Auton Agent Multi-Agent Syst (2014) 28:637–681
DOI 10.1007/s10458-013-9235-z

Learning potential functions and their representations for multi-task reinforcement learning

Matthijs Snel · Shimon Whiteson

Published online: 5 September 2013

Abstract In multi-task learning, there are roughly two approaches to discovering representations. The first is to discover *task relevant* representations, i.e., those that compactly represent solutions to particular tasks. The second is to discover *domain relevant* representations, i.e., those that compactly represent knowledge that remains invariant across many tasks. In this article, we propose a new approach to multi-task learning that captures domain-relevant knowledge by learning potential-based shaping functions, which augment a task's reward function with artificial rewards. We address two key issues that arise when deriving potential functions. The first is what kind of target function the potential function should approximate; we propose three such targets and show empirically that which one is best depends critically on the domain and learning parameters. The second issue is the representation for the potential function. This article introduces the notion of k-relevance, the expected relevance of a representation on a sample sequence of k tasks, and argues that this is a unifying definition of relevance of which both task and domain relevance are special cases. We prove formally that, under certain assumptions, k-relevance converges monotonically to a fixed point as k increases, and use this property to derive *Feature Selection Through Extrapolation of k-relevance* (FS-TEK), a novel feature-selection algorithm. We demonstrate empirically the benefit of FS-TEK on artificial domains.

Keywords Multi-task reinforcement learning · Feature selection · Abstraction · Potential-based shaping · Transfer learning

1 Introduction

Real-world autonomous agents constantly face problems with various degrees of similarity to problems encountered before. Often, exploiting these similarities is vital to solving the new

M. Snel (✉) · S. Whiteson
Intelligent Systems Lab Amsterdam, Universiteit van Amsterdam,
Amsterdam, The Netherlands
e-mail: m.snel@uva.nl

S. Whiteson
e-mail: s.a.whiteson@uva.nl

problem with acceptable cost in terms of time, money, or damage incurred. For example, a person learning a new language can do so more quickly if he understands the general structure of language; a robot navigating a new building should not need to crash into obstacles to learn that doing so is bad.

This article considers how an agent facing a sequence of tasks can best exploit its experience with previous tasks to speed learning on new tasks, via two complementary approaches. First, by extracting knowledge from this experience in a way that can be leveraged by learning algorithms, and second, by automatically discovering good representations for that knowledge; for example, a subset of the agent's sensory features. We consider a *reinforcement learning* (RL) [73] setting in which agents learn control policies, i.e., mappings from states to actions, for sequential decision problems based on feedback in the form of reward. Typically, the agent tries to estimate the value of a state under a given policy as the sum of future rewards, or *return*, it can expect under that policy.

Traditionally, the aim has been to converge to the optimal policy, which maximizes expected return. However, expected *online return* (return incurred while the agent is learning and interacting with the environment), rather than convergence, is often more important. By guiding the agent's exploration strategy, the application of prior knowledge to the task is one tool for improving online return.

In *transfer learning*, prior knowledge is derived from previous tasks seen by the agent or other agents. More specifically, transfer learning aims to improve performance on a set of *target tasks* by leveraging experience from a set of *source tasks*. Clearly, the target tasks must be related to the source tasks for transfer to have an expected benefit. In *multi-task reinforcement learning* (MTRL), this relationship is formalized through a *domain*, a distribution over tasks from which the source and target tasks are independently drawn [76,86]. Even so, relatedness can come in different forms. This article focuses on discovering shared representational structure between tasks and the knowledge captured by that structure.

Discovering a single compact representation (e.g., a subset of the agent's sensory features) that is able to capture individual task solutions in the entire domain is the objective of many multi-task methods (e.g. [18,30,70,83]). We call this a *task-relevant* representation: a single representation with which a different function is learned in each task. Its power lies in its compactness, which makes each task easier to learn.

Other approaches utilize a cross-task representation that, while not necessarily useful for representing task solutions, captures task-invariant knowledge (e.g. [8,20,22,34,62]). We call this a *domain-relevant* representation, which can serve as a basis for a *single* cross-task function that captures (approximately) invariant domain knowledge. Roughly speaking, such a function obviates the need to re-learn task-invariant knowledge and biases the agent's behavior.

The key insight underlying the work presented in this article is that task-relevant and domain-relevant representations are fundamentally different: not only can they capture different concepts, but they can employ different features. For example, consider a robot-navigation domain. In each task, the robot is placed in a different building in which it must locate a bomb that is placed at a fixed location that depends on the building. An optimal policy for a given task need condition only on the robot's position in the building; position is therefore task relevant. However, in each task, the robot must re-learn how position relates to the bomb's location. By contrast, while distance sensors are not needed in a task-specific policy, they can be used to represent the task-invariant knowledge that crashing into obstacles should be avoided, and are therefore domain relevant.

Some approaches learn a single new representation that may combine task-relevant and domain-relevant representations [2,5,8,40,79]. For example, instead of training a separate neural network per supervised learning (SL) task, Caruana [8] trains a single network on all tasks in parallel. Learning benefits from the shared hidden layer between tasks, which captures task-invariant knowledge, but the network can also represent task-specific solutions through the separate task weights. However, such approaches have several limitations. First, task solutions may interfere with each other and with the task-invariant knowledge [9]. Second, any penalty term for model complexity [2,40] affects both task and domain-relevant representations simultaneously. Lastly, mapping the original problem representation to a new one makes it harder to interpret the solutions and decipher which knowledge is domain-relevant.

These problems can be avoided by maintaining separate task-relevant and domain-relevant representations. In each task, the agent learns a policy using the task-relevant representation and is aided by a cross-task function using the domain-relevant representation. This approach also makes it easier to override the bias of the cross-task function, which is important when that bias proves incorrect [9]. The cross-task function can be represented as, e.g., advice rules [81], or a *shaping function* [34,53], the approach we focus on in this article.

Shaping functions, which augment a task's reward function with additional artificial rewards, have proven effective in single-task [53], multi-agent [4] and multi-task [34,68,69] problems, with applications to, e.g., foraging by real robots [15], robot soccer [10], and real-time strategy games [52]. Ng et al. [53] showed that, to preserve the optimal policy of the ground task, the shaping function should consist of a difference of *potentials* $\Phi : \mathbf{X} \to \mathbb{R}$, which, like value functions, specify the desirability of a given state. In MTRL, shaping functions can be learned automatically by deriving the potential function from the source tasks [34,68,69], e.g., the navigating robot described above could learn a shaping function that discourages bumping into obstacles.

The primary aim of this article is to address the two key steps involved in trying to maximize online return through potential functions in MTRL. The first step is selecting the appropriate target for the potential function to approximate. This question does not arise in SL, since the targets are given and thus any single cross-task function strives to minimize a loss function with respect to all tasks' targets simultaneously. An intuitive choice of target in MTRL is the optimal value function of each task; approximating this target leads to the solution that is closest in expectation to the optimal solution of the unknown next task. However, there is no guarantee that using a potential function that approximates this target leads to the best online return. Therefore, in Sect. 3, we propose three different targets.

Given a target, the second key step is to find the representation that approximates this target as closely as possible while at the same time generalizing well given limited domain information (observed tasks); i.e., to select a domain-relevant representation. Previous work on multi-task shaping [34] relied on manually designed representations. Other MTRL work that employed an analogue of domain-relevant representations also manually designed them [21,22,63], or learned them but did not learn a cross-task function [20]. Therefore, in Sect. 4, we introduce the notion of k-relevance, the expected relevance on a sequence of k tasks sampled from the domain, and argue that this is a unifying definition of relevance of which both task and domain relevance are special cases. We prove that, under certain assumptions, k-relevance is an approximately exponential function of k, and use this property to derive *Feature Selection Through Extrapolation of k-relevance* (FS-TEK), a novel feature-selection algorithm that applies to both tabular representations and linear function approximators with binary features. The key insight behind FS-TEK is that change in relevance observed on task sequences of increasing length can be extrapolated to more accurately predict domain relevance.

Finally, we present an empirical analysis that evaluates these two steps on multiple domains, including one that requires function approximation. First, we show that which potential function is best depends critically on the domain and learning parameters. Then, we demonstrate empirically the benefit of FS-TEK on these same domains.

The remainder of the article is structured as follows. The next section addresses some essential background on shaping functions and RL. Thereafter, Sects. 3 and 4 discuss the theoretical aspects of the potential functions and notion of relevance that we propose. Section 5 provides an experimental validation of these concepts. The article concludes with an overview of related work in Sect. 6 and a discussion in Sect. 7.

2 Background

This section introduces the notation used in the article and provides further background on shaping functions and the standard framework for solving RL problems. In the last subsection, we formalize the problem setting considered in this article.

2.1 Reinforcement learning

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*. The agent's goal is to construct a *policy*—a mapping from states to actions—that maximizes the expected cumulative reward, or expected *return*. *Value-based methods* do so by learning a *value function*, which for each state–action pair estimates the expected return after picking that action in that state.

In this article, we assume a factored state representation [7], in which each state is described using a set X of state variables or features: $\mathsf{X} = \{X_1, X_2, \ldots, X_p\}$. An assignment of a specific value to feature X_i is denoted by x_i, with the vector $\mathbf{x} = (x_1, x_2, \ldots, x_p)$ describing one state. Unless specified otherwise, uppercase letters (X) denote random variables, and lowercase letters (x) denote their values. Similarly, lowercase boldface denotes vectors of values ($\mathbf{x}$), and uppercase boldface ($\mathbf{X}$) denotes both a random vector and the set of all possible $\mathbf{x}$. See Table 2 of Appendix 1 for glossary of terms used in this section and throughout the article.

A RL task is typically modeled as a *Markov decision process* (MDP), a tuple $m = \langle \mathbf{X}_m, \mathsf{A}, P_m, R_m \rangle$ with set of states $\mathbf{X}_m$ and set of actions A. For convenience, X^+ denotes the feature set that includes the action as a feature and $\mathbf{X}_m^+ = \mathbf{X}_m \times \mathsf{A}$ denotes the state–action space of a task. Since this article focuses on MTRL, we subscript the entities specific to a given task with m to help distinguish between tasks. We assume the same feature and action set for all tasks, and that all actions are allowed in any $\mathbf{x} \in \mathbf{X}_m$. Given state $\mathbf{x}$ in task m, the probability of transitioning to state $\mathbf{x}'$ given action a is $P_m(\mathbf{x}, a, \mathbf{x}')$, or $P_m^{\mathbf{x}a\mathbf{x}'}$, and results in expected reward $R_m(\mathbf{x}, a, \mathbf{x}')$, or $R_m^{\mathbf{x}a\mathbf{x}'}$.

The aim of an RL agent is to maximize the return R_t, a discounted sum over rewards:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \tag{1}$$

where r_{t+1} is the reward received after taking action a_t in state $\mathbf{x}_t$ at time step t and γ is a discount factor in [0, 1] that determines the relative importance of future rewards.

The return depends on the agent's policy $\pi : \mathbf{X}_m \times \mathsf{A} \rightarrow [0, 1]$ s.t. $\sum_a \pi(\mathbf{x}, a) = 1$, which assigns a probability to action a in state $\mathbf{x}$. The value of state–action pair $(\mathbf{x}, a)$ under policy π is the expected return when starting in $\mathbf{x}$, taking a, and following π thereafter, and is denoted by $Q_m : \mathbf{X}_m^+ \rightarrow \mathbb{R}$. The optimal value function, which defines the maximum expected return the agent can accrue, is defined by the *Bellman optimality equation*:

$$Q_m^*(\mathbf{x}, a) = \sum_{\mathbf{x}' \in \mathbf{X}_m} P_m^{\mathbf{x}a\mathbf{x}'} \left[R_m^{\mathbf{x}a\mathbf{x}'} + \max_{a'} Q_m^*(\mathbf{x}', a') \right]. \tag{2}$$

An optimal policy π^* achieves maximum expected return and can be derived from the optimal Q-function by setting $\pi^*(\mathbf{x}, \mathrm{argmax}_a\, Q_m^*(\mathbf{x}, a)) = 1$. It follows that the optimal value of state $\mathbf{x}$ is $V_m^*(\mathbf{x}) = \max_a Q_m^*(\mathbf{x}, a)$.

Solving (2) explicitly requires a model of the MDP; this is the domain of *planning* methods such as dynamic programming [6]. For example, *policy iteration* starts out with a random policy, and iteratively computes the value function for the policy and makes the policy greedy with respect to the new value function, until convergence. On the other hand, *value iteration* works by iteratively computing solutions to (2) directly.

In RL problems, however, no model is available and the agent must learn about its environment by interacting with it. *Model-based* RL methods do so by iteratively learning a model of the environment and improving the policy by planning on the estimated model; in contrast, *model-free* methods, which we employ in this article, estimate a value function or policy directly from interaction with the environment [73].

A widely used class of model-free methods is *temporal-difference* (TD) learning [72]. When used for control, the TD update takes the form

$$Q(\mathbf{x}, a) \leftarrow Q(\mathbf{x}, a) + \alpha\delta, \tag{3}$$

where α is a learning rate and δ the *TD error*. There are various approaches to computing the latter, depending on which TD control algorithm is used. Two of the most popular are Sarsa [58,64], and Q-Learning [84]. For Sarsa,

$$\delta = r + \gamma Q(\mathbf{x}', a') - Q(\mathbf{x}, a), \tag{4}$$

where r is the reward incurred, $\mathbf{x}'$ the next state and a' the action the agent took at $\mathbf{x}'$. For Q-Learning,

$$\delta = r + \gamma \max_{a^*} Q(\mathbf{x}', a^*) - Q(\mathbf{x}, a). \tag{5}$$

Basic RL methods do not cope well with tasks with large state spaces. One tool for speeding up learning in such problems is the use of *eligibility traces* [73]. Here, a state–action pair gets assigned a *trace* with value one (so-called *replacing traces*) when visited; each trace is decayed by a factor $\gamma\lambda$ on each time step, where γ is the discount factor of the MDP and λ is a decay parameter. At each time step, instead of just updating the most recent state–action pair, *all* state–action pairs with traces significantly above zero are updated as $Q(\mathbf{x}, a) \leftarrow Q(\mathbf{x}, a) + \alpha\delta e(\mathbf{x}, a)$, where $e(\mathbf{x}, a)$ is the trace value. When combined with eligibility traces, Sarsa and Q-Learning are called Sarsa(λ) and Q(λ), respectively.

Function approximators can also improve performance in large state spaces and are required for continuous state spaces. Linear approximators represent the Q-value for a given state–action pair as $\phi(\mathbf{x}, a)^{\mathrm{T}}\theta$, where $\phi(\mathbf{x}, a)$ maps the state–action pair (possibly nonlinearly) to a feature vector representation, and θ is the vector of weights. Combined with eligibility traces, the update takes the form $\theta \leftarrow \theta + \alpha\delta\mathbf{e}$, where $\mathbf{e}$ is the vector of traces. In this article we employ *tile coding* [1,73], a linear function approximator based on a binary feature vector.

2.2 Shaping

The concept of shaping stems from the field of operant conditioning, where it denotes a training procedure of rewarding successive approximations to a desired behavior [67]. In RL, it may refer either to training the agent on successive tasks of increasing complexity, until the desired complexity is reached [17,27,57,59,61,65], or, more commonly, to supplementing the MDP's reward function with additional, artificial rewards [3,14,15,24,38,49,50,53,85]. This article employs shaping functions in the latter sense.

Because the shaping function modifies the rewards the agent receives, the shaped agent may no longer learn an optimal policy for the original MDP (e.g. [57]). Ng et al. [53] show that, in order to retain the optimal policy, a shaping function should consist of a difference of *potential functions* over states. Like a value function, a potential function $\Phi : \mathbf{X} \mapsto \mathbb{R}$ specifies the desirability of a given state. Potential-based shaping functions take the form $F^{\mathbf{x}}_{\mathbf{x}'} = \gamma\Phi(\mathbf{x}') - \Phi(\mathbf{x})$; hence, a positive reward is received when the agent moves from a low to a high potential, and a negative reward when moving in the opposite direction. Similarly to potentials in physics and potential field methods in mobile robot navigation [36], a shaping potential thus results in a "force" encouraging the agent in a certain "direction".

State potential functions can miss out on additional information provided by the actions for a state. To address this, Wiewiora et al. [87,88] introduced shaping functions of the form $F^{\mathbf{xa}}_{\mathbf{x}'a'} = \gamma\Phi(\mathbf{x}', a') - \Phi(\mathbf{x}, \mathbf{a})$, where a' is defined as in the learning rule. In this form, shaping functions closely resemble advice-giving methods [45,81] in that they bias an agent's policy towards the actions that the shaping function estimates as most valuable.[1] Wiewiora et al. show that using F is equivalent to initializing the agent's Q-table to the potential function, under the same experience history. However, some important differences remain. Unlike shaping, initialization biases the agent's actions *before* they are taken. In addition, shaping can be applied to RL with function approximation and in cases where the experimenter does not have access to the agent's value function [88]. Either way, the results in this article apply equally well to shaping as to initialization methods.

2.3 Problem setting

We define a multi-task *domain d* to be a pair $d = \langle D, \mathsf{M} \rangle$, where D is a distribution over tasks $D(m) = \Pr(m)$, and $D(m) > 0$ for all $m \in \mathsf{M}$, the set of all MDPs in the domain. This definition matches the *generalized environment* proposed in [86]. The domain state space is $\mathbf{X}_d = \bigcup_{m \in \mathsf{M}} \mathbf{X}_m$, and similarly $\mathbf{X}^+_d = \bigcup_{m \in \mathsf{M}} \mathbf{X}^+_m$. As noted previously, we assume the same feature and action set for all tasks. However, results from this article can be extended to different feature and action sets through the use of inter-task mappings (see [76] for an overview of these methods).

Given a domain d and one or more source tasks drawn with replacement from d, the agent's goal is to maximize expected online return on an unknown target task sampled from d; this measure also implicitly captures the time to reach a good policy. In general, we are interested in a scenario in which the agent is "interacting sequentially" with the domain, similar to lifelong learning [79]. That is, starting with an empty history h and potential function $\Phi : \mathbf{X}^+_d \to 0$, it goes through the following steps:

[1] The authors termed these *potential-based advice*; specifically, *look-ahead advice* for the formula introduced here. We use the term "shaping" for both methods, and let function arguments resolve any ambiguity.

1. Receive a task m sampled with replacement from the domain according to $D(m)$. The task model is unknown.
2. Learn the solution to m with a model-free learning algorithm, aided by Φ. Add the solution to the solution history h.
3. Update Φ based on h.
4. Go to step 1.

Nonetheless, this article applies equally well to batch scenarios in which the agent receives a sequence of source tasks sampled from the domain upfront; the solutions to this sequence then just become the history h.

