should find the shortest path to the goal from a given start state, and can move forward or backward, or rotate 90° left or right. Following Konidaris and Barto [6], we place five beacons in the world, with the first beacon placed at the goal location and the other four distributed randomly (figure 1). Beacons emit a signal that the agent can detect. The domain consists of a uniform distribution over tasks with the goal and the last four beacons placed randomly, and the first beacon always at the goal.

In order to be able to distinguish separate features, we define an observation function $O : S \mapsto X$, where $X \in \mathbb{R}^d$ maps the true underlying states of the MDP to factored agent observations $\mathbf{X} = (X_1, \ldots, X_d)$. For a given $s \in S$, $O(s)$ returns a corresponding observation (an assignment of values to the features) $\mathbf{x} = (x_1, \ldots, x_d)$. This essentially places our problem definition in a *Partially Observable MDP* (POMDP) framework, where the observation function provides the agent with observations that correlate with, but do not completely resolve, the underlying state. However, we assume full observability if all features are observed: the function $O(\cdot)$ is defined such that there is a one-to-one mapping from observations to states. The observation function is useful because the task might become POMDP if the agent observes less than the full feature set, which might happen if features that are deemed useless are removed.

We assume that all tasks share the same set of features. The beacon domain has nine: (x, y) is the position on the grid, *angle* is the agent's current angle, *coinflip* randomly takes on the value 0 or 1 on each step, and $beacon_i$ is Manhattan distance to beacon $i \in [1, 5]$.

5. METHODS

We wish to investigate whether it is possible to find a shaping function in a multi-task setting, such as the one specified in section 4, without pre-specifying which of the features it should be based on. Evolution is a good candidate because of its inherent ability to find properties that are consistently present and retain the same meaning across tasks; moreover, it has proven successful in finding shaping functions in a single-task setting [2].

We employ two different methods, and investigate the pros and cons of each (see figure 2 for a schematic representation). The first, *simulation-based evolution*, evaluates the fitness of a shaping function by simulating the agent on a sample of tasks and measuring how much reward it accrues. The second, *regression-based evolution*, calculates fitness by measuring how well the shaping function predicts state value $V^*(s) = \max_a Q^*(s,a)$.

5.1 Simulation-Based Evolution

It is intuitive that simulation-based evolution is a good candidate for our goal: by explicitly simulating the agent's interaction with tasks sampled from the domain, it directly optimises for shaping functions that maximise the agent's return across those tasks, and hence result in the fastest inter-task learning.

The simulation-based genetic algorithm (GA) evolves a real-valued genome that directly codes for the weights and bias of a linear potential function $\Phi(\mathbf{x}) = \mathbf{w}^\intercal\mathbf{x}+b$, where b is the bias term. For fitness evaluation (figure 2), the genome is decoded by setting each weight in the potential function to the value specified by the corresponding gene in the genome. A Q-learning agent using the decoded potential function for shaping is subsequently evaluated on a sample of N tasks from the domain for M episodes, and fitness is calculated as $\frac{1}{NM}\sum_{i=0}^{N}\sum_{j=0}^{M} R_{ij}$, where R_{ij} is the return in task i, episode j. Return is calculated using task i's base reward function, i.e., excluding the additional rewards provided by the shaping function.

5.2 Regression-Based Evolution

Simulating agent-task interaction for each fitness evaluation is costly, especially for complex or realistic task simulations that are computationally intensive. Therefore, we compare to a second, computationally cheaper, method that requires significantly less simulation time. This method does not optimise directly for maximisation of return across tasks, and is instead based on the idea that a good potential function approximates the optimal value function, as also argued by Ng et al. [12]. To see why this is intuitive, consider Wiewiora's result on potential-based shaping [19]: using a potential function for shaping is equivalent to using it to initialise the value function. Initialising the value function with an approximation of the optimal value function clearly helps to reduce convergence time, and might in addition result in a policy that is already near-optimal. Since evolving an estimator of the optimal value function does not require explicit simulation, we can save a significant amount of simulation time.

