

Behavioural Domain Knowledge Transfer for Autonomous Agents

Benjamin Rosman

Mobile Intelligent Autonomous Systems
Modelling and Digital Science
Council for Scientific and Industrial Research
South Africa

Abstract

An agent continuously performing different tasks in the same domain has the opportunity to learn, over the course of its operational lifetime, about the behavioural regularities afforded by the domain. This paper addresses the problem of learning a task independent behaviour model based on the underlying structure of a domain which is common across multiple tasks presented to an autonomous agent. Our approach involves learning action priors: a behavioural model which encodes a notion of local common sense behaviours in the domain, conditioned on either the state or observations of the agent. This knowledge is accumulated and transferred as an exploration behaviour whenever a new task is presented to the agent. The effect is that as the agent encounters more tasks, it is able to learn them faster and achieve greater overall performance. This approach is illustrated in experiments in a simulated extended navigation domain.

Introduction

Consider a long-lived robot that is expected to carry out a number of different tasks in the same (or similar) environments over the course of its lifetime. We assume that the tasks which will be assigned to it are varied but related, for example all pertaining to the manipulation of different objects around a home or factory, and using these objects in different ways. Furthermore, we assume that the set of tasks which will be encountered by this robot are unknown *a priori*, and as such it will be required to learn to complete these tasks as they arise.

This paper questions what can be learnt so as to ensure that the performance of the agent improves with each new task it is required to tackle. Clearly there will be commonalities between behaviours if the tasks exist within a common domain, in which case relearning everything from scratch is wasteful. To this end we seek to transfer knowledge from a bank of previous tasks in order to facilitate faster learning of new tasks (Taylor and Stone, 2009). This question is considered within the context of reinforcement learning.

We focus on one important aspect of transfer, in learning about the domain itself. While different tasks each nec-

essarily have different utility or reward functions, their dependence on a single domain implies that locally their solutions will contain behavioural elements which are persistent across multiple tasks. This can be seen for example in the fact that some constraints such as obstacle avoidance or the selection of shortest paths are general to large classes of problems. In addition, complex behaviours such as the use of particular tools at specific times may be useful as the components of full policies required to solve different tasks.

What is common to all of these examples is a preference for choosing certain actions over others under particular conditions. We thus seek to model these local action selection preferences as a form of prior knowledge which can be extended as the agent learns about different tasks in the domain. This then acts as a form of action pruning, which is also evident in human behaviour as a mechanism for constraining the search space during sequential decision making tasks (Huys et al., 2012; Harre, Bossomaier, and Snyder, 2012). We refer to this model of task independent domain specific local behaviours as *action priors* (Rosman, 2014).

Learning this domain behaviour model provides a factorisation of the policies of an agent into a domain space (actions taken as a result of the structure of the domain) and a task space (actions taken specifically to complete a particular task), which can be viewed as an extension of the concept of decomposing tasks into an agent space and problem space (Konidaris and Barto, 2006). In this paper, we show how such a model can be useful for seeding learning of new tasks in the same domain.

We consider two different ways of modelling action priors. If the domain remains unchanged between tasks, we can learn about action preferences conditioned on specific states. On the other hand, it is often more realistic to consider cases where the domain may change slightly between tasks. For example, the agent may move between different buildings, or the building may change as equipment is moved around. Under these conditions we can instead learn at the level of observation based behaviours. This provides transfer through a set of local controllers which can be more flexibly applied to new instances of a domain.

The action prior approach is illustrated in a simulated factory domain, which is an extended navigation domain that requires simple manipulation actions to be taken at specific locations. We show how action priors provide positive

knowledge transfer when learning multiple tasks, using both state and observation representations.

Domains and Tasks

Throughout this paper, we adopt the standard formalism used in reinforcement learning and assume that a problem faced by a learning agent is fully specified by a Markov Decision Process (MDP). An MDP is defined as a tuple (S, A, T, R, γ) , where S is a finite set of states, A is a finite set of actions which can be taken by the agent, $T : S \times A \times S \rightarrow [0, 1]$ is the state transition function where $T(s, a, s')$ gives the probability of transitioning from state s to state s' after taking action a , $R : S \times A \rightarrow \mathbf{R}$ is the reward function, where $R(s, a)$ is the reward received by the agent when transitioning from state s with action a , and $\gamma \in [0, 1)$ is a discount factor.