3 Potential functions for multi-task learning

In this section, we formulate a definition of an optimal potential-based shaping function and, since we cannot solve this expression exactly, derive three approximations to this function for the learning case. For simplicity, we assume a tabular learning algorithm L. Since we are interested in maximizing online return, an optimal shaping function is one based on a potential function that maximizes expected online return across target tasks:

$$\Phi_L^* = \underset{\Phi}{\operatorname{argmax}}\ \mathrm{E}\left[R_m|\Phi, L\right] = \underset{\Phi}{\operatorname{argmax}} \sum_{m\in\mathsf{M}} D(m)\mathrm{E}\left[\sum_{t=0}^{\infty}\gamma^t r_{t,m}\big|\Phi, L\right], \tag{6}$$

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 is essentially a hidden variable since the potential function does not take it as input. Thus the potential function that satisfies (6) may perform poorly in some tasks, though it performs best in expectation across tasks.

Since shaping with Φ is equivalent to initializing the Q-table with it, solving (6) is equivalent to finding the best cross-task initialization of the Q-table. Unfortunately, because of interacting unknown task and learning dynamics, there is no obvious way to compute such a solution efficiently in the learning case, and search approaches quickly become impractical. However, it is possible to derive a solution for the planning case that provides the lowest bounds on the number of iterations needed to converge.

In the following sections, we first derive an expression for the optimal value table initialization given that the task models are available and solved using value iteration. We show that the optimal initialization in this case minimizes, in expectation, the weighted geometric mean max-norm with the optimal value function in the target task. We then discuss three strategies for efficiently approximating Φ_L^* for the learning case.

3.1 Optimal initialization for value iteration

In the planning case, an optimal initialization is one that minimizes the expected number of iterations to solve the target task.

Theorem 1 *The initial value function Q_0^* that in expectation minimizes the number of iterations needed to solve a given task m from a domain d by value iteration is, for $\gamma \in (0, 1)$,*

$$\begin{aligned} Q_0^* &= \underset{Q_0}{\operatorname{argmax}} \sum_m D(m)\log_\gamma \|Q_m^* - Q_0\|_\infty \\ &= \underset{Q_0}{\operatorname{argmin}}\ \log_\gamma \prod_m \|Q_m^* - Q_0\|_\infty^{D(m)} \end{aligned}$$

Proof By Banach's theorem, the value iteration sequence for a single task converges at a geometric rate [6]:

$$\|Q^*_m - Q^n_m\|_\infty \le \gamma^n \|Q^*_m - Q_0\|_\infty,$$

where Q^n_m is the value function on task m after n iterations and $\|\cdot\|_\infty$ denotes the max-norm. This equation provides a lower bound on the number of iterations n needed to get within an arbitrary distance of Q^*_m, in terms of γ and the initial value function Q_0. That is, to get within ε of Q^*_m, then $\|Q^*_m - Q^n_m\|_\infty \le \varepsilon$, which is satisfied if $\gamma^n \|Q^*_m - Q_0\|_\infty \le \varepsilon$. Let $\delta_m = \|Q^*_m - Q_0\|_\infty$. Assuming $\delta_m > 0$ and $\varepsilon < \delta_m$, then

$$\begin{aligned}\gamma^n \delta_m &\le \varepsilon\\ n &\ge \log_\gamma \frac{\varepsilon}{\delta_m}\\ &= \log_\gamma \varepsilon - \log_\gamma \delta_m.\end{aligned}$$

For multiple tasks the expected lower bound is

$$\begin{aligned}\bar{n} &\ge \sum_m D(m)\left[\log_\gamma \varepsilon - \log_\gamma \delta_m\right]\\ &= \log_\gamma \varepsilon - \sum_m D(m) \log_\gamma \delta_m. \qquad (7)\end{aligned}$$

Thus $\bar{n}$ can be minimized by maximizing $\sum_m D(m) \log_\gamma \delta_m = \sum_m D(m) \log_\gamma \|Q^*_m - Q_0\|_\infty$.

Note that $\sum_m D(m) \log_\gamma \|Q^*_m - Q_0\|_\infty = \log_\gamma \prod_m \|Q^*_m - Q_0\|_\infty^{D(m)}$, which equals the log weighted geometric mean of the max-norm between Q_0 and Q^*_m. Therefore, since $\gamma < 1$, the optimal Q_0 *minimizes* the weighted geometric mean of the max-norm. □

Unfortunately, this expression is in general non-linear and non-convex. However, it seems that good approximations are obtainable by simply minimizing the average distance, according to some norm, between Q^*_m and Q_0. We exploit this intuition in the heuristic approaches to initialization for the *learning* setting that we propose below.

3.2 Initialization for the learning case

Intuitively, a good initialization of the Q-function is the one closest in expectation, according to some norm, to some desired target value function Q_m of the unknown next task m the agent will face. This can be seen as a definition of a *cross-task value* function Q_d that predicts the expected value of a given state–action pair $(\mathbf{x}, a)$ on an unknown new task sampled from the domain, given the target values observed for $(\mathbf{x}, a)$ on previous tasks. If the norm is Euclidean, Q_d gives the least-squared-error prediction of the value of $(\mathbf{x}, a)$ on the new task. That is, it minimizes the mean squared error (MSE) across tasks:

$$Q_d = \underset{Q_0}{\operatorname{argmin}} \left(\sum_{m \in \mathsf{M}} D(m) \sum_{\mathbf{x}, a \in \mathbf{X}^+_m} \Pr(\mathbf{x}, a|m) \Big[Q_m(\mathbf{x}, a) - Q_0(\mathbf{x}, a) \Big]^2 \right), \qquad (8)$$

where $\Pr(\mathbf{x}, a|m)$ is a task-dependent weighting over state–action pairs that determines how much each pair contributes to the error. As is common in least squares, one may want to weight non-uniformly, in our case for example if some state–action pairs occur more often than others. It is not immediately clear how to define $\Pr(\mathbf{x}, a|m)$. In Sect. 3.6, we discuss four possible options for this distribution. For now, we assume $\Pr(\mathbf{x}, a|m)$ is given. By setting the gradient of (8) to zero and solving, we obtain:

$$Q_d(\mathbf{x}, a) = \sum_{m \in \mathsf{M}} \Pr(m|\mathbf{x}, a) Q_m(\mathbf{x}, a). \tag{9}$$

Note that $\Pr(m|\mathbf{x}, a)$ is a natural way of selecting the right tasks to average over for a given pair: it is zero for any $(\mathbf{x}, a) \notin \mathbf{X}_m^+$, leaving such pairs out of the average.

For linear binary function approximators, solving (8) leads to

$$\mathbf{w}_d^i = \sum_{m \in \mathsf{M}} \Pr\left(m|\mathbf{f}^i = 1\right) \mathbf{w}_m^i, \tag{10}$$

where $\mathbf{w}_d^i$ and $\mathbf{w}_m^i$ are the weight of feature i across tasks and in task m respectively, and $\mathbf{f}^i$ is the value of feature i (either zero or one). See Appendix 3 for the derivation. Thus, instead of averaging per state–action pair as in (11), Eq. 10 averages per binary feature. In the following sections, we follow the format of (11) and use table-based value functions, unless mentioned otherwise.

In SL, the targets Q_m are given. In RL, an intuitive choice of target might be the optimal value function of each task. However, there is no guarantee that using a potential function that approximates this target leads to the best online return. In the following three subsections, we propose three different types of Q_m to use as target, each leading to a different potential function.

3.3 Least squared error prediction of Q_m^*

By setting the target to be the optimal value function Q_m^* of each task, we obtain:

$$Q_d^*(\mathbf{x}, a) = \sum_{m \in \mathsf{M}} \Pr(m|\mathbf{x}, a)\, Q_m^*(\mathbf{x}, a). \tag{11}$$

While approaches equivalent to (11) have been used successfully as potential functions or initializations in previous work [68,74], they are not guaranteed to be optimal with respect to the online return. Since Q_d^* makes predictions based on the optimal policy of each task, it may be too optimistic: first, it is likely to overestimate the value of some state–action pairs in some tasks. Second, the actions that Q_d^* estimates as best may not be cautious enough: it assumes the optimal policy will be followed from the next timestep onward, and hence ignores the uncertainty caused by exploration. This may negatively affect online return, in particular in risky tasks, i.e, tasks where some actions may result in a large negative reward, for example a helicopter crash or a fall down a steep cliff. Note that this "risky" ordering of actions may occur even in cases where value is estimated correctly (for example, for states that only occur in one task). We observe this phenomenon experimentally in the cliff domain in Sect. 5.1.

3.4 Least squared error prediction of $\tilde{Q}_m$

In some cases, the agent's Q-function on a task m may never reach Q_m^*, even after learning. This may happen, for example, when using function approximation or an on-policy algorithm with a soft policy. When this occurs, it may be better to use a potential function based 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:

$$\tilde{Q}_d(\mathbf{x}, a) = \sum_{m \in \mathsf{M}} \Pr(m|\mathbf{x}, a)\tilde{Q}_m(\mathbf{x}, a). \tag{12}$$

3.5 Value of the optimal cross-task policy

The previous two approaches to defining cross-task value for the potential function both rely on value function for tasks that have already been solved. Such approaches may be too optimistic, even if (12) is used instead of (11), since they are based on the result of learning and implicitly assume a (near-) optimal policy will be followed from the next time step onward, which is not typically the case during learning.

In this section, we propose another definition which, in a sense, more closely resembles the traditional definition of value in that it estimates value of a *single* fixed cross-task policy $\mu : \mathbf{X}_d \times \mathsf{A} \to [0, 1]$ s.t. $\sum_a \mu(\mathbf{x}, a) = 1$ that assigns the same probability to a given state–action pair, regardless of the current task. This definition might also be more suitable for use as Φ since, like Φ, it is fixed across tasks. The value function of the best possible cross-task policy, μ^*, will typically make lower estimates than either Q^*_d or $\tilde{Q}_d$, since μ^* is usually not optimal in every (or any) task. For example, consider a domain with a goal location in an otherwise empty room. If the distribution over goal locations is uniform, and the state provides no clue as to the goal position, then μ^* is a uniform distribution over actions in every state. We define the cross-task value of a state under a stationary policy μ as

$$Q^\mu_d(\mathbf{x}, a) = \sum_{m \in \mathsf{M}} \Pr(m|\mathbf{x}, a) Q^\mu_m(\mathbf{x}, a). \tag{13}$$

Like (11) and (12), this follows (11), except that it averages over the values of a *single* policy instead of multiple task-dependent ones.

The fact that μ is task-independent makes the task essentially a hidden variable and MTRL similar to a partially observable MDP (POMDP), for which μ is a *memoryless policy* that conditions only on the current observation and Q^μ_d is similar to the value of such a policy as defined in [66]. Therefore, there need not exist a stationary policy (stochastic or deterministic) that maximizes the value of each state simultaneously [66]. Consequently, traditional dynamic programming methods may not apply. One way to overcome this problem is to assign a scalar value to each possible policy:

$$V^\mu_d = \sum_{\mathbf{x} \in \mathbf{X}_d} \Pr(\mathbf{x}) V^\mu_d(\mathbf{x}) \tag{14}$$

$$V^\mu_d(\mathbf{x}) = \sum_{a \in \mathsf{A}} \mu(\mathbf{x}, a) Q^\mu_d(\mathbf{x}, a) \tag{15}$$

where V^μ_d is the domain-wide value of μ and $\Pr(\mathbf{x})$ is a distribution that assigns an appropriate measure of weight to each state $\mathbf{x}$. Two options for $\Pr(\mathbf{x})$ are the start-state distribution or the occupation probability of $\mathbf{x}$, $\Pr(\mathbf{x}|\mu)$. For the POMDP case, Singh et al. [66] show that defining the optimal policy as

$$\mu^* = \underset{\mu}{\operatorname{argmax}} \sum_{\mathbf{x}} \Pr(\mathbf{x}|\mu) V^\mu_d(\mathbf{x}) \tag{16}$$

is equivalent to maximizing the average payoff per time step.

3.6 Choosing $\Pr(\mathbf{x}, a|m)$

All three proposed potential function types take the general form

$$Q_d(\mathbf{x}, a) = \sum_{m \in \mathsf{M}} \Pr(m|\mathbf{x}, a) Q_m(\mathbf{x}, a),$$

where $\Pr(m|\mathbf{x}, a) = \Pr(\mathbf{x}, a|m) D(m) / \Pr(\mathbf{x}, a)$. As indicated in Sect. 3.2, it is not immediately clear how to define $\Pr(\mathbf{x}, a|m)$. Four options are:

1. *The stationary distribution induced by the policy corresponding to each Q_m* For Q_d^* and Q_d^μ, this would be π_m^* and μ^*, respectively. For $\tilde{Q}_d$ it would be the soft policy (e.g. ε-greedy). Since Q_d averages over all Q_m, this seems a good option. However, it may be problematic for Q_d^* and $\tilde{Q}_m$ as it represents only the distribution over $(\mathbf{x}, a)$ *after* learning. There may be state–action pairs that the policy never, or rarely, visits in some tasks, but clearly the values in these tasks should still be included in the average.
2. *The distribution induced by the learning process* Since we are interested in improving performance during learning, another choice may be a distribution over $(\mathbf{x}, a)$ *during* learning. However, it is unclear how to define this distribution, since it depends, among other things, on when learning is halted and the trajectory through state–action space taken during learning. One possibility is to take one sample from all possible such trajectories and define
$$\Pr(\mathbf{x}, a|m) = \sum_{t=1}^{T} \Pr(\mathbf{x}, a|m, \pi_t) \Pr(\pi_t|m),$$
where $\Pr(\pi_t|m) = 1/T$ for the soft policy the agent was following at time step t when learning task m, $\Pr(\mathbf{x}, a|m, \pi_t)$ is the stationary distribution over $(\mathbf{x}, a)$ given π_t, and T is a predetermined stopping criterion, such as a fixed number of learning steps or a convergence threshold. Clearly, this option does not apply to Q_d^μ.
3. *The start-state distribution and uniform random policy* This defines
$$\Pr(\mathbf{x}, a|m) = \Pr(\mathbf{x}_0 = \mathbf{x}|m) \Pr(a),$$
where $\Pr(\mathbf{x}_0 = \mathbf{x}|m)$ is the start-state distribution of m and $\Pr(a) = 1/|\mathsf{A}|$ for all a. This definition appropriately captures the distribution over $(\mathbf{x}, a)$ when the agent enters a new task without prior knowledge.
4. *The uniform distribution* This defines $\Pr(\mathbf{x}, a|m) = 1/|\mathbf{X}_m^+|$ for all $(\mathbf{x}, a) \in \mathbf{X}_m^+$.

For Q_d^* and $\tilde{Q}_d$, the latter option seems the most sensible one. Under the first and third definitions, a probability of zero might be assigned to a state–action pair in some tasks, or even in all, which is clearly not desirable. Also, since (11) and (12) are concerned with prediction of the optimal (approximate) value of a pair $(\mathbf{x}, a)$ in a new target task, the underlying assumption is that *after* taking a in $\mathbf{x}$, the optimal (soft) policy for that task will be followed. However, the first two definitions in the list above make assumptions about the policy that has been followed *so far*, which is irrelevant in this context. Empirical comparison of the options revealed no significant difference for Q_d^* and $\tilde{Q}_d$. For Q_d^μ, the stationary distribution induced by μ was found to work best in most cases.

4 Representation selection

So far, the discussion has focused on different potential functions, without regard to the knowledge the agent has of the domain; in fact, the formulas in the previous section are based on the set of all tasks in the domain, thereby implicitly assuming full domain knowledge. This discussion is important for selecting the right potential function; as Sect. 5.1 will show, the best-performing potential function may depend on the domain and learning parameters, even with full domain knowledge.

In practice, the agent's knowledge of the domain is limited by the number of tasks it has observed. Therefore, we now turn to the setting in which only a sample sequence of tasks is available to the agent. A central aim in this setting is to *generalize* well from the sample to new data. As in SL, representation is key to generalization. This section discusses the theory underlying the second key step in maximizing online return through potential functions for MTRL: finding the representation that approximates this target as closely as possible while at the same time generalizing well given limited domain information (observed tasks). To this end, we propose a definition of relevance that can be used for learning representations in MTRL. This definition applies to any potential function type; in fact, it applies to any table-based function that maps vectors to scalars.

In MTRL, new data may consist of both new state–action pairs and new tasks. Our central objective, therefore, is to discover which information is relevant across tasks, which we call *domain-relevant* information. While our definition also captures which information is relevant *within* tasks (task relevant), this subject has been extensively addressed in existing research. Therefore, we focus primarily on discovering domain-relevant representations.

The robot navigation example in the introduction illustrates the distinction between task and domain relevance. Recall that here, the robot needs to locate a bomb at a fixed task-dependent location in buildings with different layouts, where each task is a different building. Now imagine that in addition, a building can be green or red; in red buildings, the robot receives negative reward when standing still (e.g. because it is in enemy territory and needs to keep moving). To describe an optimal policy for each task, only position, which constitute a Markov state representation, is needed to represent state. However, the value of a given position needs to be re-learned in every task. Since position does not represent information that can be retained between tasks, this feature is task relevant. Building color is not useful within a given task, since its value is constant. However, across tasks it represents information about standing still, which receives additional punishment in red buildings. Therefore, this feature is domain relevant, but not task relevant. Finally, distance sensors are useful both within and across tasks, and are therefore both task and domain relevant.

Projecting the full feature set onto a subset of the task-relevant features may create an abstract state space that is smaller than the original state space, and can therefore help learn the task more quickly, under certain conditions on the projection [43]. In the multi-task setting, it is possible to define this abstract space *before* entering a new task, by identifying valid abstractions (task-relevant representations) of previously experienced tasks and transferring these to the new task (e.g. [19,37,40,83]).

Domain-relevant features also allow for a higher level of abstraction, but in addition allow the agent to *deduce rules from the abstract representation and reason with them, right from the start of a new task*, thereby obviating the need to re-learn this task-invariant knowledge. Of course, it should be possible for the agent to override the heuristic rules given new information garnered from interaction with a specific task. Shaping functions are a good candidate for these kind of rules, endowing the agent with prior knowledge that gradually degrades as the agent accumulates experience of a task.

Li et al. [43] provide an overview and classification of several state abstraction schemes in reinforcement learning. We employ their definition of a state abstraction function:

Definition 1 (*State abstraction function*) A *state abstraction function* $\phi : \mathbf{X} \mapsto \mathbf{Y}$ induces a partition on $\mathbf{X}$; $\phi(\mathbf{x}) \in \mathbf{Y}$ is the abstract state $\mathbf{y}$ corresponding to $\mathbf{x}$, and the inverse image $\phi^{-1}(\mathbf{y})$ is the set of ground states that corresponds to $\mathbf{y}$ under abstraction function ϕ.

In the following subsections, we propose a definition of relevance that is based on a state representation's ability to predict expected return within tasks (for task-relevant features) or across tasks (for domain-relevant features). We argue that task and domain relevance are special cases of a single underlying concept, k-relevance, the expected relevance on a sequence of k tasks sampled from the domain. We show formally that k-relevance converges to a fixed point in the limit, and under certain assumptions does so monotonically. We use this property to introduce FS-TEK, a novel feature selection algorithm. The key insight behind FS-TEK is that change in relevance observed on task sequences of increasing length can be extrapolated to more accurately predict domain relevance.