The gain compared to the simulation-based method is in the fitness evaluation, which only requires simulation of the agent-task interaction once, before starting the GA (figure 2). During simulation, a sample of N tasks is taken and the agent is run for M episodes on each task. A data set of E examples is then constructed by taking the union of the agent's value tables from each task: thus, each example is an observation-value pair. Fitness is calculated as $-\frac{1}{E}\sum_{i=0}^{E}|\hat{V_i} - V_i|$, where $\hat{V_i}$ is the estimator's estimate of V_i, the value the agent learned for observation i.

6. FEATURE CATEGORY DEFINITIONS

As discussed previously, research that has focused on discovering a shaping function across tasks has pre-specified its representation to be based on those features that retain the same semantics across tasks. However, a precise definition of what this means has never been established. To the best of our knowledge, we are the first to propose a categorisation of features in multi-task RL settings based on their properties, and provide a formal definition of those properties. This definition will help us form a hypothesis about the features that methods for finding shaping functions will find important, as well as increase understanding of different uses of information in multi-task settings. In addition, it could facilitate the automatic discovery of shaping functions without pre-specifying their representation.

It is intuitive that there is a difference in the features from the beacon domain. First and foremost, *coinflip* is obviously useless since it is random. Feature $beacon_1$ is always placed at the goal, so a reduction in $beacon_1$ value always means the agent is closer to the goal. Hence this feature retains the same meaning across tasks. We call such a feature *domain informative*.

The other beacons are scattered randomly in each task, so they do not seem to provide any useful information. The $x, y, angle$ features are useful within a task because they uniquely identify each state; therefore, once the agent has learned the value of each position and angle, it has solved the task. However, the agent will have to re-learn this value in every new task, because the goal changes position randomly. We call features that are informative within a task but do not retain the same meaning across tasks *task informative.*

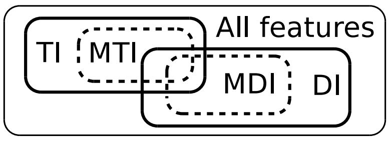


Figure 3: Venn diagram of feature categories. (M)TI=(minimal) task informative; (M)DI=(minimal) domain informative.

Our formal definitions follow Parr et al. [13] in defining a feature to be informative if it helps predict the immediate reward or next state, given an observation and an action. By this definition, a feature is informative if it provides information on the distribution over possible next states and immediate rewards. We can use this to formally define the feature categories as follows:

Task-informative features are features that, when observed *on their own*, provide information on the distribution over possible next states and expected immediate reward, i.e., X_i is a task-informative feature if for a given task t there exists an $a \in A$ for which $\mathcal{P}(s'|a,t) \neq \mathcal{P}(s'|X_i,a,t)$ or $E\left[\mathcal{R}(r|a,s',t)\right] \neq E\left[\mathcal{R}(r|X_i,a,s',t)\right]$.

Thus, the feature may or may not be seen as informative given other features present in the observation (which potentially provide the same information), but by this definition the feature provides information when observed on its own, and could thus be useful.

Domain-informative features are features that, when observed *on their own*, provide information on the distribution over possible next states and expected immediate reward *across tasks*, i.e. a feature X_i is domain informative if $\mathcal{P}(s'|a) \neq \mathcal{P}(s'|X_i,a)$ or $E[\mathcal{R}(r|a,s')] \neq E[\mathcal{R}(r|X_i,a,s')]$. Note the absence of t in this definition, since the feature provides information on distributions across tasks.

Note that these categories are not mutually exclusive: a domain-informative feature might well also be a task-informative feature, such as $beacon_1$ in our example. A purely domain informative feature is one that does not provide information within a task, yet can be used to distinguish (classes of) tasks in the domain. For example, if in tasks with a red floor a negative reward is received when taking action a_1, but in tasks with a green floor a positive reward is received for a_1, then floor colour is domain informative. It is not task informative because the observed colour is the same on every time step within a given task.

Also note that, as mentioned earlier, a feature may or may not be seen as informative given other features present in the observation. For example, if we were to add another feature that encodes the angle in radians instead of in degrees, this would not add any information. The feature is task informative of itself, but is redundant given the other features present. There is a minimal subset of task-informative features, that we label *MTI-features*, that makes all other task-informative features redundant within a task.