A policy $\pi : S \times A \rightarrow [0, 1]$ for an MDP describes the probability of selecting an action in each state. The return, generated from an episode of running the policy π is the accumulated discounted reward $\bar{R}^\pi = \sum_k \gamma^k r_k$, for r_k being the reward received at step k . The goal of a reinforcement learning agent is to learn an optimal policy $\pi^* = \arg \max_\pi \bar{R}^\pi$ which maximises the total expected return of an MDP, where typically T and R are unknown.

Many approaches to learning an optimal policy involve learning the value function $Q^\pi(s, a)$, giving the expected return from selecting action a in state s and thereafter following the policy π (Sutton and Barto, 1998). Q is typically learnt by iteratively updating values using the rewards obtained by simulating trajectories through the state space, e.g. the Q-learning algorithm (Watkins and Dayan, 1992). A greedy policy can be obtained from a value function defined over $S \times A$, by selecting the action with the highest value for any given state.

We now define a domain by the tuple $D = (S, A, T, \gamma)$, and a task as the MDP $\tau = (D, R, S_0)$. In this way we factorise the environment such that the state set, action set and transition functions are fixed for the whole domain, and each task varies only in the reward function, and the set of initial states $S_0 \subseteq S$.

This factorisation allows us to learn a model of the domain which is task independent, and it is the local action selection probabilities based on this domain information which we can transfer between tasks. Note that this domain model is accessible by studying the commonalities between several different tasks. As such, we distill domain specific knowledge from a set of task specific behaviours. We refer to this knowledge as action priors.

Action Priors

Consider a setting in which an agent has prolonged experience in a domain D . By this we mean that the agent has had to solve a set of tasks in D , and we use the resulting set of optimal policies to extract the behavioural regularities as a form of structural information about the domain.

Given an arbitrary set of tasks $\mathcal{T} = \{\tau\}$ in D and their corresponding optimal policies $\Pi = \{\pi_\tau^*\}$, we wish to learn

for each state s a distribution over the action set, representing the probability of each action being used in an optimal policy in s , aggregated over the tasks \mathcal{T} . This is then used as a model of domain behaviour in subsequent tasks to prioritise the actions in each state.

For each state $s \in S$, let the action priors $\theta_s(A)$ be this distribution over the action set A , describing the usefulness of each action in optimally solving the tasks in \mathcal{T} using the policies Π . To construct these action priors, we first define the utility of an action a in a state s under a policy π as

$$U_s^\pi(a) = \delta(\pi(s, a), \max_{a' \in A} \pi(s, a')), \quad (1)$$

where $\delta(\cdot, \cdot)$ is the Kronecker delta function: $\delta(a, b) = 1$ if $a = b$, and $\delta(a, b) = 0$ otherwise. As a result $U_s^\pi(a) = 1$ if and only if a is the best (or tied best) action in s under π .

Now consider the utility $U_s^\Pi(a)$ of an action a in a state s under a set of policies drawn from a *policy library* Π . This value is a weighted sum, given as

$$U_s^\Pi(a) = \sum_{\pi \in \Pi} w(\pi) U_s^\pi(a), \quad (2)$$

where $w(\pi) \in \mathbf{R}$ is a weight for the policy π . This weight can be used to indicate a prior probability over the policies if they are not equiprobable. The inclusion of this weight factor also allows us to include suboptimal policies in the policy library, by allocating them lower weights. In our experiments we do not assume to have knowledge about this distribution, and so uniformly set $w(\pi) = 1, \forall \pi \in \Pi$.

The fact that the domain knowledge is constructed from the policies solving a sampled set of tasks \mathcal{T} raises the possibility that \mathcal{T} is not wholly representative of the complete set of possible tasks in D . We adjust for this by forming an augmented policy set $\hat{\Pi}$, defined as $\hat{\Pi} = \Pi \cup \pi_0$, where π_0 is the uniform policy: $\pi_0(s, a) = \frac{1}{\|A\|}, \forall s \in S, a \in A$. The utility of this policy is then $U_s^{\pi_0}(a) = 1, \forall s \in S, a \in A$. In this case, the weight $w(\pi_0)$ represents the likelihood of encountering a new task outside the sampled set \mathcal{T} .

Given a state s , for each action a the utility $U_s^{\hat{\Pi}}(a)$ estimates the value of the state-action pair (s, a) in D under the augmented policy set $\hat{\Pi}$. Lacking more specific knowledge of the current task, an agent should choose actions according to these values. To select an action, sample from the action prior $a \sim \theta_s(A)$, where the action prior itself is sampled from a Dirichlet distribution $\theta_s(A) \sim \text{Dir}(U_s^{\hat{\Pi}}(a))$.