4.1 Relevance

Since our main goal is to find good representations for potential functions, our targets for abstraction are Q_d^*, $\tilde{Q}_d$, or Q_d^{μ}, the three types of potential function proposed in Sect. 3. Which of these is used does not matter for the theory, as any Q-function can serve as a basis for relevance. Therefore, in what follows, we remain agnostic to the target for relevance.

Since the agent has only experienced a sample sequence of tasks from the domain, it cannot compute this target Q_d exactly. Instead, it can approximate Q_d by computing a cross-task value function based on the sequence. Let $c = (c_1, c_2, \ldots, c_k)$ be a sequence of $|c|$ tasks sampled from M, and $\mathbf{X}_c = \bigcup_{c_i \in c} \mathbf{X}_{c_i}$.

In the following, we treat the action as just another state feature so that it can be abstracted away if it is irrelevant. Let $\mathbf{x}^+ \in \mathbf{X}_c^+$ denote a state–action pair. Let Q_c be a cross-task Q-function computed on a sequence of tasks c, $Q_c : \mathbf{X}_c \times \mathsf{A} \mapsto \mathbb{R}$. If $|c| = 1$, $Q_c = Q_m$. All of the definitions for Q_d in Sect. 3 follow the general form

$$Q_d\left(\mathbf{x}^+\right) = \sum_{m \in \mathsf{M}} \Pr\left(m|\mathbf{x}^+\right) Q_m\left(\mathbf{x}^+\right).$$

Thus, we have for Q_c:

$$Q_c\left(\mathbf{x}^+\right) = \sum_{i=1}^{|c|} \Pr\left(c_i|\mathbf{x}^+, c\right) Q_{c_i}\left(\mathbf{x}^+\right), \tag{17}$$

where

$$\Pr\left(c_i|\mathbf{x}^+, c\right) = \frac{\Pr\left(\mathbf{x}^+|c_i\right)\Pr(c_i|c)}{\Pr\left(\mathbf{x}^+|c\right)},$$

where $\Pr(c_i|c) = 1/|c|$ for all c_i, since the probability of sampling a task from the domain $D(m)$, is already reflected in $\Pr(c)$. Furthermore, given a sample of tasks, the best we can do given the prior of the sample is to assign each task a probability corresponding to its frequency in the sample, which is accomplished by $1/|c|$.

Several notions of predictive power on Q_c are possible. For example, the Kullback–Leibler divergence between the marginal distribution over Q_c and the distribution conditioned on the representation of which we wish to measure relevance [68], or the related measures of conditional mutual information [29] or correlation [47]. The disadvantage of these measures is that they do not take the magnitude of the impact of a representation into account; for example, two representations may cause equal divergence, but the difference in expected return associated with one set may be much larger than the other. To address this problem, we propose a measure of relevance[2] that is proportional to the squared error in predicting Q_c.

Let $\phi(\mathbf{X}_c^+) = \mathbf{Y}_c$ denote all possible projections of $\mathbf{X}_c^+$ onto Y using ϕ. For ease of notation, denote the set of ground state–action pairs belonging to an abstract state–action pair $\mathbf{y} \in \mathbf{Y}_c$, $\phi^{-1}(\mathbf{y})$, by $\mathbf{X}_c^{\mathbf{y}}$. Defining the Q-function of an abstract pair as the weighted average of the values of its corresponding ground pairs is a natural definition.

Definition 2 (*Abstract Q-function*) Given abstraction $\phi : \mathbf{X}_c^+ \rightarrow \mathbf{Y}_c$ and any Q-function $Q_c : \mathbf{X}_c^+ \rightarrow \mathbb{R}$, the abstract Q-function $\bar{Q}_c(\phi) : \mathbf{Y}_c \rightarrow \mathbb{R}$ is defined as the weighted average of the Q-values of ground pairs $\mathbf{x}^+$ corresponding to abstract pair $\mathbf{y}$:

$$\bar{Q}_c(\phi, \mathbf{y}) = \sum_{\mathbf{x}^+ \in \mathbf{X}_c^{\mathbf{y}}} \Pr\left(\mathbf{x}^+ | \mathbf{y}\right) Q_c\left(\mathbf{x}^+\right). \tag{18}$$

It follows that the Q-value of the null abstract state, denoted $\bar{Q}_c^{\emptyset}$, which corresponds to the empty set of features, is the mean of all Q-values.

The Q-value of a given abstract state–action pair $\mathbf{y}$ generally differs from those of at least some ground state–action pairs corresponding to $\mathbf{y}$. The error ε for a given abstract state–action pair is the weighted mean squared error of ground pairs with respect to $\mathbf{y}$.

Definition 3 (*Abstraction error $\varepsilon_\phi(\mathbf{y}, Q_c)$*) Given abstraction $\phi : \mathbf{X}_c^+ \rightarrow \mathbf{Y}_c$ and any Q-function $Q_c : \mathbf{X}_c^+ \rightarrow \mathbb{R}$, the error of a given abstract state–action pair $\mathbf{y}$ with respect to its corresponding ground pairs is given by

$$\varepsilon_\phi(\mathbf{y}, Q_c) = \sum_{\mathbf{x}^+ \in \mathbf{X}_c^{\mathbf{y}}} \Pr\left(\mathbf{x}^+ | \mathbf{y}\right) \left[Q_c\left(\mathbf{x}^+\right) - \bar{Q}_c(\phi, \mathbf{y})\right]^2. \tag{19}$$

It follows naturally that the relevance of an abstraction with respect to a function Q_c is the total error incurred by applying the abstraction to Q_c.

Definition 4 (*Relevance*) Given abstraction $\phi : \mathbf{X}_c^+ \rightarrow \mathbf{Y}_c$ and any Q-function $Q_c : \mathbf{X}_c^+ \rightarrow \mathbb{R}$, the relevance $\rho(\phi, Q_c)$ of ϕ with respect to Q_c is defined as the weighted mean squared error on Q_c that results from applying ϕ:

$$\rho(\phi, Q_c) = \sum_{\mathbf{y} \in \mathbf{Y}_c} \Pr(\mathbf{y}|c) \varepsilon_\phi(\mathbf{y}, Q_c). \tag{20}$$

When using an abstract Q-function as defined in (18), this equals the sum of weighted variances of the Q-values of ground state–action pairs corresponding to a given $\mathbf{y}$:

$$\rho(\phi, Q_c) = \sum_{\mathbf{y} \in \mathbf{Y}_c} \Pr(\mathbf{y}|c)\, \mathrm{Var}\left(Q_c(\mathbf{X}_c^{\mathbf{y}})\right). \tag{21}$$

Note that the aim is therefore to find an abstraction with *low* relevance: this abstracts away non-relevant parts of the original representation and keeps the relevant parts. This is in line with the terminology employed by Li et al. [43], who designate the abstraction that preserves Q^* as a Q^*-irrelevance abstraction. Similarly, an abstraction with low relevance on Q_c should be thought of as a Q_c-irrelevance abstraction.

[2] Relevance is not a measure in the strict mathematical sense; because of dependence between feature sets, $\rho(\mathsf{F} \cup \mathsf{G}) \neq \rho(\mathsf{F}) + \rho(\mathsf{G})$ for some disjoint feature sets F and G and relevance ρ.

4.2 k-Relevance

The previous section defined the relevance of an abstraction with respect to a Q-function computed on a sequence of tasks. This single definition captures both task and domain relevance; the former just requires a one-task sequence. Naturally, relevance often depends on both the length of the task sequence over which it is computed and on the tasks that the sequence contains. For example, in the robot navigation domain as a whole, the position feature is irrelevant since, given a certain position, there is no way to know what direction to go. In other words, in Q_d, the average value function computed over the whole domain, Q-values will not vary with the position feature, and therefore the position feature will not result in any error on Q_d if it is discarded.

However, given a sample sequence of two tasks, it is likely that position is still relevant, e.g., if the bomb is in the same general area of the building in both tasks. In practice, the agent, while interacting with the domain, will have at its disposal a growing sequence of tasks based on which it must construct a domain-relevant representation. Doing so is challenging because representations may appear relevant given the observed sequence but actually not be domain irrelevant. Intuitively, the longer the sequence, the better sequence relevance approximates domain relevance. Therefore, instead of just computing relevance on the currently observed sequence, we should try to predict how it *changes* with increasing sequence length. Decreasing relevance could mean that the feature, while relevant on the current sequence, is not relevant in the long run. However, it is not enough to simply observe that the relevance of a given representation is decreasing because it may plateau above zero.

To predict how relevance changes as more tasks are observed, we use the definitions in the previous section to define the *k-relevance* of a representation as the expected relevance ρ_k over all possible sequences of length k. Let C_k be the set of all possible sequences c of length k, i.e., $|\mathsf{C}_k| = |\mathsf{M}|^k$. The probability of sampling a given sequence c from C_k is $\Pr(c) = \prod_{i=1}^{|c|} D(c_i)$. Then we have the following definition for k-relevance:

Definition 5 (*k-relevance*) The k-relevance of an abstraction $\phi : \mathbf{X}_d^+ \to \mathbf{Y}_d$ in a domain $d = \langle D, \mathsf{M} \rangle$ is the expected relevance of ϕ on a given Q-function computed from a task sequence of length k sampled from M according to D

$$\rho_k(\phi) = \mathrm{E}\left[\rho(\phi, Q_c)|c \in \mathsf{C}_k\right] = \sum_{c \in \mathsf{C}_k} \Pr(c)\rho(\phi, Q_c). \tag{22}$$

Given this definition, an abstraction ϕ is strictly domain relevant, DR(ϕ), if and only if its k-relevance ρ_k remains positive as k goes to infinity:

$$\mathrm{DR}(\phi) \Leftrightarrow \lim_{k \to \infty} \rho_k(\phi) > 0, \tag{23}$$

and strictly domain irrelevant otherwise:

$$\neg\mathrm{DR}(\phi) \Leftrightarrow \lim_{k \to \infty} \rho_k(\phi) = 0. \tag{24}$$

Our goal is thus to find a domain-irrelevant abstraction, as applying such an abstraction yields a domain-relevant representation. In Sect. 4.3, we show that ρ_k converges to the true domain relevance as k goes to infinity.

As k tends to infinity, the average over all sample sequences of length k will approach the true distribution over tasks in the domain ever more closely. However, the value of k at which k-relevance is a reasonably accurate approximation of the true domain relevance depends largely on the number of tasks in the domain.

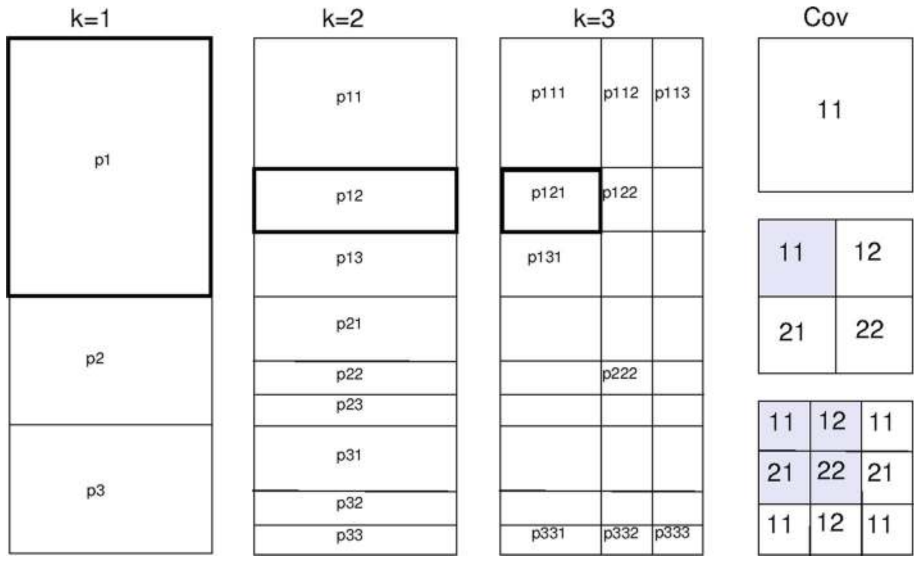


Fig. 1 *Left three columns* ($k = 1 \dots 3$): probability of each sequence as k increases, shown as portions of a *rectangle* with area 1, on a 3-task domain with $D(1) = 0.5$, $D(2) = D(3) = 0.25$. Probability of sequence $(1, 2)$ is indicated as $p12$. For each $k + 1$, the area of each sequence for k is split into new sequences with area proportional to D. *Right column* covariance matrices for the sequences marked in *bold on the left*. The weight of each matrix element is equal, namely $1/k^2$. Relevance on a given sequence is the sum of all matrix elements. In turn, k-relevance is the sum over all sequences

Similarly, an abstraction may be called task relevant, $\mathrm{TR}(\phi)$, if and only if its one-relevance is greater than zero:

$$\mathrm{TR}(\phi) \Leftrightarrow \rho_1(\phi) > 0. \tag{25}$$

Thus, a sensible abstraction to use in the domain for the value function or policy would be one that is (near-)irrelevant according to this definition and thus the expected error on the value function of a task introduced by the abstraction is negligible. We believe this is a more natural definition than, e.g., preserving any feature that has been found relevant in any task in the domain [83].

Thus, k-relevance is a unifying notion of relevance that naturally captures both task and domain relevance and can therefore be used for finding abstractions for both the value function of a new task and the potential function to use on that task. In the next section, we show that k-relevance converges to a fixed point as k tends to infinity, a result which makes it possible to extrapolate domain relevance from observed tasks, as we show in Sect. 4.4.

4.3 Properties of k-relevance

Figure 1 illustrates how k-relevance changes with k. Relevance of a given ϕ is the same on each single-task sequence for a given task m, i.e. sequences for which each element $c_i = m$. For $k = 1$, k-relevance comprises only single-task sequences, but the share of these sequences decreases exponentially with k; in the example, it is $D(1)^k + D(2)^k + D(3)^k$. In general, the sequences represent the true distribution over tasks ever more closely.

For $k = 2$, every task is combined with every task in the domain. Hence for $k > 2$, no new task combinations arise; however, we need a way to quantify relevance on, e.g.,

$c = (1, 2, 3)$ given the relevance on $(1, 2)$, $(1, 3)$, and $(2, 3)$. We show in Appendix 2 that it follows, from the expression of relevance in terms of variance (Eq. 21) that relevance on any sequence equals the sum of covariances between the Q-functions of all task pairs involved in the sequence. The right column of Fig. 1 visualizes this; the highlighted areas correspond to the covariance matrix (and hence relevance) for $k-1$; therefore, $k+1$-relevance is a function of k-relevance. The following two theorems follow directly from this, together with the fact that sequences approximate the true distribution over tasks ever more closely.

Theorem 2 *Let ϕ be an abstraction with abstract Q-function as in Definition 2, and let $\rho_k = \rho_k(\phi)$ for any k. Let $d(x, y) = |x - y|$ be a metric on $\mathbb{R}$, and let $f(\rho_k) = \rho_{k+1}$ map k-relevance to $k+1$-relevance. Then f is a contraction; that is, for $k > 1$ there is a constant $\kappa \in (0, 1]$ such that*

$$d\left(f(\rho_k),\ f(\rho_{k-1})\right) \leq \kappa d\left(\rho_k, \rho_{k-1}\right).$$

Furthermore, if $d(\rho_2, \rho_1) \neq 0$, then f is a strict contraction, i.e. there is a $\kappa \in (0, 1)$ such that the above holds.

Proof See Appendix 2.

Corollary 1 *The sequence $(\rho_k)_{k=1}^{\infty}$, $k = 1, 2, \ldots$, converges to a fixed point, namely the domain relevance of ϕ.*

Proof Since $(\mathbb{R}, d)$ is a complete metric space and f is a contraction, f has a unique fixed point by Banach's fixed-point theorem. □

Theorem 3 *If all tasks share the same distribution over state–action pairs* $\Pr(\mathbf{x}^{+}|m)$, *then ρ_k is a monotonically increasing or decreasing function of k, or is constant.*

Proof See Appendix 2.

In other words, ρ_k never changes direction. Note that having the exact same distribution over state–action pairs also implies having the exact same state–action space. By these two theorems, ρ_k can be approximated by an exponential or power law function. Even when tasks do not share the same distribution over state–action pairs, an exponential function can still approximate where the sequence $(\rho_k)_{k=1}^{\infty}$ will converge to, given sufficient k. In addition, our experiments suggest that ρ_k is also monotonic in these cases.

4.4 Feature selection with k-relevance

We now introduce FS-TEK, a novel algorithm for finding relevant abstractions in MTRL that exploits the notion of k-relevance introduced in the previous sections. FS-TEK focuses on finding domain-relevant representations, since determining task-relevant representations using k-relevance is relatively straightforward. As mentioned previously, the key idea behind FS-TEK is to fit an exponential function to a candidate representation's relevance based on an observed sequence of tasks and then extrapolate it to estimate the representation's domain relevance.

Since it is a feature selection algorithm, FS-TEK constructs abstractions of the form $\phi(\mathbf{x}) = \mathbf{x}[\mathsf{Y}] = \mathbf{y}$, where $\mathsf{Y} \subseteq \mathsf{X}$ and $\mathbf{x}[\mathsf{Y}]$ denotes the values of the features Y in vector $\mathbf{x}$. That is, the abstract state $\mathbf{y}$ to which a state $\mathbf{x}$ is mapped consists of the values of the features in some relevant set Y. Viewed another way, Y is the result of removing some irrelevant set

Algorithm 1 FS-TEK

```
Input: a sequence $s$ of task solutions $Q_{s_i}, i \in \{1, 2, \ldots, |s|\}$; $\alpha(k)$, the confidence level
Output: A set $\mathsf{R}$ of features to remove
1:
2: $\mathsf{R} \leftarrow \emptyset$
3: $f(k;\theta) = \theta_1 + \theta_2 \exp(-\theta_3 k)$
4: repeat
5:    $\mathsf{F} \leftarrow \emptyset$      //Features marked for removal in this iteration
6:    Calculate $D$, the $|s| \times |\mathsf{X}^+ - \mathsf{R}|$ matrix of $k$-relevance per feature, using BKR
7:
8:    // Each feature for which extrapolated function of relevance does not differ
9:    // significantly from 0 is marked for removal
10:   for all $X_i \in \mathsf{X}^+ - \mathsf{R}$ do
11:      $\hat{\theta} \approx \underset{\hat{\theta}}{\text{argmin}} \sum_{k=1}^{|s|} \left[ f(k;\hat{\theta}) - D(k,i) \right]^2$      //least-squares fit
12:      $a = \min\{a : \frac{df(k;\hat{\theta})}{dk}(a) = 0\}$
13:      if $f(a;\hat{\theta}) - \text{CI}(a, \alpha(|s|)) \leq 0$ then
14:         $\mathsf{F} \leftarrow \mathsf{F} \cup X_i$
15:      end if
16:   end for
17:
18:   // Of all marked features, the one with weakest forward relevance is removed
19:   for all $X_i \in \mathsf{F}$ do
20:      $r(i) = \text{FR}(s, \mathsf{R} \cup X_i)$
21:   end for
22:   $\mathsf{R} \leftarrow \mathsf{R} \cup \text{argmin}_i\, r(i)$
23: until $\mathsf{F} = \emptyset$
```

$\mathsf{F} = \mathsf{X} - \mathsf{Y}$ from X. In the following, when we refer the relevance of a set F, we mean the relevance of the abstraction that removes F from X.