Since we are assuming full observability, in our multi-task RL model this is a set of features that are necessary for full observability. In the beacon domain there is one such set, namely $x, y, angle$: this set uniquely identifies each state, and removing one of the features would result in partial observability. There is a similar subset of domain-informative features that makes all other domain-informative features redundant, which we have labelled *MDI-features* (figure 3).

There is an alternative way to formulate the definitions that enables an agent to distinguish task- and domain-informative features based on the Q-values it has learned, which is practical in settings in which the agent does not have direct access to the underlying states of the MDP.

Rewriting equation 1 using the observation function defined earlier, and substituting $\mathbf{x}$ for $O(s)$ gives

$$Q^*(s,a) = \sum_{s'} \mathcal{P}(s'|\mathbf{x},a) \left[E[\mathcal{R}(r|\mathbf{x},a,s')] + \gamma \max_{a'} Q^*(s',a') \right]. \tag{2}$$

Note that the optimal Q-values solely depend on $\mathcal{P}(s'|\mathbf{x},a)$ and $E[\mathcal{R}(r|\mathbf{x},a,s')]$. Since our definitions of feature categories also solely depend on these quantities, they can be rewritten in terms of the information the features provide on optimal Q-values.

By treating Q as a random variable and measuring the task or domain "informativeness" of a feature by the change in certainty it causes about Q^* (when all features are observed, certainty is 100%, but the fewer features are observed, the more uncertain an estimator will be about Q^*), the definitions can be rewritten as follows:

Task-informative features X_i is a task informative feature if for a given task t there exists an $a \in A$ for which

$$p(Q^*|a,t) \neq p(Q^*|X_i,a,t). \tag{3}$$

Domain-informative features X_i is domain informative if there exists an $a \in A$ for which

$$p(Q^*|a) \neq p(Q^*|X_i,a). \tag{4}$$

7. EXPERIMENTS AND RESULTS

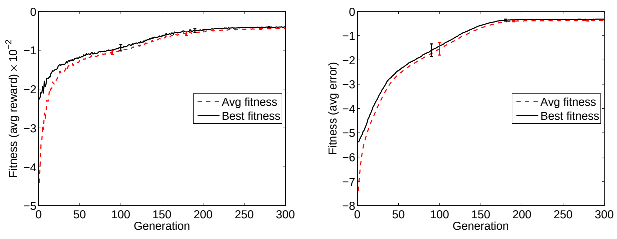
As explained in section 5, we designed two alternative methods for finding shaping functions in our domain: a simulation-based method, which explicitly optimises for shaping functions that result in the highest inter-task return; and a computationally cheaper regression-based method, which optimises for good predictors of V^* across tasks.

In the first set of experiments, we evaluate the performance of both methods and discuss the cost of each. In the second, we use our definitions of feature categories to form a hypothesis about which features should be used for the value function (task informative) and shaping function (domain informative), and design experiments to verify this hypothesis.

7.1 Experimental Set-up

All experiments are run on the shortest-path-to-goal grid world problem as described in section 4. The base reward function R provides a reward of -1 on every time step and a reward of 0 for reaching the goal.

We use a tile coding of the $\langle x, y, angle \rangle$ features for the linear function approximator with one tile for every feature



(a) Simulation-based GA (b) Regression-based GA

Figure 4: GA performance. Error bar is 1 stdev.

value. Thus, together with the 5 beacons and the *coinflip* feature, the agent observes a total of 30 features.

The GA therefore uses real-valued chromosomes of length 31 (including the bias weight) in population of 60 individuals, and employs binary tournament selection. Mutation is applied with $p = 0.2$ and adds a uniformly distributed value $\delta \in [-0.03, 0.03]$ to the weight. No crossover is used.

Fitness evaluation for the simulation-based approach takes place with a Q-learning agent with learning rate $\alpha = 0.2$, discount factor $\gamma = 1$, and an ϵ-greedy policy with $\epsilon = 0.1$. The agent is evaluated on a sample of 5 tasks for 25 episodes per task.