This Dirichlet distribution is parametrised by concentration parameters $(\alpha(a_1), \alpha(a_2), \dots, \alpha(a_{\|A\|}))^T$ and so for each state s , we maintain a count $\alpha_s(a)$ for each action $a \in A$, which is updated for every policy in $\hat{\Pi}$. The initial values of $\alpha_s(a) = \alpha_s^0(a)$ are known as the pseudocounts, and can be initialised to any value by the system designer to reflect prior knowledge. If these counts are the same for each action in a state, i.e. $\alpha_s(a) = k, \forall a \in A$ this returns a uniform prior, which results in each action being equally favourable in the absence of further information.

The pseudocount $\alpha_s^0(a)$ is a hyperprior which models prior knowledge of the tasks being performed by the agent. If the variance between tasks is small, or a large number

of training tasks are provided, then this hyperprior is set to a smaller value. However, if there is great diversity in the tasks, and the agent will not be expected to sample them thoroughly, then a larger hyperprior will prevent the action priors from over-generalising from too little data.

We wish these counts to describe the number of times an action a was considered optimal in a state s , across a set of policies Π . We thus augment Equation (2) and set

$$\alpha_s(a) = U_s^{\hat{\Pi}}(a) = \sum_{\pi \in \hat{\Pi}} w(\pi) U_s^\pi(a) \quad (3)$$

$$= \sum_{\pi \in \Pi} w(\pi) U_s^\pi(a) + \alpha_s^0(a). \quad (4)$$

This provides a natural intuition for the counts as the weighted utility of Π , and the hyperprior is then given by $\alpha_s^0(a) = w(\pi_0)$.

Typically, one does not want to maintain a full library of policies. As a result, the α counts can be learnt by the agent in an online manner as it learns the solutions to new tasks. In this way, when the agent solves task τ^{t+1} , the counts for each state-action pair can be updated by the values in π^{t+1} . This provides an online version of Equation (3) as

$$\alpha_s^{t+1}(a) \leftarrow \begin{cases} \alpha_s^t(a) + w(\pi^{t+1}) & \text{if } \pi^{t+1}(s, a) = \\ \max_{a' \in A} \pi^{t+1}(s, a') & \\ \alpha_s^t(a) & \text{otherwise.} \end{cases} \quad (5)$$

This model weights each action by the number of independent tasks which require the selection of that particular action in that state, which is used as the prior probability of that action in that state for any future tasks in the domain.

Observation Based Priors

In the previous section, action priors were defined as distributions over actions, conditioned on the current state. In this section we extend these definitions such that the action priors are instead conditioned on observations.

There are several reasons for this representation change. The most obvious being that the transition function T or even the state space S may not be task independent, and may instead differ between task instances. This may be the case for example, when an agent is tasked with exploring different buildings. It is not sensible to condition action priors on states, if the connectivity of those states changes between task instances. Instead, the agent should condition action priors on observable features of the states – features which would persist across tasks, even if state identities do not. This representation change allows the action priors to generalise to tasks in similar environments.

Another justification for using observation based priors is that one may not always have full observability of s , meaning that different states cannot be uniquely distinguished. This is the case in partially observable reinforcement learning problems (Kaelbling, Littman, and Cassandra, 1998) which typically require the solution of partially observable Markov decision processes (POMDPs). State information, such as exact world coordinates of a mobile robot, is not always accessible. Similarly, there may be states in S which

have not been explored during training, and so no action prior would be available for these states, which may be required later. In both these scenarios, it is again sensible to instead base the action priors on whatever features of the state are observed (perhaps with high confidence). Observation based action priors provide the ability to transfer to unseen state and action combinations.

Basing these priors on observations rather than states involves changing the dependence of θ from $s \in S$ to $\phi : S \rightarrow \mathcal{O}$, where ϕ is the mapping from state space S to the observation (perception) space \mathcal{O} . The observed features of s are described by $\phi(s)$. The state based priors can thus be considered as a special case of observation based priors, with $\phi(s) = s$.

Note that we are not solving a partially observable problem, but are instead building domain models based on some partial information signals. Using observations rather than exact state descriptions allows for more general priors, as they are applicable to different states emitting the same observations. This also enables pooling of the experience collected from different states with similar observations, to learn more accurate models. Note that this does not necessarily imply that the observation space is smaller, but that it is structured in that similar observations may suggest similar behaviours. This observation space could in fact be clustered into templates (Rosman and Ramamoorthy, 2012a).