The abstraction function $\phi(\cdot)$ is fundamentally different for binary linear function approximator representations than table-based representations; in the former case, the function needs to identify and cluster abstract states directly in feature space. However, FS-TEK itself operates similarly for both represention s.

FS-TEK's main loop is based on an iterative *backward elimination* procedure (e.g.[28,32]). Because of interdependence between features, it is not always sufficient to select features based on their relevance in isolation. Features may be irrelevant on their own, but relevant together with another feature; similarly, a feature that seems irrelevant may become relevant once another feature is removed. Backward elimination starts with the full feature set and iteratively removes features according to a measure of relevance (k-relevance, for FS-TEK). In contrast, *forward selection* methods start with the empty set and iteratively add features. While forward selection may yield a smaller final feature set, it may miss interdependencies between features (e.g. [28]). Therefore, FS-TEK's main procedure uses backward elimination. Nonetheless, it attempts to combine the advantages of both methods by using forward selection to decide which feature to remove when more than one feature is marked for elimination.

Algorithm 1 specifies the main body of FS-TEK. It takes as input the sequence s of solutions to observed tasks and a parameter $\alpha(k)$ that specifies the confidence level for the statistical test on relevance, possibly depending on sample sequence length k.

In each iteration, FS-TEK starts by computing the k-relevance of each feature united with the current set of features to be removed, with k ranging from one to the current number of observed tasks (line 6; the details of BKR, backward-k-relevance, are provided in Algorithm 2 below). For each feature, this results in a dataset with k as input and relevance as target (each column of the matrix D). The algorithm subsequently does a nonlinear least-squares fit of the exponential function $f(k;\theta)$ to each feature's relevance data (line 11).

Next, the function value and confidence interval are computed for the point where f asymptotes (lines 12–15; in line 13, $\mathrm{CI}(a, \alpha(|s|)$ is the confidence interval at point a for the length of the current observed sequence of tasks). If the confidence interval's lower bound is less than or equal to zero, the feature is classified as domain irrelevant and added to the set of features to be removed (line 20).

This procedure often marks more than one feature for removal. FS-TEK uses forward selection to decide which of the marked features to remove (lines 19–21). In this check, the relevance of the set of features to remove is tested in isolation, without taking into account the features that remain. Contrary to BKR, which computes relevance for a range of k, forward relevance (FR) computes relevance only on the currently observed sequence of tasks. Using forward selection can enable FS-TEK to select a smaller subset of relevant features and provides a "second opinion" to supplement the noisy estimate of BKR.

The forward and backward relevance methods operate essentially according to similar principles. Algorithm 2 describes BKR, which computes, for each feature not yet marked for removal, the k-relevance for the given k. Given a set of task sequences C_k, the average Q-function for each sequence is computed according to Eq. 17 (lines 3–5). Each feature's k-relevance is then the average relevance taken over those Q-functions, as defined in (22) (lines 8–12). Relevance is computed for the abstraction ϕ_i that removes from the full feature set a feature set consisting of the features removed in previous iterations united with the given feature.

Algorithm 2 BKR: Backward-k-Relevance

```
Input: a sequence $s$ of task solutions $Q_{s_i}$, $i \in \{1, 2, \ldots, |s|\}$; $\mathsf{R}$ the set of features currently marked for
    removal (possibly empty); $k$
Output: $\rho_{\mathbf{k}}$, a row vector with the $k$-relevance of each $X_i \in \mathsf{X}^+ - \mathsf{R}$
1: $\mathsf{C}_k \leftarrow N$ sample combinations from the $\binom{|s|}{k}$ possible combinations of length $k$ from $s$
2: // Compute the average Q-function $Q_c$ for each sequence
3: for all $c \in \mathsf{C}_k$ do
4:     $Q_c(\mathbf{X}^+) = \sum_{i=1}^{|c|} \Pr(c_i|\mathbf{X}^+, c) Q_{c_i}(\mathbf{X}^+)$        // Equation 17
5: end for
6:
7: // Compute $k$-relevance of each feature
8: for all $X_i \in \mathsf{X}^+ - \mathsf{R}$ do
9:     $\mathsf{Y} = \mathsf{X}^+ - (\mathsf{R} \cup X_i)$
10:    $\phi_i : \mathbf{X} \to \mathbf{X}[\mathsf{Y}]$
11:    $\rho_k(\phi_i) = \frac{1}{N} \sum_{c \in \mathsf{C}_k} \rho(\phi_i, Q_c)$        // Equation 22
12: end for
```

FR (Algorithm 3) is similar but only computes relevance based on the current task sequence. In addition, it adds features to the empty set instead of removing them from the full set. It first computes an abstract function based on the set G of features to be removed (lines 2–4). Forward relevance is then the relevance of the null abstraction (the empty set of features) with respect to the Q-function based on G: i.e., the difference between the error made by using G and that made by using no features.

Not taking into account the complexity of the nonlinear least-squares optimization procedure, FS-TEK's complexity is mainly determined by BKR. Let the number of features $|\mathsf{X}^+| = P$, the size of the state–action space $|\mathbf{X}_s^+| = N$, and the maximum number of sequences

Algorithm 3 FR: Forward Relevance

```
Input: a sequence $s$ of task solutions $Q_{s_i}$, $i \in \{1, 2, \ldots, |s|\}$; the set of features to be tested $\mathsf{G}$
1:
2: $Q_s(\mathbf{X}^+) = \sum_{i=1}^{|s|} \Pr(s_i|\mathbf{X}^+, s) Q_{s_i}(\mathbf{X}^+)$    // Equation 17
3: $\phi : \mathbf{X}_s^+ \rightarrow \mathbf{Y}_s$, where $\mathbf{Y}_s = \mathbf{X}_s^+[\mathsf{G}]$
4: $\bar{Q}_s(\mathbf{y}) = \sum_{\mathbf{x}^+ \in \mathbf{X}_s^{\mathbf{y}}} \Pr(\mathbf{x}^+|\mathbf{y}) Q_s(\mathbf{x}^+)$    // Abstract function based on $\mathsf{G}$, according to (18)
5: $\phi_\emptyset : \mathbf{X}_s^+ \rightarrow \emptyset$
6: return $\rho(\phi_\emptyset, \bar{Q}_s)$
```

$max\_seqs = M$. BKR's worst-case time complexity is $O(MkN + (P - |\mathsf{R}|)(N + MN))$. However, in practice FS-TEK does not re-sample the sequences at each iteration and computes the average Q-functions based on the sequences only once at the start of the algorithm, so the MkN term can be removed. This results in a complexity of $O(N(P - |\mathsf{R}|)(1 + M))$ for BKR when used by FS-TEK.

In the worst case, all features in X^+ need to be removed. FS-TEK's first iteration incurs a cost of $O(|s|NP(1 + M))$, since R is empty. On the second iteration, $|\mathsf{R}| = 1$, and thus the cost is $O(|s|N(P - 1)(1 + M))$. Since a total of P iterations is needed, the cost in terms of P progresses as $P + P - 1 + P - 2 + \cdots + 1 = P(P + 1)/2$. Therefore, the total cost is $O(|s|N(P^2 + P)(1 + M))$, i.e., quadratic in the number of features. Also, while the cost is linear in the size of the state–action space N, in the worst case N scales exponentially with the number of features P. In practice, however, usually not all features need to be removed, and features frequently covary such that N does not scale exponentially with P.

5 Empirical evaluations

In the previous two sections, we proposed solutions for the two key steps involved in maximizing online return in MTRL through potential functions: selection of a good potential function, and finding a good representation for this function in order to generalize well given limited domain knowledge. For the first step, we proposed three different types of potential functions. For the second step, we proposed the novel feature selection algorithm, FS-TEK, which extrapolates change in relevance to predict true domain relevance. In this section, we evaluate these contributions empirically on multiple domains.

5.1 Evaluating potential functions

We begin by empirically comparing the three potential functions proposed in Sect. 3 to a baseline agent that does not use shaping, and introduce the four domains used for all experiments. While most of these domains are simple, they enable the illustration of critical factors in the performance of shaping functions in MTRL.

For comparison purposes, in this section we assume the agent has perfect knowledge of each domain and thus compute each potential function using all tasks in the domain. Our goal is to demonstrate the *theoretical* advantages of each type of potential function. Assuming perfect domain knowledge enables comparisons of the potential functions' maximum performance, untainted by sampling error. In Sect. 5.2, we consider the more realistic setting in which only a sample sequence of tasks is available, and in which generalization to unseen next tasks, and thus learning a good representation for the potential function, is important.

5.1.1 Episodic cliff domain

To illustrate a scenario in which Q^*_d, 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 [73]. The agent starts in one corner of the grid and needs to reach a goal location in the corner that is in the same row or column as the start state, while avoiding stepping into the cliff that lies in between the start and goal.

The domain contains all permutations with the goal and start state in opposite corners of the same row or column with a cliff between them (eight tasks in total). Each task is a 4×4 grid world with deterministic actions N, E, S, W, states (x, y), and a -1 step penalty. Falling down the cliff results in $-1,000$ reward and teleportation to the start state. The distribution over tasks is uniform. We compute each potential function according to the definitions given in Sect. 3. Finding the cross-task policy μ^* has been shown to be NP-hard for POMDPs [82]. We use a genetic algorithm (GA) to approximate it.[3]

To illustrate how performance of a given Φ depends on the learning algorithm, we use two standard RL algorithms, Sarsa and Q-Learning. Since for Q-Learning, $Q^*_d = \tilde{Q}_d$, we use Sarsa's solution for $\tilde{Q}_d$ for both algorithms. Both algorithms use an ε-greedy policy with $\varepsilon = 0.1$, $\gamma = 1$, 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, since the information it provides is already deducible from the agent's position and the number of states per task is not affected. However, the number of states in the domain does increase: one result of adding the sensor is that tasks no longer have identical state spaces.[4]

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. Table 1a shows performance without the cliff sensor. On this domain, Q^μ_d performs very poorly; one reason may be that the GA did not find μ^*, but a more likely one is that, due to the structure of the domain, even μ^* would incur low return for each state, yielding a 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 Q-Learning, Q^*_d and $\tilde{Q}_d$ do better than the baseline, with the latter doing significantly better.

The situation changes when the cliff sensor is added (we did not retest the potential function that did worse than the baseline), as shown in Table 1b. Though the sensor does not speed learning within a task, it provides useful information across tasks: whenever the cliff sensor activates, 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, More precisely, the state–action space $\mathbf{X}^+_d$ is enlarged and fewer state–action pairs are shared between tasks.

[3] We employ a standard real-valued GA with population size 100, no crossover and mutation with $p = 0.5$; mutation adds a random value $\delta \in [-0.05, 0.05]$. Policies are constructed by a softmax distribution over the chromosome values.

[4] Note that the addition of this sensor is not the same as the manual separation of state features for the value and potential function as done in [34,63]—see related work (Sect. 6). In the experiments reported in this section, both functions use the exact same set of features.

Table 1 Mean total reward and 95 % confidence interval for various shaping configurations and learning algorithms on the cliff domain

	Q-Learning	Sarsa
(a) Without sensor		
Q_d^{μ}	-19.77 ± 2.43	-512 ± 130
No shaping	-5.86 ± 0.12	-3.86 ± 0.10
Q_d^*	-5.13 ± 0.17	-3.96 ± 0.11
$\tilde{Q}_d$	-4.74 ± 0.19	-3.93 ± 0.11
(b) With sensor		
Q_d^{μ}	–	–
No shaping	-5.85 ± 0.13	-3.96 ± 0.11
Q_d^*	-5.44 ± 0.12	-3.67 ± 0.12
$\tilde{Q}_d$	-4.75 ± 0.17	-3.37 ± 0.13

All numbers $\times 10^4$

Under these circumstances, both Q_d^* and $\tilde{Q}_d$ significantly outperform baseline Sarsa, with the latter, again, doing best. The picture for Q-Learning remains largely the same.

5.1.2 Continuing cliff domain

The continuing cliff domain is the same as the episodic version with the cliff sensor, except that there are no terminal states. Instead there is a reward of zero on every step and ten for passing through the goal state and teleporting to the start state. We hypothesized an additional benefit for shaping here, since the reward function is sparser (see e.g. [39] for a formal relation between the benefit of shaping and sparsity of reward). To our surprise, however, this was not always the case. Ironically, one main reason seems to be the sparse reward function. In addition, the presence of an area of large negative reward next to the goal state makes the task even more difficult to learn. For increasing exploration rate ε, the optimal ε-greedy agent takes ever larger detours around the cliff, partly because of the sparse reward function; from around $\varepsilon = 0.05$, it huddles in the corner of the grid indefinitely, without ever attempting to reach the goal. For this reason, we used $\varepsilon = 0.01$ for all experiments.

Figure 2 shows the mean cumulative reward of Sarsa and Q-learning under various learning rates and shaping regimes. Here, we used two different methods for computing $\tilde{Q}_d$: one uses the exact values of the optimal ε-greedy policy for each task for $\varepsilon = 0.01$, as computed by a soft version of policy iteration[5], and the other uses solutions as computed by Sarsa run on each task in the domain with $\varepsilon = 0.01$, $\alpha = 0.07$, for 10^7 steps. These two value functions are different since, as it turns out, Sarsa converges to the wrong solution.

The figure shows a markedly different picture from the episodic cliff world results. Even at its optimal setting, Sarsa does not significantly outperform Q-Learning on this domain. Perhaps even more surprising, especially since Sarsa converges to the wrong solution, is that unshaped Sarsa at its optimal learning rate outperforms shaped Sarsa at its optimal setting; the shaped version dominates only from around $\alpha = 0.15$, and generally not significantly.

Figure 3 reveals what is happening. Figure 3a makes clear that, even though shaping has a disadvantage when measured over a large number of timesteps, it does provide an initial performance boost which lasts up to around 10^4 steps. However, as the two rightmost graphs show, the long-term learning dynamics of shaped Sarsa ultimately result in inferior performance: it is plagued by long periods of stasis, in which it keeps far from the cliff and thus

[5] In the policy improvement step, the policy is made only ε-greedy w.r.t. the value function.

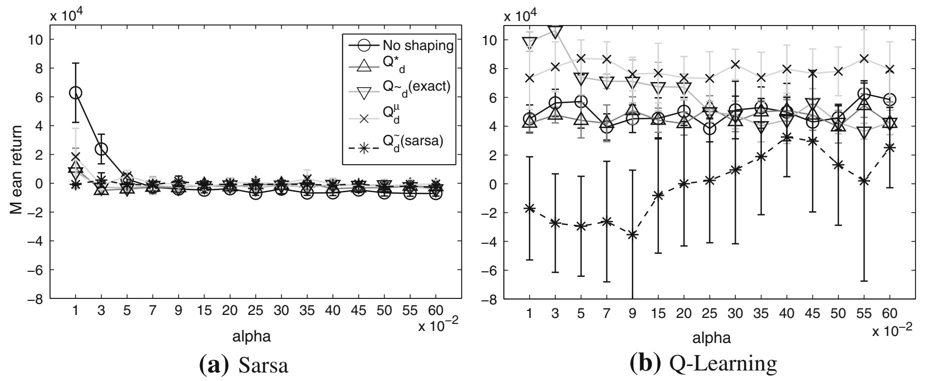


Fig. 2 Mean return and 95 % confidence intervals of Sarsa and Q-Learning on the continuing cliff domain, under various shaping regimes. Return is cumulative reward over 10^5 steps and averaged over all tasks in the domain (i.e. 8 runs)

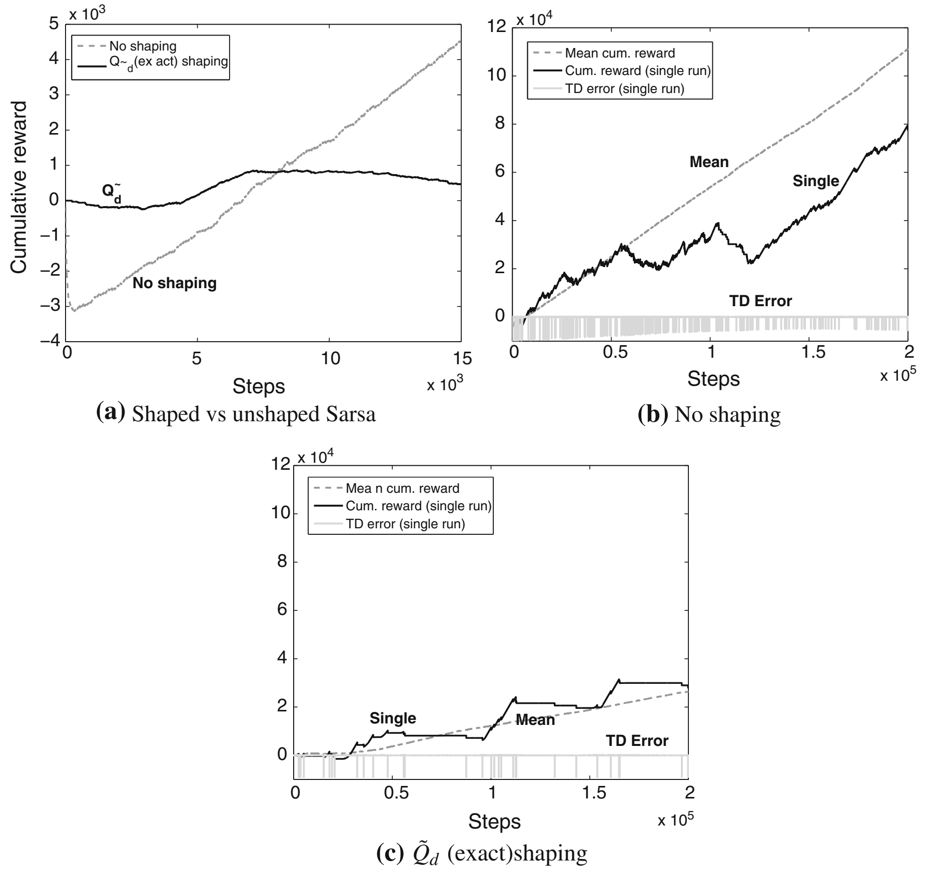
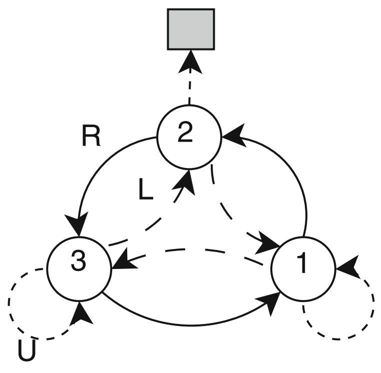
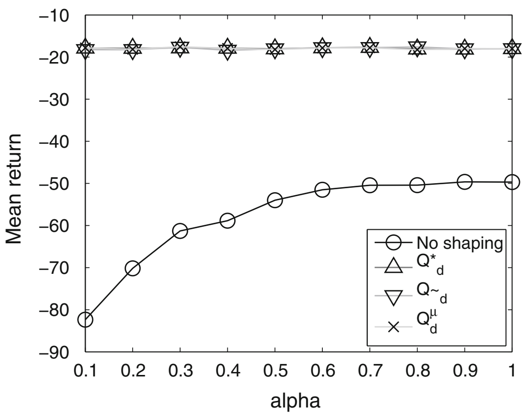


Fig. 3 Cumulative reward of unshaped and shaped Sarsa on the continuing cliff domain. Mean curve (*light gray*, *dashed*) represents average over 100 runs; also shown are example cumulative reward (*black*, *solid*) and TD error from a single run



(a) Example triangle task



(b) Performance

Fig. 4 Example task and mean return of Q-Learning on the triangle domain, under various shaping regimes. Return is cumulative reward over ten episodes and averaged over all tasks in the domain (i.e. 12 runs). All differences between the shaping methods are significant, except between $\tilde{q}_d$ and q_d^{μ}. Points shown are for $\varepsilon = 0.01$, the best-performing setting

the goal, and no reward is incurred. The likely reason is the same as for the eventual stasis of unshaped Sarsa (not shown), which results from Sarsa's inability, under the low exploration rate, to escape from the strong local optimum in the corner of the grid (recall that under higher exploration rates, the corner of the grid becomes the global optimum). Since shaped Sarsa is closer to convergence than unshaped Sarsa, it discourages the agent from approaching any location that might contain a cliff, resulting in an initial performance boost but also earlier onsets of stasis.