For the regression-based method, sample data is collected once, before starting the GA, by running a Q-learning agent with learning rate $\alpha = 0.3$, discount factor $\gamma = 1$, and an ϵ-greedy policy with $\epsilon = 0.9$ on a sample of 20 tasks for 3000 episodes per task.

The GA is run for 300 generations. Once it has finished, the potential function that the final population champion codes for is used in a shaping function for a Q-learning agent with $\alpha = 0.2$, $\gamma = 1$, $\epsilon = 0.1$ that is run on a sample of 5 tasks for 1000 episodes per task.

All results in the subsequent sections are averages over 10 independent runs.

7.2 Discovering a Shaping Function

Figure 4 shows that both methods significantly improve average population fitness and population champion fitness for both methods before converging on a stable fitness level after 300 generations. Average population fitness is very close to that of the champion, indicating that all individuals in the population encode for functions that are good for shaping (simulation-based method) or good estimators of V^* (regression-based method). To accurately compare the two and find out whether the estimators are also good for shaping, figure 5 shows how the performance of a Q-learning agent using the best function found by both methods for shaping compares to the performance of a Q-learning agent that learns without shaping. As expected, performance with shaping is much better than performance without shaping.

Furthermore, the regression-based shaping function performs nearly as well as the simulation-based one. However, the simulation-based function causes the agent to find the goal substantially quicker in the first episode: 140 steps on average, versus 195 for the regression-based function. In addition, the simulation-based function reaches a slightly lower average number of steps after 1000 episodes: for the final episode, the agent needs 1.6 fewer steps on average than for the regression-based function. Both these differences are significant ($p < 0.01$, Student t-test). We provide one pos-

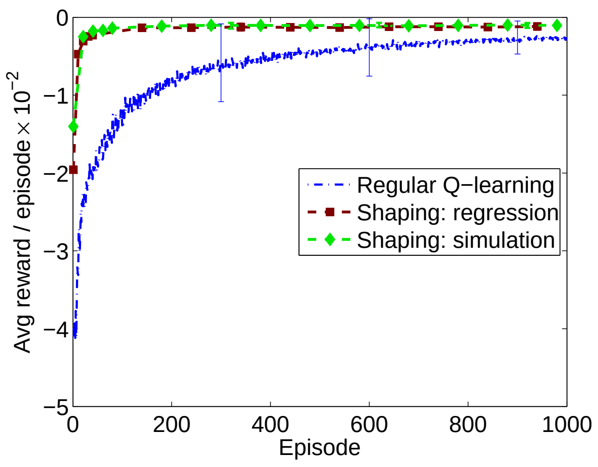


Figure 5: Performance of the shaping functions found by the two methods compared to performance of a Q-learner without shaping. Error bar is 1 stdev.

sible explanation for this difference in performance in the next section, where we look at the weights found by both methods.

Each method's computation time mainly depends on two components: simulation of agent-task interaction and computing the output of the potential function $\Phi(\cdot)$. The time the first component requires is mainly determined by how many episodes have to be simulated, whereas the time for the second depends on how complex the potential function is and how many times its output is computed. In our experiments, the simulation-based method uses 2.25×10^6 episodes and about 2.03×10^8 function evaluations for a single run, versus 6.00×10^4 and 2.88×10^8 for the regression-based method. The number of episodes simulated by the simulation-based method is thus around two orders of magnitude larger than for the regression-based method. Although the exact difference depends on the experimental setting, it is clear that the simulation-based method generally requires significantly more simulation than the regression-based one. Thus, even though the latter performs slightly worse than the former, it can be a valuable alternative in cases where agent-task interaction is relatively expensive compared to shaping function evaluation, which is the case in most settings.

7.3 Evaluation of Feature Definitions

Now that we know that evolution can find a shaping function without pre-specifying a separate representation in at least two different ways, in the second set of experiments we investigate whether evolution reaches the same conclusion about feature relevance as our definitions predict. We do this by calculating the "informativeness" of each feature, using our definitions, to form a hypothesis about which features should be used for the value function (task informative) and shaping function (domain informative), and test this hypothesis by looking at the weights evolved by the simulation- and regression-based method result of section 7.2, and by using an evolutionary method that explicitly selects for features that make learning those functions easy for gradient descent.