There is a trade-off between the generality and the usefulness of the priors. This is a function of the choice of observation features, and the amount of action information captured by these features. These depend on properties of the tasks and environments. More general observation features imply less informative action priors. On the other hand, the more specific these features are (up to exact state identification), the less portable they are to new states.

Exploration using Action Priors

Action priors provide the agent with a model of transferred knowledge about which actions are sensible in situations in which the agent has several choices to explore. They are thus useful for seeding search in a policy learning process. We demonstrate this modified exploration process with an adaptation of the Q-learning algorithm (Sutton and Barto, 1998), called ϵ -greedy Q-learning with State-based Action Priors, or ϵ -QSAP (Rosman and Ramamoorthy, 2012b), which is shown in Algorithm 1. The parameter $\epsilon \in [0, 1]$ controls the trade-off between exploration and exploitation, and $\alpha^Q \in [0, 1]$ denotes the learning rate (not to be confused with the Dirichlet distribution counts $\alpha_s(a)$). Both α^Q and ϵ are typically annealed after each episode.

The difference between this and standard ϵ -greedy Q-learning is seen on line 5. This is the action selection step, consisting of two cases. The first case deals with exploiting the current policy stored in $Q(s, a)$ with probability $1 - \epsilon$, and the second case with exploring other actions $a \in A$ with probability ϵ . The exploration case is typically handled by choosing the action uniformly from A , but instead we choose with probability based on the prior $\theta_s(a)$ to shape the action selection. Although this algorithm uses the state

Algorithm 1 ϵ -greedy Q-learning with State-based Action Priors (ϵ -QSAP)

Require: action prior $\theta_s(a)$

- 1: Initialise $Q(s, a)$ arbitrarily
 - 2: **for** every episode $k = 1 \dots K$ **do**
 - 3: Choose initial state s
 - 4: **repeat**
 - 5: $a \leftarrow \begin{cases} \arg \max_a Q(s, a), & \text{w.p. } 1 - \epsilon \\ a \in \mathcal{A}, & \text{w.p. } \epsilon \theta_s(a) \end{cases}$
 - 6: Take action a , observe r, s'
 - 7: $Q(s, a) \leftarrow Q(s, a) + \alpha^Q [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
 - 8: $s \leftarrow s'$
 - 9: **until** s is terminal
 - 10: **end for**
 - 11: **return** $Q(s, a)$
-

based action priors, conditioning on observations instead is a trivial extension.

The effect of this modified exploration mechanism is that the agent exploits the current estimate of the optimal policy with high probability, but also explores, and does so with each action proportional to the number of times that action was favoured in previous tasks under the transferred model. This leverages the assumption that there is inherent structure in the domain which gives rise to behavioural regularities which can be identified across multiple tasks.

Exploration thus occurs by randomly sampling actions according to the probabilities that they are optimal, given the previously encountered tasks. Choosing actions randomly in this way is an established action selection mechanism (Wyatt, 1997; Dimitrakakis, 2006) known as Thompson sampling (Thompson, 1933) which, rather than being a heuristic, has been shown to be a principled approach to dealing with uncertainty (Ortega and Braun, 2013). As these probabilities relate to the domain, rather than the current task, this is still traded off against exploiting the current Q-function.

Experiments

To illustrate the action prior approach, we introduce the factory domain: an extended navigation domain involving a mobile manipulator robot in a factory. The layout of the factory consists of an arrangement of walls, with some procurement and assembly points placed around the factory. Additionally there are express routes, which represent preferred paths of travel, corresponding to regions where collisions with other factory processes may be less likely. The domain used in these experiments is shown in Figure 1.

The robot has an action set consisting of four movement actions (*North*, *South*, *East* and *West*), each of which will move the robot in the desired direction provided there is no wall in the target position, a *Procure* action, and an *Assemble* action. *Procure*, when used at procurement point i , provides the robot with the materials required to build component i . *Assemble*, when used at assembly point i , constructs component i , provided the robot already possesses the materials required for component i .

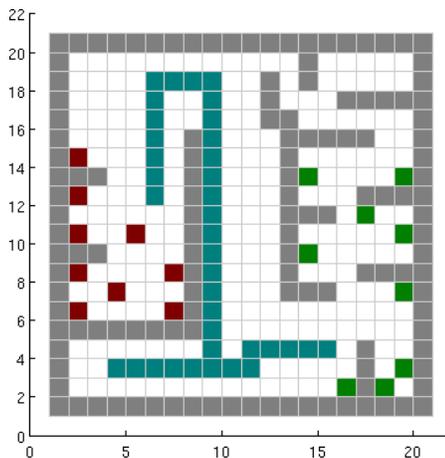


Figure 1: The factory domain. Grey cells are obstacles, white cells are free space, green cells are procurement points, red cells are assembly points, and cyan cells are express routes.