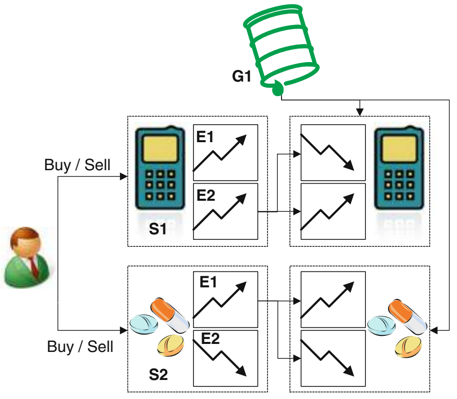
5.1.3 Triangle domain

Each task in the episodic Triangle domain consists of three non-terminal states, in which three actions can be taken (Fig. 4a). 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 task color: red (1) or green (0); if red, the agent receives a -10 penalty for self-transitions, in addition to the -1 step reward that is default in every task. x_3 is constant within a task, but may change from one task to the next. The goal may be at any state and the effect of actions L (dashed) and R (solid) may be reversed. Action U (dotted) always results in either a self-transition or a goal-transition (when the goal is next to the current state). There are thus 12 tasks in total.

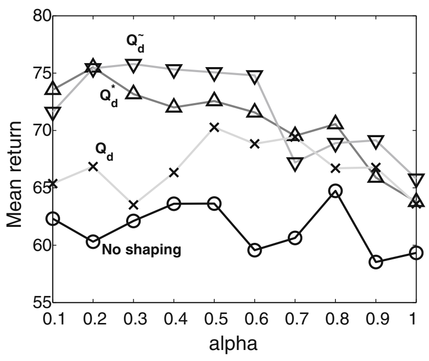
We compare performance of the different shaping regimes on this domain with a Q-Learning agent with discount factor $\gamma = 1$. Not surprisingly, there is no significant difference between the potentials in this domain: while Q_d^* estimates values higher than $\tilde{Q}_d$, which estimates higher than Q_d^{μ}, differences in estimates are minimal and the ordering of actions is the same.

5.1.4 Stock-trading domain

The binary stock-trading domain is an attractive domain for comparison since it is an established benchmark [13,37,71], is stochastic, and has an easily varied number of states and



(a) Stock domain



(b) Performance

Fig. 5 Schematic overview of, and mean Q-Learning return on, the stock-trading domain under various shaping regimes. **a** See main text. **b** Return is cumulative reward over 300 learning steps and averaged over 20 runs for all tasks in the domain (i.e. 240 runs). Differences between the methods are generally significant, except between $\tilde{Q}_d$ and Q^*_d

tasks. An example task is displayed in Fig. 5a. The domain consists of a number of sectors *S*, such as telecom (*s1* in the example) and pharmaceuticals (*s2*). Each contains a *E* items of equity (stock). Each stock is either rising (1) or falling (0). An agent can buy or sell all stocks of one whole sector at a time; sector ownership is indicated by a one (owned) or zero (not owned) in the state vector. Therefore, if the agent owns pharma but not telecom, the part of the state vector pertaining to stock in the example would be (0, 1, 1, 1, 0, 1, 0): the first two elements indicate sector ownership, the next three what telecom stocks are doing, and the final two what pharma stocks are doing. At each timestep, for each sector that it owns, the agent receives a reward of +1 for each stock that is rising and one for each stock that is falling. Thus in the example, the agent would earn a reward of one, since there is one stock rising in pharma.

The probability of stocks rising in a given sector s, P_s, depends on two factors: the number of stocks rising in s in the previous timestep, and the influence of G global factors (in the example, oil is the only global factor). How stocks and globals influence P_s is task-dependent. In the example, the only telecom stock of influence is e_2; in pharma, it is e_1. In a given task, stocks within a sector may be influenced by any combination of stocks in that sector. Stocks that are rising increase P_s; stocks that are falling decrease it.

Globals behave just like stocks in that they can rise (1) or fall (0). However, globals always affect all sectors simultaneously. The effect of a global varies per task; for a given rising global, it increases P_s in half the tasks, and decreases it in the other half, making its net cross-task effect zero. The exact formula for determining the probability that stocks in a given sector s will rise in a given task m is:

$$P_s^m = 0.1 + 0.8\frac{R_s^m + 3R_g^m}{I_s^m + 3G},$$

where I_s^m is the number of stocks of influence in s in m, G is the number of globals, R_s^m is the number of stocks of influence in s in m that are rising, R_g^m is the number of globals that are rising and increase P_s^m in m when rising, plus number of globals that are falling and increase P_s^m in m when falling.

Thus, in the example, assuming that oil prices are on the rise and increase the probability that stocks rise when falling, for $s =$ telecom,

$$P_s^m = 0.1 + 0.8\frac{1 + 3 \times 0}{1 + 3 \times 1} = 0.3.$$

A domain is defined by the tuple $\langle S, E, G\rangle$; the number of tasks in the domain is $(2^E - 1)2^G$. A state is represented by $S + SE + G$ binary features, so $|\mathbf{X}_m| = 2^{(S+SE+G)}$, and $|\mathsf{A}| = 2S$.

We ran a Q-Learning agent on a domain with $S = 1$, $E = 3$, and $G = 2$. As shown in Fig. 5, Q_d^* and $\tilde{Q}_d$ perform best in this domain; once again $\tilde{Q}_d$ seems to be slightly superior, although the difference is not significant. Q_d^μ lags behind, although its performance seems more resistant to changes in α; for higher learning rates, the difference with the other two shaping functions disappears.

5.1.5 Summary

Our experiments showed that which potential type is best is highly dependent on the domain, learning algorithm, and learning parameters. In the domains used in this section, prediction of optimal value Q_d^* never significantly outperforms the other shaping functions. One might expect that, since Q-Learning is an off-policy algorithm and Sarsa an on-policy one, Q_d^* would be best suited for the former and $\tilde{Q}_d$ for the latter. However, Q-Learning combined with $\tilde{Q}_d$ or Q_d^μ generally outperformed the other options, while no such effect could be observed for Sarsa. Finally, on the continuing cliff domain unshaped Sarsa outperforms shaped Sarsa for low learning rates. This shows that not only task relatedness can negatively affect transfer, but also the learning algorithm and learning parameters.

5.2 Evaluating representation selection

In this section, we test FS-TEK on the same domains used in Sect, 5.1. We compare to another backward elimination algorithm that also uses our definition of relevance, but does not make use of the multi-task information by extrapolating. This algorithm, iterated BKR (IBKR) is identical to FS-TEK, except that it only computes relevance for $k = |s|$, the length of the sequence of experienced tasks. In addition to IBKR, we compare FS-TEK to shaping functions constructed without FS; with randomly selected features; and with fixed features, in which only the true domain-relevant features are used.

In each experiment, a learning agent interacts sequentially with each domain. After each task, the agent constructs a new potential function based on the solutions of the tasks solved so far. The potential function is based on Q_c^* as defined in (17), unless specified otherwise. Likewise, relevance is computed based on Q_c^*. Although Sect. 5.1 showed that, in some domains, using a potential based on $\tilde{Q}_d$ works better, doing so is impractical here as it would require solving each task twice (once using Q-Learning, and then again using, e.g., Sarsa to compute the potential function).

For each domain, we use a fixed learning rate α and ε-greedy exploration, where $\varepsilon = 0.01$ and α is set to the best value found in Sect. 5.1. For each new potential function (i.e., for each number of tasks seen), we test the agent on ten tasks sampled from the domain; the whole procedure is repeated for 500 runs. Performance is measured only after three tasks, since FS-TEK requires at least this many tasks (since the exponential function to be estimated has three parameters).

While our implementation of FS-TEK largely corresponds to that outlined in Algorithms 1 and 2, there are some practical tweaks. Especially for a small number of experienced tasks,

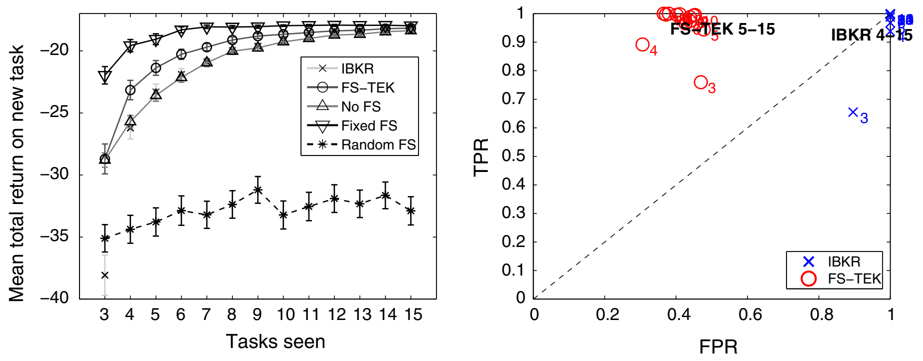


Fig. 6 Performance of shaping functions constructed using various FS methods (*left*) and ROC space (*right*) of the BKR and FS-TEK methods

the Jacobian of the estimated exponential may be ill-conditioned, often preventing reliable extrapolation and computation of the confidence interval. When this happens, we mark the feature as "unsure" until reliable estimates can be made in a subsequent iteration. In addition, for a given call to FS-TEK, any feature that is neither marked for removal nor "unsure" in any iteration, is kept indefinitely and not re-checked on subsequent iterations. This greatly improves speed and yields equal or better performance on the domains under consideration. We use the Levenberg–Marquardt algorithm [48] for the nonlinear least-squares fit.

5.2.1 Triangle domain

For this domain, we set the confidence level for IBKR to $\alpha = 0.15$, while for FS-TEK, we used $\alpha(k) = \min(\max(k - 4, 0) \times 0.06 + 10^{-4}, 0.3)$, i.e., it is 10^{-4} up to $k = 5$, from which point it linearly increases with k until a maximum of 0.3. An increasing confidence level for FS-TEK of this kind, and a constant level for IBKR, were found to work best by a coarse parameter search on this domain. Recall that higher confidence means that features are more easily marked as relevant. Increasing α makes sense since as more tasks are observed, estimates become more certain and the confidence interval can be tightened.

The left panel of Fig. 6 shows the mean return of shaping functions constructed using the various FS methods. FS-TEK achieves a significant performance improvement over both regular shaping and shaping with IBKR and is the first to match the performance of fixed FS, in which the correct features were hard-coded. The other methods also eventually reach that performance since, once the experienced tasks and their observed frequency approach the true set of tasks and distribution, the need for generalization disappears.

The right panel shows the receiver operating characteristic (ROC) space of the IBKR and FS-TEK method, plotting the false positive rate (FPR) against the true positive rate (TPR). In the context of this article, the TPR indicates the ratio of features correctly classified as domain relevant out of all domain relevant features, while the FPR indicates the ratio of features incorrectly classified as domain relevant out of all domain irrelevant features. Ideally, TPR = 1 and FPR = 0. From here on, we denote ROC by FPR/TPR, e.g., 0/1 in the ideal case. In the plot, the numbers next to the markers indicate the number of tasks seen.

The triangle domain contains one domain-irrelevant feature, namely the state number. The other three features are DR. In this domain, IBKR mostly achieves an ROC of 1/1, which amounts to never removing any features and is equivalent to the vanilla shaping function—

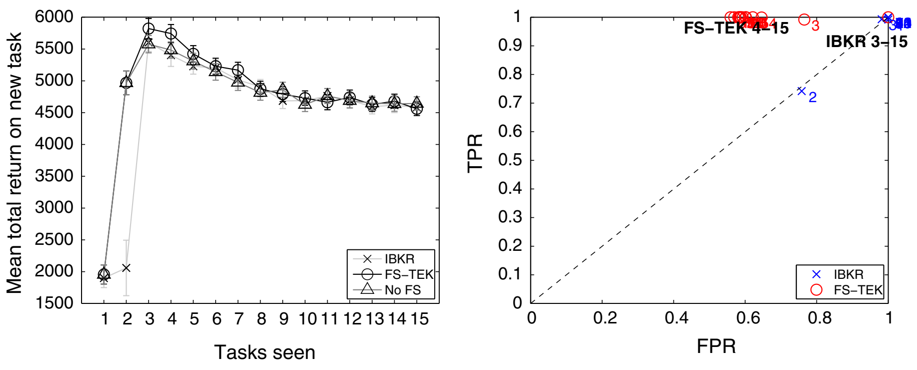


Fig. 7 Cliff domain when Q_d^* is used as potential. No fixed FS is shown since there are no features to remove; random FS is not shown because of its inferior performance

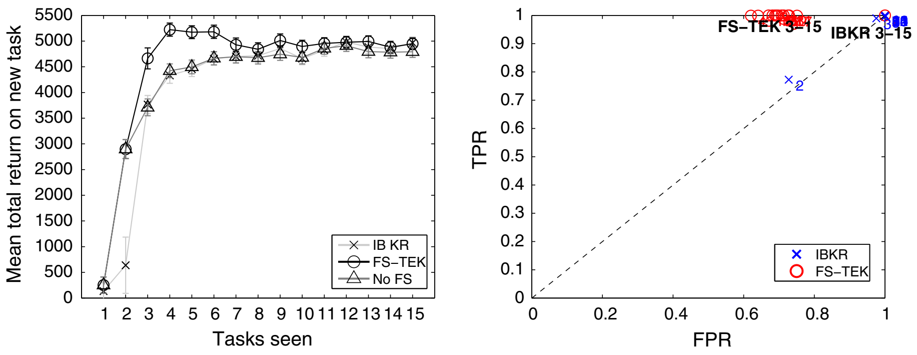


Fig. 8 Cliff domain when $\tilde{Q}_d$ is used as potential. No fixed FS is shown since there are no features to remove; random FS is not shown because of its inferior performance

indeed, their performance is nearly identical. FS-TEK does a better job, achieving an ROC of around 0.45/1 after five tasks seen, meaning it nearly always identifies the right DR features (as did IBKR), and in addition removes the irrelevant feature 55 % of the time.

5.2.2 Cliff domain

While the cliff domain contains no domain-irrelevant features, the first two features (encoding position) are only very weakly domain relevant. For the ROC space plot, therefore, we marked the position features as domain irrelevant; this shows that FS-TEK removes the position features about 40 % of the time (Fig. 7), which explains its slight performance gain.

Because of the peculiar progress of performance with number of observed tasks, we plotted performance from one observed task onward. The initial increase and subsequent decrease in performance with number of observed tasks for all methods is interesting. Since using $\tilde{Q}_d$ as potential was found to work better on this domain (Sect. 5.1), we were curious if the same trend would happen for $\tilde{Q}_d$. Figure 8 shows the results.

Clearly, the same trend does not happen for $\tilde{Q}_d$, and the benefit of FS-TEK is greater with this potential type. Moreover, while $\tilde{Q}_d$ does better than Q_d^* in the long run, as shown in Sect. 5.1, Q_d^* outperforms $\tilde{Q}_d$ for low number of observed tasks. The explanation must therefore be sought in difference between $\tilde{Q}_d$ and Q^*_d. With respect to the cliff sensor, $\tilde{Q}_d$ encourages the agent to move away from the cliff, while Q^*_d does not. Of course, when a cliff direction has not been encountered yet in previous tasks, both potentials have uniform preference over actions. With respect to position (i.e. when the cliff sensor shows no reading), Q^*_d pushes the agent towards the center of the grid; $\tilde{Q}_d$, on the other hand, pushes the agent towards the edges of the world. In short, $\tilde{Q}_d$ encourages exploration, but to shy away from a cliff once one is encountered; Q^*_d encourages sitting in the center, but to stay near a cliff once one is encountered.

The likelihood that the agent has encountered a given cliff increases with k. Therefore, for Q^*_d the likelihood that the agent will stick close to a cliff and fall into it increases with k. At some point, this likelihood together with the tendency to push the agent towards the center gains critical mass and performance declines. For $\tilde{Q}_d$, performance increases, since as the agent encounters more cliffs it is less likely to fall into them; in addition, this potential function increasingly encourages exploring the edges of the world and thus discovering the cliff in the current task sooner.

5.2.3 Stock domain

For the stock domain, we used the settings $e = 1$, $o = 2$ as detailed in Sect. 5.1.4, but tested for h ranging from 1 to 5. For $h = 1$, the size of the state–action space $|\mathbf{X}^+| = 32$ and $|\mathsf{M}| = 6$; these numbers double every value of h until for $h = 5$, $|\mathbf{X}^+| = 512$ and $|\mathsf{M}| = 96$. Recall from Sect. 5.1.4 that e represents sector ownership features, o represents number of stocks per sector and h represents task-dependent global variables that positively or negatively (depending on the task) affect the probability that stocks rise. For the current settings, stock ownership is completely irrelevant, while the stock features and action are domain relevant. The domain is challenging because the H features are strongly task relevant, much more so than the other features, but domain irrelevant (across all tasks their effect cancels out). In addition, the stock and action features are only weakly domain relevant. This means that the H features appear strongly domain relevant when an insufficient number of tasks have been experienced; moreover, they add noise to the shaping function when selected.

For $h < 4$, we used a confidence level $\alpha = 10^{-4}$ for FS-TEK and $\alpha = 10^{-3}$ for IBKR. For higher h, we used an $\alpha(k)$ that increases with k for FS-TEK; this was found to result in better performance. On the other hand, IBKR performed equally well for varying and constant α, so we kept it constant at $\alpha = 10^{-2}$ for $h \geq 4$.