To determine what features our definitions predict are domain informative, we use equations 3 and 4 to calculate a feature's information gain with respect to Q^*, based on data sampled in the same way as for the regression-based method. We calculate information gain based on the Kullback-Leibler divergence [7], which measures the distance in bits between two distributions μ and σ as $D_{KL}(\mu||\sigma) = \sum_x \mu(x) \log_2 \frac{\mu(x)}{\sigma(x)}$. To calculate the information gain of a single feature, we follow a method similar to that used in [5].

7.3.1 *Domain-Informative Features*

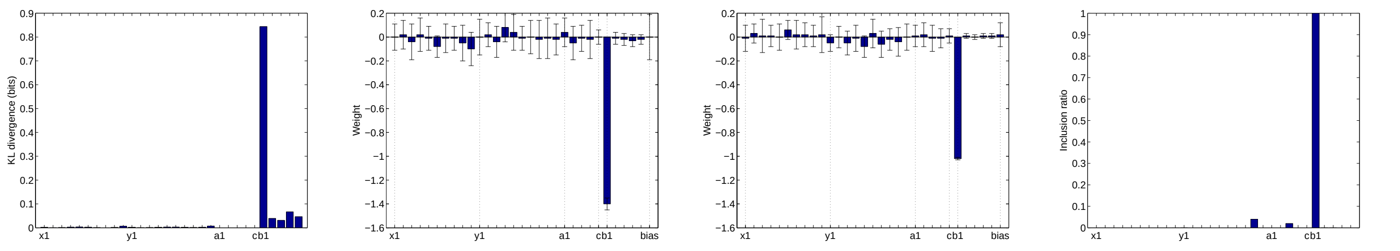
For the domain-informative case, for calculating what our definitions predict we define $\delta(x, a) = D_{KL}(p(Q|x, a)||p(Q|a))$, which gives the gain on Q for a single feature value. In order to reach the gain for a feature, we sum over all possible values of the feature, multiplied by their prior. We need to do this for each action, and reach a single value for the gain by taking the average over all actions: $\Delta(X) = \sum_x \frac{p(x)}{|A|} \sum_{a \in A} \delta(x, a)$. This results in the predictions as shown in figure 6a.

The weights evolved by the two methods of the first set of experiments are shown in figures 6b and 6c. As expected, both methods base their potential function exclusively on $beacon_1$, the only domain-informative feature in our experiments: the weights for the other features show only an insignificant deviation from 0. This confirms the predictions of our definitions.

Looking at the magnitude of the weights, simulation-based evolution finds a weight of -1.4, whereas regression-based evolution sticks to -1.0, which is probably the cause of the difference in performance of the shaping functions. It is not immediately clear why the difference in weights arises. The one found by regression is "correct" since there is an almost one-to-one mapping from goal distance to optimal Q-value. The one for the simulation-based method might result in a pessimistic initialisation of Q-values for states far away from the goal, but since the agent spends most of its time relatively close to the goal, this does not adversely affect performance. In fact, the results show that having a weight of greater magnitude results in a better shaping function. This difference in weights may, however, be particular to our domain.

In order to investigate whether an evolutionary algorithm that explicitly selects for features for the shaping function finds the same result as our predictions, we let the GA evolve a population of 100 binary chromosomes that indicates which features the functions should include. Single-point crossover is applied with $p = 0.8$, and binary mutation with $p = 0.001$. Just like in the regression-based method, a dataset is created by letting the agent interact with the task before starting the GA. For fitness evaluation, a linear estimator is created with inputs corresponding to the features that are switched on in the genome. The data is separated into a training and test set, after which the estimator is trained for one epoch by on-line gradient descent with learning rate $\alpha = 0.0006$ on the training set and tested on the test set. Like for the regression-based method, fitness is the negative RMSE achieved on the test set. Results are averaged over 30 independent runs.

The results for the domain-informative case are shown in figure 6d. Not surprisingly, these results reconfirm the predictions of our definitions. Therefore, we have shown that our formalisation of domain-informative features is accurate.



(a) Prediction of definitions (b) Weights for the simulation-based function (c) Weights for the regression-based function (d) Explicit feature selection by GA

Figure 6: Domain-informative features as predicted by our definitions (a), according to the weights found by the simulation- (b) and regression-based (c) methods, and as found by explicit evolutionary feature selection (d). Error bars indicate 1 stdev. Y-axis in (d) is ratio of times a feature was included in an individual.

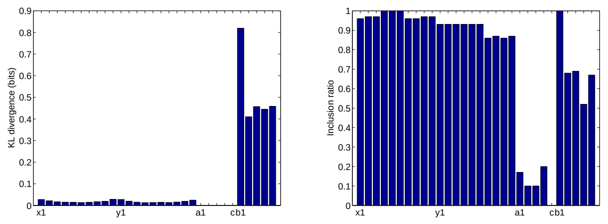
7.3.2 Task-Informative Features

Contrary to the previous experiments, for distinguishing task-informative features evolution should select for features that result in the best estimator of Q *for a single task*. Therefore, instead of treating the sampled dataset as a whole, the estimator is trained and tested on the data for each sampled task separately. For calculating information gain, we now define $\delta(x, a, t) = D_{KL}(p(Q|x, a, t)||p(Q|a, t))$ and $\Delta(X) = \frac{1}{N}\sum_t^N \sum_x \frac{p(x)}{|A|} \sum_{a \in A} \delta(x, a, t)$ (since in our domain tasks are uniformly distributed, we can divide by the number of sampled tasks).

The resulting predictions are shown in figure 7a. As expected, the x and y features are more task than domain informative, although perhaps not as much as one would expect. However, we verified that if the separate features are treated as a single feature (so one x instead of 10, and the same for y and *angle*) they are more informative than beacon 1. Contrary to our expectations, *angle* is not deemed very task informative (not even when the separate *angle* features are grouped into one), even though it is necessary for full observability. This might be because in order to reach the goal, the agent only needs to rotate at most once, and thus needs angle-related information only once in theory. Even though it is hard to make out from the figure, the angle features are more informative than *coin*, which does not provide any information at all.

Beacon 1 is task as well as domain informative, as expected. What might at first seem counter-intuitive is that the other, randomly scattered, beacons are also fairly task informative. However, beacons do not change position within a task, so they can be used as "landmarks" for estimating position, and thus provide a fair amount of information.

Fitness of the GA is calculated as the negative of the *average* RMSE on each task, and thus selects for features that on average result in the best intra-task learning. Figure 7b shows the results. All ratios in the figure are significantly different from random ($p < 0.01$, Pearson chi square test), except beacon 4. Again, the positional features and beacon 1 are very task informative. Note that, even though each separate x and y feature provides little information, evolution is able to detect that together they are very informative, which is why x and y features are almost always selected. The other evolutionary findings also match what our definitions predict: the angle features are included in only 10 - 20% of the cases, the coinflip not at all, and the randomly scattered beacons about 50 - 70% of the time.



(a) Prediction of definitions (b) Selection by GA

Figure 7: Task-informative features as predicted by our definitions (a), and as selected by the GA (b). Y-axis in (b) indicates the ratio of times a feature was included in an individual.

These results show that the features deemed task informative by evolutionary feature selection also match what our definitions predict. Although the definitions apply to single features and a set of features may be more informative than the sum of the separate features, this is a general characteristic of any feature selection problem and does not contradict the fact that each separate feature is task informative. The results therefore show the validity of the definitions in our domain.

8. DISCUSSION AND CONCLUSION

This paper focuses on discovering shaping functions to speed up learning in a multi-task RL setting in which different representations are relevant for the value- and shaping function, without pre-specifying the representations for either function. In addition, it provides a definition of features that makes it clear which features should be used for both the value- and the shaping function.

We investigate two different evolutionary methods for finding a shaping function, one which evaluates fitness by explicit simulation of agent-task interaction, and a regression-based method that evolves value function estimators across tasks. Even though the shaping function that the latter method finds performs slightly worse, regression-based evolution is a valuable alternative in settings where simulation of the task is relatively expensive compared to computing shaping function output, which is the case in most settings. Furthermore, the difference in performance between the two methods was caused by a difference in weights for the shaping function that may be particular to our domain. Future work could investigate whether this difference also applies to other problem domains.