The results are shown in Fig. 9. Generally, FS-TEK significantly outperforms all other methods, but its performance deteriorates until, for $h = 4$ and higher, it performs about as well as IBKR and vanilla shaping. It may seem that this is caused by the change in ROC from $h = 4$ onward, which in turn is caused by the change in the confidence level setting. However, the opposite is true: earlier experiments showed that a constant $\alpha = 10^{-4}$ resulted in a similar ROC as for lower h, but also in a mean return that was significantly worse than any other method. Instead, the constant performance of FS-TEK in terms of ROC versus the deteriorating performance shows that the task dynamics, rather than FS-TEK's ability to identify the correct features, are the cause of the performance decline. These dynamics are such that for low H, it is more important to not mistakenly select domain irrelevant features than to select the relevant ones (this also explains why FS-TEK does better than fixed FS for $H = 1$); for higher H, having a low FPR decreases in importance while a high TPR increases in importance. This makes sense since the more h variables there are, the more their effect is diluted, and the more important (relatively) the domain relevant features become.

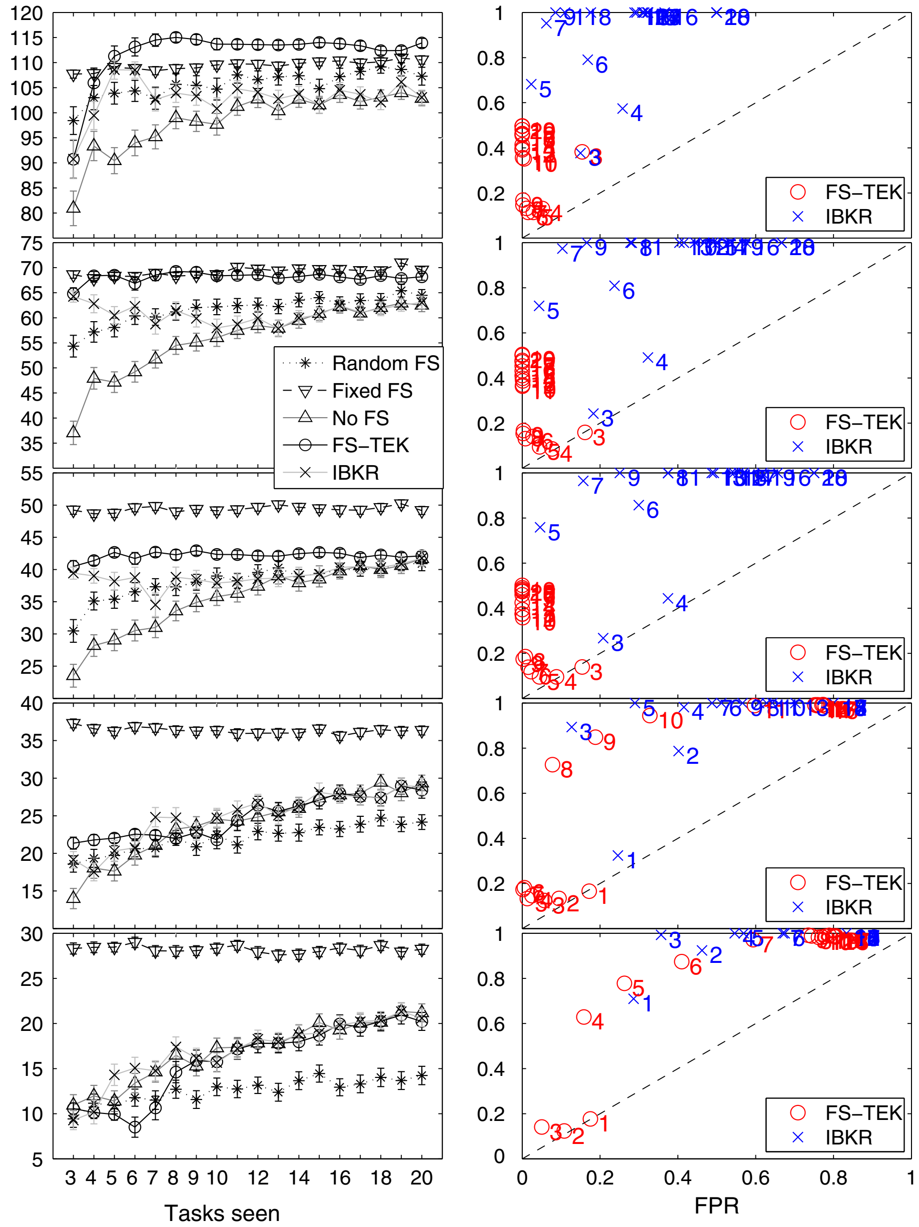


Fig. 9 Stock domain for $H = 1$ (*top*) to 5 (*bottom*). *Left column* plots tasks seen versus mean total return. *Right column* plots FPR versus TPR

5.2.4 Heat domain

To assess FS-TEK's ability to perform in a more challenging setting requiring function approximation, we also consider the heat domain, in which a circular agent with a radius of one learns to find a heat source in a 10×10 walled-off area. Note that here, we are using

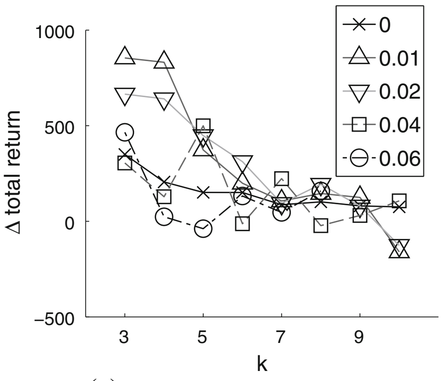
Eq. (10) to compute the potential function, and are using an abstraction function $\phi(\cdot)$ that computes abstractions directly in feature space.

State is continuous and consists of (x,y) position, the robot's heading in radians, and the intensity of the heat emitted by the source. The agent moves by going forward or backward, or turning left or right by a small amount.

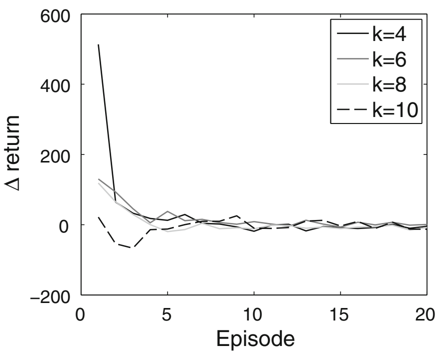
Reward per step is $-10/\sqrt(10 \times 10 + 10 \times 10) = 0.71$. An episode terminates when the heat source is within the agent's radius. The agent employs a jointly tile-coded Q-function approximator with four overlapping tilings, resulting in a total of 1,664 features. We run a Sarsa(λ) agent with $\lambda = 0.9$ and $\varepsilon = 0.05$ for 500 episodes, and compare the performance of FS-TEK to regular shaping under various levels of noise. Transition noise adds $\xi \sim \mathcal{N}(0, \sigma_{slip})$ to the agent's action and sensor noise adds $\xi \sim \mathcal{N}(0, \sigma_{sensor})$ to all state features. Since noise increases the chance of overfitting, our hypothesis is that FS-TEK should result in a greater benefit for higher levels of noise.

Figure 10 shows results for $\sigma_{slip} = 0.1$ and varying sensor noise. The left panel shows that FS-TEK achieves a significant jump in performance over the initial episodes for lower k, and as expected this benefit increases to some extent with the noise level. As observed previously, FS performs on par with regular shaping for higher k. The reason that FS has less benefit for noise levels above 0.02 seems to be that removing the right features becomes more difficult. This is confirmed by the ROC plots (Fig. 11), which show that FS-TEK's FPR increases as noise increases.

The right panel of Fig. 10 shows the benefit of FS per episode, for $\sigma_{sensor} = 0.02$ and various k. As k increases, there is a dip for early episodes; this dip is also the reason that FS and no FS perform on par for higher k. The likely explanation is that there is some small amount of information contained in the features that FS-TEK discards; therefore the regular potential function is based on more information. The fact that this happens for higher k stems from the fact that there is less overfitting because of the larger amount of data (previous experience) available. Since only the potential function and not the value function has a reduced feature set, this slightly wrong bias is quickly overcome.



(a) Improvement in total return per k



(b) Improvement per episode, for $\sigma_{sensor} = 0.02$

Fig. 10 Heat domain for $\sigma_{slip} = 0.1$. *On the left* the improvement in total return that FS-TEK achieves over regular shaping for the first five episodes, per k. *Each line* represents a different value for σ_{sensor}. *On the right* the improvement in return that FS-TEK achieves over regular shaping per episode, for $\sigma_{sensor} = 0.02$

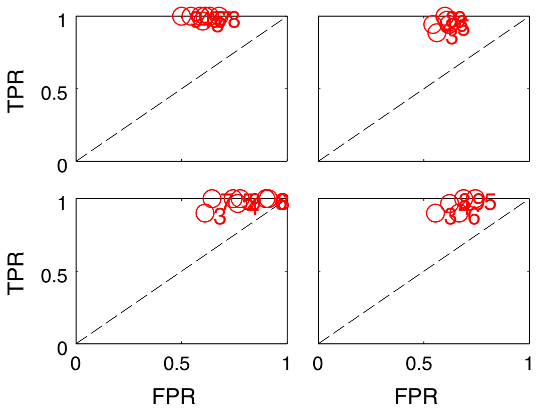


Fig. 11 FS-TEK ROC space for various noise levels on the heat domain. Clockwise starting in the *top left*, noise levels σ_{sensor} are 0.01, 0.02, 0.04, 0.06

5.2.5 Summary

Our experiments have shown that FS-TEK compares favorably to selection methods that do not explicitly exploit the history of experienced tasks. Although changes in potential function and increases in task and state space size affect the relative online return of FS-TEK compared to other methods, FS-TEK's performance in terms of ROC remains fairly constant. Our results furthermore suggest that FS-TEK can be flexibly tuned for a low FPR or high TPR, according to what is best for the domain; IBKR, on the other hand, has more trouble in filtering out domain irrelevant features, as expected. Lastly, we have succesfully applied FS-TEK to a continuous domain using a linear function approximator with binary features.

6 Related work

Our work is most related to shaping, multi-task (reinforcement) learning, and feature selection (FS). This section reviews work done in each of these areas and discusses their relationship to our own work.

6.1 Potential-based shaping

The theoretical result of Ng et al. [53], which showed that potential-based shaping functions preserve the optimal policy of the ground MDP for model-free RL, has recently been extended in various ways. Grześ and Kudenko [25] demonstrate empirically that scaling the potential can affect learning performance, and relate performance of a distance-to-goal-based potential to the discount factor γ and task size, showing that as task size increases, so should γ. Asmuth et al. [3] show that R-max, a popular model-based RL method, is still PAC–MDP [31] when combined with an admissable potential function $\Phi(\mathbf{x}, a) \geq Q^*(\mathbf{x}, a)$. In multi-agent RL, potential-based shaping provably preserves the Nash equilibrium in stochastic games [11,44]. Preservation of optimal policy and Nash equilibrium have also been shown to hold for potential functions that change while the agent is learning [12]. In practice, it has also been shown to improve strategies for the iterated Prisoner's dilemma [4] and robot soccer [10]. Learning a model of difference rewards, a concept in multi-agent learning related to shaping, has been shown to improve performance in air traffic control [56].

There have been a number of successes in learning potentials automatically. In single-task RL, one approach is to construct an initial shaping function based on intuition [38] or an initial task model [24], and refine it through interaction with the task. Elfwing et al. [15,16] evolve a shaping function that, when transferred to a real robot, results in better performance than when transferring Q-values. Other work has learned a shaping function on abstractions that are either provided [26] or also learned [49]. This latter approach is related to ours in that it explores different representations for the potential and value function. However, our work benefits from the MTRL setting in that it learns the abstractions offline, in between tasks, and therefore does not incur a cost while interacting with the next task.

Konidaris and Barto [34,35] were the first to learn a shaping function automatically in a multi-task environment by estimating the value function based on optimal solutions of previously experienced tasks. They base the value function on *problem space*, the Markov representation necessary for optimally solving tasks, and the potential function on *agent space*, the representation that retains the same semantics across tasks. However, they pre-specify both spaces. Similar pre-designed separate representions were employed in [63], in which an optimal reward function is searched on a distribution of tasks. As mentioned earlier, this article substantially extends this work by comparing different potential functions and offering a method for automatically discovering both task and domain-relevant representations.

6.2 Multi-task reinforcement learning

The field of MTRL has rapidly developed in recent years and is too large to survey comprehensively. Instead, we focus on approaches most similar to ours; see Taylor et al. [76] for an extensive survey of the field.

In [74], the average optimal Q-function of previously experienced tasks is used to initialize the model of a new task. Although the authors state that the average optimal Q-function is always the best initialization, our article has shown otherwise. Mehta et al. [51] assume fixed task dynamics and reward features, but different reward feature weights in each task. Given the reward weights of a target task, they initialize the task with the value function of the best stored source task policy given the new reward weights. In [77], the value function of a single source task is transformed via inter-task mappings and then copied to the value function of the target task. This work is a special case of ours since it considers only one source task. In this case, the cross-task potential as defined in Eq. 11 equals the value function of that source task by definition, and thus the initial policy based on the potential would be the same as the source task policy. However, taking more source tasks into account leads to significantly better performance on the target task. While we have not considered tasks with different feature and action spaces, our method can be applied to such scenarios using inter-task mappings, making the two approaches complementary.

Under a broader interpretation, initialization can also be performed by transferring source task experience samples to a batch-learning method in the target task [42]. Similar to other initialization methods, bias towards the source task knowledge decreases as experience with the target task accumulates. Other forms of initialization are to use a population of evolved source task policies as initial population for an evolutionary algorithm in the target task [78], or to use source task information to set the prior for Bayesian RL [41,89].

Advice-giving methods, which suggest an action to the agent given a state, are closely related to potential-based shaping. Torrey et al. [81] identify actions in source tasks with higher Q-values than others, and use this information to construct rules on action preferences that are added as constraints to a linear program for batch-learning Q-function weights in the target task. In [80], this work is extended by using inductive logic programming to extract the rules. Taylor et al. [78] learn rules that summarize a learned source task policy and incorporate these as an extra action in the target task, to be learned by a policy-search method. All these approaches are flexible in that they can deal with different state features and actions between tasks by a set of provided inter-task mappings. However, these mappings have to be provided by humans, except in [78], where they can be learned if provided with a description of task-independent state clusters that describe different objects in the domain. These clusters are similar to our notion of domain-relevant abstractions, which FS-TEK discovers automatically.

6.3 Representation learning

Both FS and state abstraction methods have been applied to single-task RL, with especially FS seeing a recent surge in interest [23,29,33,46,54,55]. Although similar methods have also been applied to transfer learning, most of these learn task-relevant representations for SL [2,5] or RL [18,30,37,40,60,70,75,83]; the latter aim to reduce the state space of the target task, or find good representations for value functions or policies. However, none of these approaches learn domain-relevant representations.

In the SL literature, work that does learn such representations includes lifelong [79] and multitask [8] learning. Both paradigms develop a representation that is shared between tasks by using training examples from a set of tasks instead of a single task, and show that this representation can improve performance by generalizing over tasks, if tasks are sufficiently similar. In RL, Foster and Dayan [20] identify shared task subspaces by identifying shared structure between task value functions; by augmenting the value function and policy representation with these subspaces, new tasks can be learned more quickly. While the idea of shared structure in task value functions is similar to ours, a limitation of this method is that it requires a single transfer function between tasks and only allows changes in the reward function. Similar to the domain-relevant features defined in this article and to the agent space of Konidaris and Barto [34], Frommberger and Wolter [22] define *structure space* as the feature space that retains the same semantics across tasks, and learn a structure space policy between tasks. Frommberger [21] applies the same concept to tile-coding functions for generalization within and across tasks. However, in both cases the structure space is hand-designed, as in [34].

While not explicitly concerned with representation learning, Sherstov and Stone [62] construct a task-independent model of the transition and reward dynamics by identifying shared outcomes and state classes between tasks, and using this for action transfer. This can be viewed as a kind of model-based domain relevance, and thus an interesting direction for future work on model-based MTRL.

There are two primary characteristics distinguishing our approach from the other cross-task methods discussed here. First, it captures within-task and cross-task relevance within a single definition. Second, it exploits multi-task structure by considering how representation relevance changes with increasing task sequence length and using that to predict relevance on the entire domain.

7 Discussion and future work

We have approached the multi-task learning problem by extracting shared information between tasks, identifying which state features are relevant to this information, and using potential functions based on these features to capture the information and guide the agent in new tasks.

7.1 Potential functions

This paper proposed and empirically compared three different methods for constructing potential functions based on past experience, and showed that which one is best depends on the domain, learning algorithm and parameters. Further studies are needed to determine what domain factors influence the best potential type. We conjecture that risk may be one such factor; since the optimal value function Q^* does not take exploration risk into account, potential functions based on Q^* might make it more likely that the agent enters "disaster states", if these are present in the domain. We observed this experimentally in the cliff domain in Sect. 5.1. In such domains, using more cautious potential functions such as $\tilde{Q}_d$ or Q_d^μ may work better. In addition, Q_d^μ seems to be more robust to changes in learning rate, which may be because it is constructed based on a single fixed cross-task policy.

Learning algorithm and parameters may also affect the benefit of transfer: in the cliff domain, an unshaped Sarsa agent outperformed the shaped agent. However, we think this scenario is rare, and applies in particular to scenarios in which an on-policy potential function is applied to a domain with a sparse reward function (where it may be difficult for the agent to improve upon the initial policy suggested by the potential), and disaster states. The latter may cause on-policy potential functions to too strongly bias the agent away from disaster states, causing the agent to get stuck in areas with no reward.

While using a potential function based on a different value function than the agent has learned may be beneficial, it may not always be practical. However, shaping, and the application of prior knowledge in general, are most useful in scenarios where the cost (and risk) of acquiring samples outweighs the cost of computation. In these cases, it may be well worth the extra cost of computing a potential based on a different value function.

7.2 Representations

Extrapolating relevance on tabular representations and linear binary function approximators has shown a significant benefit over regular shaping and methods that select features based only on the current task sequence. In our experience, FS-TEK's sole parameter, the confidence level, is an important tool for improving performance by tuning how aggressively FS-TEK discards features. In this sense, it is more an "aggressiveness" parameter than a confidence level, since setting this parameter to values near 10^{-3} is not uncommon. How to set the parameter is domain-dependent; increasing it tends to increase FPR but decrease TPR, and vice versa. Thus, the parameter setting should depend on whether it is more important to retain true positives or discard true negatives.

It is not immediately clear how to extend FS-TEK to non-binary linear or nonlinear function approximators, since the abstractions that FS-TEK relies upon may not be well defined there. One approach could be to learn these approximators using supervised regression on multiple task solutions simultaneously on task sequences of increasing length, and extrapolate changes in feature weights to obtain a measure similar to k-relevance. Extending k-relevance and FS-TEK to these domains is an important direction for future work.

While FS-TEK is an algorithm for FS, our definition of relevance applies to more general abstractions, e.g., including context-dependent ones. Therefore, a promising direction for future work is to extend FS-TEK to discover such abstractions. Similarly, while we tested our algorithm and definition of domain-relevance on potential functions, it is likely that the definition is equally applicable to other related transfer approaches, such as advice and rule-based methods [81]. Another possible application is to function approximators such as cross-task tile coding [21], which is based on a *structure space* defined by domain-relevant features. Our definition and algorithm could therefore enable the automatic discovery and construction of the appropriate coding, instead of hand-designing it. Domain relevance might also be useful for constructing informed priors in Bayesian MTRL. For example, Wilson et al. [89] use a hierarchical Bayesian approach in which distributions over tasks are drawn from a distribution over *classes* of tasks. Since purely domain-relevant features (i.e., features that are not task-relevant) similarly define task classes, they could form a useful basis for a prior.