We propose a categorisation of features into task informative and domain informative features and provide a formal definition of each category. Task-informative features, in particular the minimal subset of these features, are relevant for the value function. Domain-informative features are useful for the shaping function. In addition, using our definitions we can compute exactly *how* informative a feature is within or across tasks. As shown in section 7.3, this can sometimes lead to counter-intuitive results. However, our definitions are validated by an evolutionary method that selects for the same features as those deemed informative by our definitions. Also note that, although this paper uses the Q-learning algorithm, the feature categories are algorithm-independent.

Since this paper presents first results on the regression-based GA and feature categories, we have intentionally kept the implementation of the beacon domain simple. In particular, we have assumed a single state space for the domain. The state space size may change through either a different number of features (e.g., additional sensors) in the observation, or, more likely, a change in feature value range (e.g., different environment size). Since a feature's relevance is based on its prediction of value, the latter should not affect feature categorisation, and previous work [6] has already shown the effectiveness of shaping functions across tasks of different size. The effect of the first scenario depends on the prior probability of sampling a task with additional features. Although these effects are certainly worth considering, space limitations prevent in-depth discussion. We do, however, intend to more explicitly address both cases in future work.

Although the evolutionary selection method we use is good for evaluating the definitions we propose, it is too costly to be beneficial. However, we believe the framework we propose is comprehensive enough to be used by more powerful feature selection methods such as [5] that result in a lower total cost of learning, and we are currently investigating the use of such a method. Such a feature selection method could also use the definition of task informativeness in value function approximation scenarios to reduce the dimensionality of the approximator and thus further speed up learning. This could be especially useful in high-dimensional applications such as visual robotics, where observations consist of video input.

In the beacon domain, the domain-informative feature used for shaping is also task relevant. This allows the feature to be used in a shaping function based on a potential function over states. As indicated in section 6, there are, however, also examples of domain-informative features that are not task informative. These kind of features cannot be used by a potential function over states, but could possibly be used by a potential function over state-action pairs [20]. Thus, our framework also makes clear in which settings either type of potential function could be used, and we intend to investigate both types of settings in future work.

Acknowledgements

We thank George Konidaris and the anonymous reviewers for useful comments on an earlier version of this article.

References

[1] M. Dorigo and M. Colombetti. Robot shaping: developing autonomous agents through learning. *Artificial Intelligence*, 71(2):321–370, 1994.

[2] 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.

[3] K. Ferguson and S. Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*, 2006.

[4] M. Grzes and D. Kudenko. Learning shaping rewards in model-based reinforcement learning. In *Proc. AAMAS 2009 Workshop on Adaptive Learning Agents*, 2009.

[5] D. Koller and M. Sahami. Toward optimal feature selection. In *Proc. 13th International Conference on Machine Learning*, pages 284–292, 1996.

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

[7] S. Kullback and R. Leibler. On information and sufficiency. *Annals of Math. Statistics*, 22:76–86, 1951.

[8] A. Laud and G. DeJong. Reinforcement learning and shaping: Encouraging intended behaviors. In *Proc. 19th Int. Conference on Machine Learning*, pages 355–362, 2002.

[9] A. Lazaric. *Knowledge Transfer in Reinforcement Learning*. PhD thesis, Politecnico di Milano, 2008.

[10] B. Marthi. Automatic shaping and decomposition of reward functions. In *Proc. 24th International Conference on Machine Learning*, pages 601–608, 2007.

[11] M. J. Matarić. Reward functions for accelerated learning. In *Proc. 11th Int. Conference on Machine Learning*, 1994.

[12] 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.

[13] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proc. 25th International Conference on Machine Learning*, pages 752–759, 2008.

[14] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *Proc. 15th International Conference on Machine Learning*, 1998.

[15] 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.

[16] M. Snel and G. Hayes. Evolution of valence systems in an unstable environment. In *Proc. 10th Int. Conference on Simulation of Adaptive Behavior*, pages 12–21, 2008.

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

[18] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[19] E. Wiewiora. Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19:205–208, 2003.

[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.