Acknowledgments We thank George Konidaris, Hado van Hasselt, Eric Wiewiora, Lihong Li, Christos Dimitrakakis and Harm van Seijen for valuable discussions, and the anonymous reviewers for suggesting improvements to the original article.

Appendix 1: Glossary

Table 2 Glossary of terms

$\mathbf{X}_m$	State space of task m
A	Set of actions
P_m	Transition function of task m
R_m	Expected reward function of task m
$\mathbf{X}_m^+$	State–action space of task m
$\mathbf{x}$	A state
$\mathbf{x}^+$	A state–action pair
Q_m^*	Optimal value function of task m
D	A distribution over tasks
M	The set of all tasks
$\mathbf{X}_d$	Domain state space; union of all $\mathbf{X}_m$ in M
$\mathbf{X}_d^+$	Domain state–action space; union of all $\mathbf{X}_m^+$ in M
Φ	A potential function
Q_d	A cross-task value function; see Eq. 11
Q_d^*	Cross-task value function calculated over the optimal value function of each task; Eq. 11
$\tilde{Q}_d$	Cross-task value function calculated over the approximate value function of each task; Eq. 12
Q_d^μ	Cross-task value function based on a single cross-task policy μ; Eq. 13
ϕ	A state abstraction function; maps state $\mathbf{x}$ to abstract state $\mathbf{y}$
ϕ^-1	Inverse state abstraction function; maps $\mathbf{y}$ to the set of ground states comprising it
c	A sequence of tasks
c_i	An element from c; a task
Q_c	Cross-task value function computed over c instead of over the entire domain
$\bar{Q}_c(\phi)$	Abstract Q-function of a given Q_c, based on abstraction ϕ; Definition 2, Eq. 18
$\rho(\phi, Q_c)$	Relevance of ϕ with respect to Q_c; Definition 3, Eq. 20
$\rho_k(\phi)$	k-Relevance: expected relevance of ϕ on a task sequence of length k; Definition 5, Eq. 22

Appendix 2: Proof of Theorems in Sect. 4

As stated in Theorem 2, we are concerned with the case where the abstract Q-function is defined as in(18), i.e., the weighted average over state–action pairs in a given cluster. In this case, relevance equals the sum of weighted variances of the Q-values of ground state–action pairs corresponding to a given cluster $\mathbf{y}$ (21). Before proving the theorems, we show how relevance can be rewritten as a sum of covariances between Q-functions.

Variance of a weighted sum of n correlated random variables equals the weighted sum of covariances. We start by showing that any Q_c is a weighted sum of random variables (namely the Q-functions of each task in the sequence), and that therefore relevance can be written in terms of a weighted sum over covariances. Equation 17 is already a weighted sum, but we require a constant weight per random variable (task). Thus we rewrite (17) as

$$Q_c\left(\mathbf{x}^+\right) = \sum_{i=1}^{k} \Pr(c_i|c) Q^c_{c_i}\left(\mathbf{x}^+\right) \tag{26}$$

$$Q^c_{c_i}\left(\mathbf{x}^+\right) = \frac{\Pr\left(\mathbf{x}^+|c_i\right)}{\Pr(\mathbf{x}^+|c)} Q_{c_i}\left(\mathbf{x}^+\right), \tag{27}$$

where the last line is just a rescaling of Q_{c_i} depending on c, and $k = |c|$, the sequence length. Similarly, we define

$$Q^c_{\mathbf{y},c_i}\left(\mathbf{x}^+\right) = \frac{\Pr\left(\mathbf{x}^+|\mathbf{y}, c_i\right)}{\Pr\left(\mathbf{x}^+|\mathbf{y}, c\right)} Q_{c_i}\left(\mathbf{x}^+\right), \tag{28}$$

where $\Pr(\mathbf{x}^+|\mathbf{y}, c_i) = 0$ for all $\mathbf{x}^+ \notin \mathbf{X}^{\mathbf{y}}_{c_i}$, as the values of $Q^c_{c_i}$ on the domain $\mathbf{X}^{\mathbf{y}}_{c_i}$.

For ease of notation, we write $\mathrm{Var}(Q_c)$ for the variance $\mathrm{Var}(Q_c(\mathbf{X}^+_c))$, leaving the domain implicit, and similarly for the covariance. Note that $\Pr(c_i|c) = 1/k$. Then relevance (21) can be written as

$$\begin{aligned}\rho(\phi, Q_c) &= \sum_{\mathbf{y}\in\mathbf{Y}_c} \Pr(\mathbf{y}|c)\,\mathrm{Var}\left(\frac{1}{k}\sum_{i=1}^{k} Q^c_{\mathbf{y},c_i}\right)\\ &= \frac{1}{k^2}\sum_{\mathbf{y}\in\mathbf{Y}_c} \Pr(\mathbf{y}|c) \sum_{i=1}^{k}\sum_{j=1}^{k} \mathrm{Cov}\left(Q^c_{\mathbf{y},c_i}, Q^c_{\mathbf{y},c_j}\right)\end{aligned} \tag{29}$$

To see how relevance changes from one sequence to the next, we need to know how the covariance between two given tasks changes. For this purpose it is easier to write the covariance as $\mathrm{Cov}(X_i, X_j) = \mathrm{E}[X_i X_j] - \mathrm{E}[X_i]\,\mathrm{E}[X_j]$; then all we need to do is quantify both expectations. Let (c, m) be the new sequence formed by appending a task $m \in \mathsf{M}$ to a given sequence c. In the following, for ease of notation, assume an abstraction that leaves the original Q-function intact, i.e. $Q^c_{\mathbf{y},c_i} = Q^c_{c_i}$. The results can be extended to general abstractions by substituting $\mathbf{X}_c = \mathbf{X}^{\mathbf{y}}_c$, $\mathbf{X}_m = \mathbf{X}^{\mathbf{y}}_m$, $\Pr(\mathbf{x}^+|c) = \Pr(\mathbf{x}^+|\mathbf{y}, c)$, and $\Pr(\mathbf{x}^+|m) = \Pr(\mathbf{x}^+|\mathbf{y}, m)$.

Lemma 1 *The expected value of a given Q-function for a given task c_i in any sequence c is the same as the expected value of the original Q-function on c_i. That is,*

$$\mathrm{E}\left[Q^c_{c_i}\right] = \mathrm{E}\left[Q_{c_i}\right]. \tag{30}$$

Proof The expected value of any $Q^c_{c_i}$ is

$$\begin{aligned}
\mathrm{E}[Q^c_{c_i}] &= \sum_{\mathbf{x}^+ \in \mathbf{X}^+_c} \Pr\left(\mathbf{x}^+|c\right) Q^c_{c_i}\left(\mathbf{x}^+\right) \\
&= \sum_{\mathbf{x}^+ \in \mathbf{X}^+_{c_i}} \Pr\left(\mathbf{x}^+|c\right) Q^c_{c_i}\left(\mathbf{x}^+\right) + \sum_{\mathbf{x}^+ \in \mathbf{X}^+_c / \mathbf{X}^+_{c_i}} \Pr\left(\mathbf{x}^+|c\right) Q^c_{c_i}\left(\mathbf{x}^+\right) \\
&= \sum_{\mathbf{x}^+ \in \mathbf{X}^+_{c_i}} \Pr\left(\mathbf{x}^+|c\right) \frac{\Pr\left(\mathbf{x}^+|m\right)}{\Pr\left(\mathbf{x}^+|c\right)} Q_{c_i}\left(\mathbf{x}^+\right) \qquad (\text{Since } Q^c_{c_i} = 0\ \forall \mathbf{x}^+ \notin \mathbf{X}^+_{c_i}) \\
&= \sum_{\mathbf{x}^+ \in \mathbf{X}^+_{c_i}} \Pr\left(\mathbf{x}^+|m\right) Q_{c_i}\left(\mathbf{x}^+\right) = \mathrm{E}[Q_{c_i}].
\end{aligned}$$

□

To put bounds on the change in $\mathrm{E}[Q^c_{c_i} Q^c_{c_j}]$, we have

Lemma 2 *For a given sequence c and new sequence (c, m) formed by appending a task $m \in \mathsf{M}$ to c,*

$$0 \leq \left|\mathrm{E}\left[Q^{(c,m)}_{c_i} Q^{(c,m)}_{c_j}\right]\right| \leq \frac{k+1}{k} \left|\mathrm{E}\left[Q^c_{c_i} Q^c_{c_j}\right]\right|,$$

where $|\cdot|$ denotes absolute value.

Proof Let $Q_{i\cdot j}\left(\mathbf{x}^+\right) = Q_{c_i}\left(\mathbf{x}^+\right) Q_{c_j}\left(\mathbf{x}^+\right)$. Then

$$\begin{aligned}
\mathrm{E}[Q^c_{c_i} Q^c_{c_j}] &= \sum_{\mathbf{x}^+ \in \mathbf{X}^+_c} \Pr\left(\mathbf{x}^+|c\right) Q^c_{c_i}\left(\mathbf{x}^+\right) Q^c_{c_j}\left(\mathbf{x}^+\right) \\
&= \sum_{\mathbf{x}^+ \in \mathbf{X}^+_c} \frac{\Pr\left(\mathbf{x}^+|c_i\right) \Pr\left(\mathbf{x}^+|c_j\right)}{\Pr\left(\mathbf{x}^+|c\right)} Q_{i\cdot j}\left(\mathbf{x}^+\right).
\end{aligned}$$

For a given task pair, the only quantity that changes from one sequence to the next is $\Pr(\mathbf{x}^+|c)$. Let $f^c_{i,j}(\mathbf{x}^+) = \Pr(\mathbf{x}^+|c_i)\Pr(\mathbf{x}^+|c_j)/\Pr(\mathbf{x}^+|c)$, and recall that, for a sequence of length k, $\Pr(\mathbf{x}^+|c) = 1/k \sum_{i=1}^k \Pr(\mathbf{x}^+|c_i)$. Therefore, on a new sequence (c, m):

$$\begin{aligned}
f^{(c,m)}_{i,j}\left(\mathbf{x}^+\right) &= \frac{\Pr\left(\mathbf{x}^+|c_i\right) \Pr\left(\mathbf{x}^+|c_j\right)}{\left(\sum_{i=1}^k \Pr\left(\mathbf{x}^+|c_i\right) + \Pr\left(\mathbf{x}^+|m\right)\right) \Big/ (k+1)} \\
&= \frac{(k+1)\Pr\left(\mathbf{x}^+|c_i\right) \Pr\left(\mathbf{x}^+|c_j\right)}{k\Pr\left(\mathbf{x}^+|c\right) + \Pr\left(\mathbf{x}^+|m\right)}.
\end{aligned}$$

Taking the ratio of $f^{(c,m)}_{i,j}$ and $f^c_{i,j}$:

$$\begin{aligned}
f^{(c,m)}_{i,j}\left(\mathbf{x}^+\right) \Big/ f^c_{i,j}\left(\mathbf{x}^+\right) &= \frac{(k+1)\Pr\left(\mathbf{x}^+|c_i\right) \Pr\left(\mathbf{x}^+|c_j\right)}{k\Pr\left(\mathbf{x}^+|c\right) + \Pr\left(\mathbf{x}^+|m\right)} \times \frac{\Pr\left(\mathbf{x}^+|c\right)}{\Pr\left(\mathbf{x}^+|c_i\right)\Pr(\mathbf{x}^+|c_j)} \\
&= \frac{k\Pr\left(\mathbf{x}^+|c\right) + \Pr\left(\mathbf{x}^+|c\right)}{k\Pr\left(\mathbf{x}^+|c\right) + \Pr\left(\mathbf{x}^+|m\right)}. \qquad (31)
\end{aligned}$$

If $\Pr(\mathbf{x}^+|m)$ is larger (smaller) than $\Pr(\mathbf{x}^+|c)$, this ratio is smaller (larger) than one. It is largest when $\Pr(\mathbf{x}^+|m) = 0$, namely $(k+1)/k$, and at its smallest it is

$$\lim_{\Pr(\mathbf{x}^+|c)\downarrow 0} \frac{k \Pr\left(\mathbf{x}^+|c\right) + \Pr\left(\mathbf{x}^+|c\right)}{k \Pr\left(\mathbf{x}^+|c\right) + \Pr\left(\mathbf{x}^+|m\right)} = 0.$$

Since $Q_{i\cdot j}(\mathbf{x}^+)$ is constant from one sequence to the next, this leads to the bounds as stated in the lemma. □

Note that especially the lower bound is quite loose, since usually $\Pr(\mathbf{x}^+|c)$ will not be that close to zero. However, for our present purposes this is sufficient. Given these facts, the proof of Theorem 2 readily follows.

Theorem 2 Let ϕ be an abstraction with abstract Q-function as in Definition 2, and let $\rho_k = \rho_k(\phi)$ for any k. Let $d(x, y) = |x - y|$ be a metric on $\mathbb{R}$, and let $f(\rho_k) = \rho_{k+1}$ map k-relevance to $k+1$-relevance. Then f is a contraction; that is, for $k > 1$ there is a constant $\kappa \in (0, 1]$ such that

$$d\left(f(\rho_k),\ f(\rho_{k-1})\right) \le c d\left(\rho_k, \rho_{k-1}\right).$$

Furthermore, if $d(\rho_2, \rho_1) \neq 0$, then f is a strict contraction, i.e. there is a $\kappa \in (0, 1)$ such that the above holds.

Proof We need to show that $|\rho_{k+1} - \rho_k| < |\rho_k - \rho_{k-1}|$ for any $k > 1$. The relevance of a given sequence consists of the sum of the elements of the covariance matrix for that sequence, where each element has weight $1/k^2$. As illustrated in Fig. 1, from one sequence c to a new sequence (c, m), the ratio of additional covariances formed by the new task m with the tasks already present in c is $(2k-1)/k^2$ and thus rapidly decreases with k. The same figure also shows that change in relevance is caused by two factors: the expansion of the covariance matrices as sequence length increases coupled with the change in sequence probability, and change in the covariance between a given task pair from one sequence to the next. Suppose that the covariance of any task pair does not change from one sequence to the next. Then clearly, since the ratio of new covariance matrix elements changes with k as $(2k-1)/k^2$ and in addition the probability of all new sequences (c, m) formed from a given c sums up to the probability of c, $|\rho_{k+1} - \rho_k| \le |\rho_k - \rho_{k-1}|$ for any $k > 1$.

Now suppose that covariances do change from one sequence to the next. As Lemmas 1 and 2 show, the maximum change in covariance from any sequence c of length k to the next is $(k+1)\operatorname{Cov}(Q^c_{c_i}, Q^c_{c_j})/k$ for any i and j. This change also decreases with k, and therefore $|\rho_{k+1} - \rho_k| \le |\rho_k - \rho_{k-1}|$ for any $k > 1$. If $|\rho_2 - \rho_1| = 0$, then by this property the difference must stay 0 and $|\rho_{k+1} - \rho_k| = 0$ for any k. In all other cases, the change in relevance is a strict contraction, $|\rho_{k+1} - \rho_k| < |\rho_k - \rho_{k-1}|$, by the above arguments. □

In the following lemma, we distinguish between variances $\operatorname{Cov}(Q^c_{c_i}, Q^c_{c_j})$, $c_i = c_j$ and covariances $\operatorname{Cov}(Q^c_{c_i}, Q^c_{c_j})$, $c_i \neq c_j$. For $k = 1$, k-relevance consists solely of variances.

Lemma 3 *The ratio of the number of variances to the number of covariances decreases with k. For a given sequence c of length k* − 1*, the ratio of new variances in all new sequences of length k formed from c is*

$$\frac{2(k-1)+N}{N(2k-1)} \tag{32}$$

where $N = |\mathsf{M}|$.

Proof Let c be any sequence on a domain with $N = |\mathsf{M}|$ tasks. Assume task $m \in \{1, 2, \ldots, N\}$, occurs $o_m \in \{0, 1, \ldots, k\}$ times in c. Then any c can be represented by an N-dimensional vector $\mathbf{o}$: $\mathbf{o} = (o_1, o_2, \ldots, o_N)$. Note that for a given sample size k, $\sum_i o_i = k$ for any c. Lastly, denote by σ the sum of elements in the last column and row of the covariance matrix—as shown in Fig. 1, these are the elements added from one sequence c to the next (c, m).

Now take any sequence c of length $k - 1$, with task counts in vector $\mathbf{o}$. Form N new sequences of length k, where each sequence is formed by adding a task from M to c. To see how the ratio of covariances changes between $k - 1$ and k, all that matters is the ratio in σ. For any new sequence formed by adding task m to sequence c, there will be $2o_m + 1$ variances in σ. Hence, in total, taken over the N new sequences formed from c, there will be $2(o_1 + o_2 + \ldots + o_N) + N = 2(k - 1) + N$ new variances. In total, taken over the N new sequences, there are $N(2k - 1)$ covariances. So the ratio of variances in σ for a given sample size k is

$$\frac{2(k-1)+N}{N(2k-1)} \tag{33}$$

This ratio decreases with k. Therefore the ratio of covariances increases with k. □

We can now prove Theorem 3.

Theorem 3 If all tasks share the same distribution over state–action pairs $\Pr(\mathbf{x}^+|m)$, then ρ_k is a monotonically increasing or decreasing function of k, or is constant.

Proof The assumption of a single distribution over state–action pairs implies that covariances do not change from one sequence to the next. This follows from Lemma 1 and Eq. 31: since $\Pr(\mathbf{x}^+|m) = \Pr(\mathbf{x}^+|c)$, the ratio resolves to one and $\mathrm{E}\left[Q_{c_i}^{(c,m)} Q_{c_j}^{(c,m)}\right] = \mathrm{E}\left[Q_{c_i}^{c} Q_{c_j}^{c}\right]$.

The rest of the proof is by cases. Given $k = 1$, ρ_{k+1} can either be smaller than, greater than, or equal to ρ_k.

Case 1: $\rho_2 < \rho_1$.
Since $\rho_2 < \rho_1$, it follows that the expected value of a covariance is lower than that of a variance: ρ_1 is made up of all possible variances in the domain, while ρ_2 in addition consists of all possible covariances. Since covariances do not change from one k to the next, covariances must be lower on average. From Lemma 3, the ratio of covariances increases with k. Within the covariances, the frequency of a given task pair does not change, and the same holds for the variances. Therefore, since covariances do not change with k, ρ_k must get ever lower with k, and ρ_k is monotonically decreasing with k.
Case 2: $\rho_2 > \rho_1$.
By a similar argument to that for case 1, ρ_k is a monotonically increasing function of k.
Case 3: $\rho_2 = \rho_1$.
Therefore $|\rho_2 - \rho_1| = 0$, and $|\rho_{k+1} - \rho_k|$ must stay 0 by Theorem 2, which shows that ρ_k is constant. □

Appendix 3: Cross-task binary function approximator

This appendix derives an average cross-task linear function approximator from a set of linear function approximators per task, where approximators are assumed to have binary features.

Let $\mathbf{w}_m$ be the weight vector of the function approximator in task m and let $Q_m(\mathbf{x}) = \mathbf{w}_m^{\mathrm{T}}\mathbf{f}_\mathbf{x}$ be the Q-value of $\mathbf{x}$. We wish to find

$$Q_d = \operatorname*{argmin}_{Q_0}\left(\sum_{m\in\mathsf{M}}\Pr(m)\sum_{\mathbf{x}}\Pr(\mathbf{x}|m)\Big[Q_m(\mathbf{x}) - Q_0(\mathbf{x})\Big]^2\right) \tag{34}$$

$$= \operatorname*{argmin}_{\mathbf{w}_0}\left(\sum_{m\in\mathsf{M}}\Pr(m)\sum_{\mathbf{x}}\Pr(\mathbf{f}_\mathbf{x}|m)\Big[(\mathbf{w}_m - \mathbf{w}_0)^{\mathrm{T}}\mathbf{f}_\mathbf{x}\Big]^2\right). \tag{35}$$

Let

$$g(\mathbf{w}_0) = \left(\sum_{m\in\mathsf{M}}\Pr(m)\sum_{\mathbf{x}}\Pr(\mathbf{f}_\mathbf{x}|m)\Big[(\mathbf{w}_m - \mathbf{w}_0)^{\mathrm{T}}\mathbf{f}_\mathbf{x}\Big]^2\right)$$

Then with $\mathbf{w}^i$ the weight of feature i and N features,

$$\begin{aligned} g\big(\mathbf{w}_0^i\big) &= \sum_{m\in\mathsf{M}}\Pr(m)\sum_{\mathbf{x}}\Pr\big(\mathbf{f}_\mathbf{x}|m\big)\Big[\big(\mathbf{w}_m^i - \mathbf{w}_0^i\big)\mathbf{f}_\mathbf{x}^i\Big]^2 \\ &= \sum_{m\in\mathsf{M}}\Pr(m)\big(\mathbf{w}_m^i - \mathbf{w}_0^i\big)^2\sum_{\{\mathbf{x}:\mathbf{f}_\mathbf{x}^i=1\}}\Pr\big(\mathbf{f}_\mathbf{x}|m\big), \end{aligned}$$

which follows from the fact that $\mathbf{f}_\mathbf{x}^i$ is either zero or one.

Let $\Pr(\mathbf{f}_\mathbf{x}^i)$ indicate $\Pr(\mathbf{f}^i = \mathbf{f}_\mathbf{x}^i)$. Furthermore, $\Pr(\mathbf{f}_\mathbf{x}|m) = \Pr(\mathbf{f}_\mathbf{x}^1, \ldots, \mathbf{f}_\mathbf{x}^N|m)$, which equals $\Pr(\mathbf{f}_\mathbf{x}^1|m)\Pr(\mathbf{f}_\mathbf{x}^2|\mathbf{f}_\mathbf{x}^1, m)\ldots\Pr(\mathbf{f}_\mathbf{x}^N|\mathbf{f}_\mathbf{x}^{N-1}, \ldots, \mathbf{f}_\mathbf{x}^1, m)$. So

$$\begin{aligned} \sum_{\{\mathbf{x}:\mathbf{f}_\mathbf{x}^1=1\}}\Pr\big(\mathbf{f}_\mathbf{x}|m\big) &= \sum_{\{\mathbf{x}:\mathbf{f}_\mathbf{x}^1=1\}}\Pr\big(\mathbf{f}_\mathbf{x}^1|m\big)\Pr\big(\mathbf{f}_\mathbf{x}^2|\mathbf{f}_\mathbf{x}^1, m\big)\cdots\Pr\big(\mathbf{f}_\mathbf{x}^N|\mathbf{f}_\mathbf{x}^{N-1}, \ldots, \mathbf{f}_\mathbf{x}^1, m\big) \\ &= \Pr\big(\mathbf{f}^1 = 1|m\big)\sum_{\mathbf{f}^2,\ldots,\mathbf{f}^N}\Pr\big(\mathbf{f}^2|\mathbf{f}^1 = 1, m\big)\cdots\Pr\big(\mathbf{f}^N|\mathbf{f}^{N-1}, \ldots, \mathbf{f}^1 = 1, m\big) \\ &= \Pr\big(\mathbf{f}^1 = 1|m\big). \end{aligned}$$

This holds for all features i and therefore, if we multiply g with 0.5 for convencience when taking the partial derivative,

$$\begin{aligned} g\big(\mathbf{w}_0^i\big) &= \frac{1}{2}\sum_{m\in\mathsf{M}}\Pr\big(\mathbf{f}^i = 1, m\big)\big(\mathbf{w}_m^i - \mathbf{w}_0^i\big)^2, \\ \frac{\partial g}{\partial \mathbf{w}_0^i} &= \sum_{m\in\mathsf{M}}\Pr\big(\mathbf{f}^i = 1, m\big)\big(\mathbf{w}_0^i - \mathbf{w}_m^i\big). \end{aligned}$$

Setting this to zero gives

$$\begin{aligned} \sum_{m\in\mathsf{M}}\Pr\big(\mathbf{f}^i = 1, m\big)\mathbf{w}_0^i &= \sum_{m\in\mathsf{M}}\Pr\big(\mathbf{f}^i = 1, m\big)\mathbf{w}_m^i \\ \mathbf{w}_0^i\Pr\big(\mathbf{f}^i = 1\big) &= \sum_{m\in\mathsf{M}}\Pr\big(\mathbf{f}^i = 1, m\big)\mathbf{w}_m^i, \\ \mathbf{w}_0^i &= \sum_{m\in\mathsf{M}}\Pr\big(m|\mathbf{f}^i = 1\big)\mathbf{w}_m^i. \end{aligned}$$

References

1. Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences*, *10*, 25–61.
2. Argyriou, A., Evgeniou, T., & Pontil, M. (2008). Convex multi-task feature learning. *Machine Learning*, *73*(3), 243–272.
3. Asmuth, J., Littman, M., & Zinkov, R. (2008). Potential-based shaping in model-based reinforcement learning. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence* (pp. 604–609). Cambridge: The AAAI Press.
4. Babes, M., de Cote, E.M., & Littman, M. L. (2008). Social reward shaping in the prisoner's dilemma. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)* (pp. 1389–1392).
5. Baxter, J. (2000). A model of inductive bias learning. *Journal of Artificial Intelligence Research (JAIR)*, *12*, 149–198.
6. Bertsekas, D. P. (1995). *Dynamic programming and optimal control*. Belmont: Athena.
7. Boutilier, C., Dearden, R., & Goldszmidt, M. (2000). Stochastic dynamic programming with factored representations. *Artificial Intelligence*, *121*(1–2), 49–107.
8. Caruana, R. (1997). Multitask learning. *Machine Learning*, *28*(1), 41–75.
9. Caruana, R. (2005). Inductive transfer retrospective and review. In *NIPS 2005 Workshop on Inductive Transfer: 10 Years Later*.
10. Devlin, S., Grzes, M., & Kudenko, D. (2011). Multi-agent, reward shaping for robocup keepaway. In *AAMAS* (pp. 1227–1228).
11. Devlin, S., & Kudenko, D. (2011). Theoretical considerations of potential-based reward shaping for multi-agent systems. In *AAMAS, AAMAS '11* (pp. 225–232).
12. Devlin, S., & Kudenko, D. (2012). Dynamic potential-based reward shaping. In *AAMAS* (pp. 433–440).
13. Diuk, C., Li, L., & Leffler, B. R. (2009). The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *ICML* (p. 32).
14. Dorigo, M., & Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, *71*(2), 321–370.
15. Elfwing, S., Uchibe, E., Doya, K., & Christensen, H. (2008). Co-evolution of shaping: Rewards and meta-parameters in reinforcement learning. *Adaptive Behavior*, *16*(6), 400–412.
16. Elfwing, S., Uchibe, E., Doya, K., & Christensen, H. I. (2011). Darwinian embodied evolution of the learning ability for survival. *Adaptive Behavior*, *19*(2), 101–120.
17. Erez, T.,& Smart, W. (2008) What does shaping mean for computational reinforcement learning? In *7th IEEE International Conference on Development and Learning, 2008. ICDL 2008* (pp. 215–219).
18. Ferguson, K., & Mahadevan, S. (2006). Proto-transfer learning in markov decision processes using spectral methods. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning*.
19. Ferrante, E., Lazaric, A.,& Restelli, M. (2008). Transfer of task representation in reinforcement learning using policy-based proto-value functions. In *AAMAS* (pp. 1329–1332).
20. Foster, D. J., & Dayan, P. (2002). Structure in the space of value functions. *Machine Learning*, *49*(2–3), 325–346.
21. Frommberger, L. (2011). Task space tile coding: In-task and cross-task generalization in reinforcement learning. In *Proceedings of the 9th European Workshop on Reinforcement, Learning (EWRL9).*
22. Frommberger, L., & Wolter, D. (2010). Structural knowledge transfer by spatial abstraction for reinforcement learning agents. *Adaptive Behavior*, *18*(6), 507–525.
23. Geramifard, A., Doshi, F., Redding, J., Roy, N., & How, J. P. (2011). Online discovery of feature dependencies. In *ICML* (pp. 881–888).
24. Grześ, M., & Kudenko, D. (2009). Learning shaping rewards in model-based reinforcement learning. In *Proceedings of AAMAS 2009 Workshop on Adaptive Learning Agents*.
25. Grzes, M., & Kudenko, D. (2009). Theoretical and empirical analysis of reward shaping in reinforcement learning. In *ICMLA* (pp. 337–344).
26. Grześ, M., & Kudenko, D. (2010). Online learning of shaping rewards in reinforcement learning. *Neural Networks*, *23*(4), 541–550.
27. Gullapalli, V., & Barto, A.G. (1992). Shaping as a method for accelerating reinforcement learning. In *Proceedings of IEEE International Symposium on Intelligent, Control* (pp. 554–559).
28. Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, *3*, 1157–1182.
29. Hachiya, H., & Sugiyama, M. (2010). Feature selection for reinforcement learning: Evaluating implicit state-reward dependency via conditional mutual information. In *ECML/PKDD* (pp. 474–489).

30. Jong, N. K., & Stone, P. (2005). State abstraction discovery from irrelevant state variables. In *IJCAI-05*.
31. Kakade, S. M. (2003). *On the sample complexity of reinforcement learning*. Ph.D. Thesis, University College London, London.
32. Koller, D., & Sahami, M. (1996). Toward optimal feature selection. In *ICML* (pp. 284–292).
33. Kolter, J. Z., & Ng, A. Y. (2009). Regularization and feature selection in least-squares temporal difference learning. In *ICML* (p. 66).
34. Konidaris, G., & Barto, A. (2006). Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of 23rd International Conference on Machine Learning* (pp. 489–496).
35. Konidaris, G., Scheidwasser, I., & Barto, A. G. (2012). Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research*, *13*, 1333–1371.
36. Koren, Y., & Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of IEEE Conference on Robotics and Automation* (pp. 1398–1404).
37. Kroon, M., & Whiteson, S. (2009). Automatic feature selection for model-based reinforcement learning in factored MDPs. In *ICMLA 2009: Proceedings of the Eighth International Conference on Machine Learning and Applications* (pp. 324–330).
38. Laud, A., & DeJong, G. (2002). Reinforcement learning and shaping: Encouraging intended behaviors. In *Proceedings of 19th International Conference on Machine Learning* (pp. 355–362).
39. Laud, A., & DeJong, G. (2003). The influence of reward on the speed of reinforcement learning: An analysis of shaping. In *ICML* (pp. 440–447).
40. Lazaric, A. (2008). *Knowledge transfer in reinforcement learning*. Ph.D. Thesis, Politecnico di Milano, Milan.
41. Lazaric, A., & Ghavamzadeh, M. (2010). Bayesian multi-task reinforcement learning. In *ICML* (pp. 599–606).
42. Lazaric, A., Restelli, M., & Bonarini, A. (2008). Transfer of samples in batch reinforcement learning. In *ICML* (pp. 544–551).
43. Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for mdps. In *Aritificial Intelligence and Mathematics*.
44. Lu, X., Schwartz, H. M., & Givigi, S. N. (2011). Policy invariance under reward transformations for general-sum stochastic games. *Journal of Artificial Intelligence Research (JAIR)*, *41*, 397–406.
45. Maclin, R., & Shavlik, J. W. (1996). Creating advice-taking reinforcement learners. *Machine Learning*, *22*(1–3), 251–281.
46. Mahadevan, S. (2010). Representation discovery in sequential decision making. In *AAAI*.
47. Manoonpong, P., Wörgötter, F., & Morimoto, J. (2010). Extraction of reward-related feature space using correlation-based and reward-based learning methods. In *ICONIP* (Vol. 1, pp. 414–421).
48. Marquardt, D. (1963). An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal of Applied Mathematics*, *11*, 431–441.
49. Marthi, B. (2007). Automatic shaping and decomposition of reward functions. In *Proceedings of 24th International Conference on Machine Learning* (pp. 601–608).
50. Matarić, M. J. (1994). Reward functions for accelerated learning. In *Proceedings of 11th International Conference on Machine Learning*.
51. Mehta, N., Natarajan, S., Tadepalli, P., & Fern, A. (2008). Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning*, *73*(3), 289–312.
52. Midtgaard, M., Vinther, L., Christiansen, J. R., Christensen, A. M., & Zeng, Y. (2010). Time-based reward shaping in real-time strategy games. In *Proceedings of the 6th International Conference on Agents and Data Mining Interaction, ADMI'10* (pp. 115–125). Berlin, Heidelberg: Springer-Verlag.
53. Ng, A., Harada, D.,& Russell, S.(1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of 16th International Conference on Machine Learning*.
54. Parr, R., Li, L., Taylor, G., Painter-Wakefield, C.,& Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *ICML* (pp. 752–759).
55. Petrik, M., Taylor, G., Parr, R., & Zilberstein, S. (2010). Feature selection using regularization in approximate linear programs for markov decision processes. In*ICML* (pp. 871–878).
56. Proper, S.,& Tumer, K. (2012). Modeling difference rewards for multiagent learning (extended abstract). In *AAMAS, Valencia, Spain*.
57. Randløv, J.,& Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of 15th International Conference on Machine Learning*.
58. Rummery, G., & Niranjan, M. (1994). *On-line q-learning using connectionist systems*. Technical Report CUED/F-INFENG-RT 116, Engineering Department, Cambridge University, Cambridge.

59. Saksida, L. M., Raymond, S. M., & Touretzky, D. S. (1997). Shaping robot behavior using principles from instrumental conditioning. *Robotics and Autonomous Systems*, *22*(3–4), 231–249.
60. van Seijen, H., Whiteson, S., & Kester, L. (2010). Switching between representations in reinforcement learning. In *Interactive Collaborative, Information Systems* (pp. 65–84).
61. Selfridge, O., Sutton, R. S.,& Barto, A. G. (1985). Training and tracking in robotics. In *Proceedings of Ninth International Joint Conference on Artificial Intelligence*.
62. Sherstov, A. A., & Stone, P. (2005). Improving action selection in MDP's via knowledge transfer. In*Proceedings of the Twentieth National Conference on Artificial Intelligence*.
63. Singh, S., Lewis, R.,& Barto, A. (2009). Where do rewards come from? In *Proceedings of 31st Annual Conference of the Cognitive Science Society* (pp. 2601–2606).
64. Singh, S., & Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, *22*(1), 123–158.
65. Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, *8*(3), 323–339.
66. Singh, S. P., Jaakkola, T., & Jordan, M. I. (1994). Learning without state-estimation in partially observable markovian decision processes. In *ICML* (pp. 284–292).
67. Skinner, B. F. (1938). *The behavior of organisms: An experimental analysis*. New York: Appleton-Century-Crofts.
68. Snel, M.,& Whiteson, S. (2010). Multi-task evolutionary shaping without pre-specified representations. In *Genetic and Evolutionary Computation Conference (GECCO'10)*.
69. Snel, M., & Whiteson, S. (2011). *Multi-task reinforcement learning: Shaping and feature selection*. In *Proceedings of the European Workshop on Reinforcement Learning (EWRL)*.
70. Sorg, J., & Singh, S. (2009). Transfer via soft homomorphisms. In *Proceedings of 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)* (pp. 741–748).
71. Strehl, A. L., Diuk, C., & Littman, M. L. (2007). Efficient structure learning in factored-state mdps. In *AAAI* (pp. 645–650).
72. Sutton, R. (1983). Learning to predict by the method of temporal differences. *Machine Learning*, *3*, 9–44.
73. Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction*. Cambridge: The MIT Press.
74. Tanaka, F.,& Yamamura, M. (2003). Multitask reinforcement learning on the distribution of mdps. In *Proceedings of 2003 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2003)* (pp. 1108–113).
75. Taylor, J., Precup, D., & Panagaden, P. (2009). Bounding performance loss in approximate mdp homomorphisms. In Koller D., Schuurmans D., Bengio Y., & Bottou L. (Eds.), *Advances in Neural Information Processing Systems* (Vol. 21, pp. 1649–1656).
76. Taylor, M., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, *10*(1), 1633–1685.
77. Taylor, M., Stone, P., & Liu, Y. (2007). Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, *8*(1), 2125–2167.
78. Taylor, M. E., Whiteson, S., & Stone, P. (2007). Transfer via inter-task mappings in policy search reinforcement learning. In *AAMAS* (p. 37).
79. Thrun, S. (1995). Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing* (pp. 640–646).
80. Torrey, L., Shavlik, J. W., Walker, T.,& Maclin, R. (2010). Transfer learning via advice taking. In *Advances in Machine Learning I* (pp. 147–170). New York: Springer.
81. Torrey, L., Walker, T., Shavlik, J. W., & Maclin, R.: Using advice to transfer knowledge acquired in one reinforcement learning task to another. In *Proceedings of the Sixteenth European Conference on Machine Learning (ECML 2005)* (pp. 412–424).
82. Vlassis, N., Littman, M. L.,& Barber, D. (2011). *On the computational complexity of stochastic controller optimization in pomdps*. CoRR abs/1107.3090.
83. Walsh, T. J., Li, L., & Littman, M. L. (2006). Transferring state abstractions between mdps. In *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*.
84. Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.
85. Whitehead, S. D. (1991). A complexity analysis of cooperative mechanisms in reinforcement learning. In *Proceedings AAAI-91* (pp. 607–613).
86. Whiteson, S., Tanner, B., Taylor, M. E.,& Stone, P. (2011). Protecting against evaluation overfitting in empirical reinforcement learning. In *ADPRL 2011: Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement, Learning* (pp. 120–127).
87. Wiewiora, E. (2003). Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, *19*, 205–208.

88. Wiewiora, E., Cottrell, G.,& Elkan, C.(2003). Principled methods for advising reinforcement learning agents. In*Proceedings of 20th International Conference on Machine Learning* (pp. 792–799).
89. Wilson, A., Fern, A., Ray, S.,& Tadepalli, P. (2007). Multi-task reinforcement learning: A hierarchical Bayesian approach. In *ICML* (pp. 1015–1022).