A task is defined as a list of components which must be assembled by the robot. The domain has 9 components, and so this list can range in length from 1 to 9, giving a total of $2^9 - 1$ different tasks.

The task rewards are defined as follows. All movement actions give a reward of -2 , unless that movement results in the robot being on an express route, for a reward of -1 . Collisions are damaging to the robot and so have a reward of -100 . *Procure* at a procurement point corresponding to an item in the task definition which has not yet been procured gives a reward of 10. *Procure* executed anywhere else in the domain yields -10 . *Assemble* at an assembly point for an item in the list which has already been procured but not assembled gives 10, and any other use of the *Assemble* action gives -10 . Successful completion of the task gives 100 and the episode is terminated.

This domain has many local task independent behaviours which could be learnt by the robot. On a local level, this includes avoiding collisions with walls, preferring express routes over standard free cells, and not invoking a *Procure* or *Assemble* action unless at a corresponding location. As all tasks are defined as procuring and assembling a list of components, this additionally provides scope for learning that regardless of the components required, the robot should first move towards and within the region of procurement points until all components have been procured, after which it should proceed to the region of assembly points.

Results with State Action Priors

The results in Figure 2, which compares the performance per episode of a learning agent using a set of different priors, demonstrates that using action priors reduces the cost of the initial phase of learning, which is largely concerned with coarse scale exploration. This figure also shows comparative performance of Q-learning with uniform priors (i.e. “standard” Q-learning), as well as with two different hand specified priors; an “expert” prior and an “incorrect” prior.

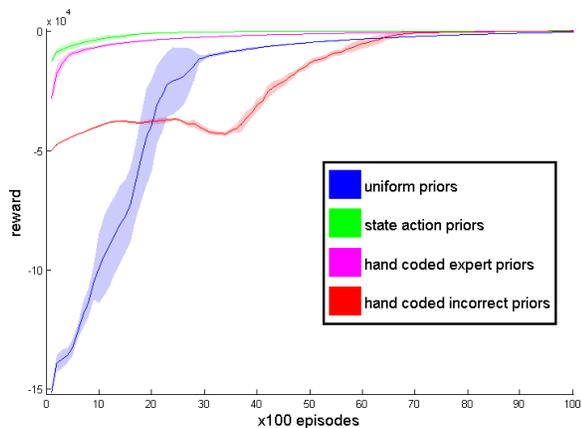


Figure 2: Comparative performance between Q-learning with uniform priors, state based action priors learned from 35 random tasks, and two different pre-specified priors (see text for details). The figure shows learning curves averaged over 15 runs, where each task was to assemble 4 components, selected uniformly at random. The shaded region represents one standard deviation.

The “expert” prior is defined over the state space, to guide the agent towards the procurement area of the factory if the agent has any unprocured items, and to the assembly area otherwise. This prior was constructed by a person, who was required to specify the best direction for each state in the domain. We note that this prior is tedious to specify by hand, as it involves an expert specifying preferred directions of motion for the entire state space of the agent (number of states in the factory \times number of different item configurations). Note that although the performance is very similar, this prior does not perform as well as the learnt prior, likely due to a perceptual bias on behalf of the expert’s estimation of optimal routing. We also compare to an “incorrect” prior. This is the same as the expert prior, but we simulate a critical mistake in the understanding of the task: when the agent has unprocured items, it moves to the assembly area, and otherwise to the procurement area. This prior still provides the agent with an improvement in the initial episodes over uniform priors, as it contains some “common sense” knowledge including not moving into walls, moving away from the start location, etc. Q-learning is still able to recover from this error, and ultimately learn the correct solution.

Figure 3 shows the speed up advantage in learning a set of $N = 40$ tasks, starting from scratch and then slowly accumulating the prior from each task, against learning each task from scratch. This illustrates the improved transfer benefits through the continued acquisition of more domain knowledge over time. This case is for a simple version of the task, which involved procuring and assembling a single item. As a result, all task variants are likely to have been encountered by the time the agent solves the final tasks.

On the other hand, Figure 4 shows that a similar effect can be observed for the case of a more complicated task, requiring the assembly of 4 randomly selected items. In this case, even by the time the learning agent has accumulated

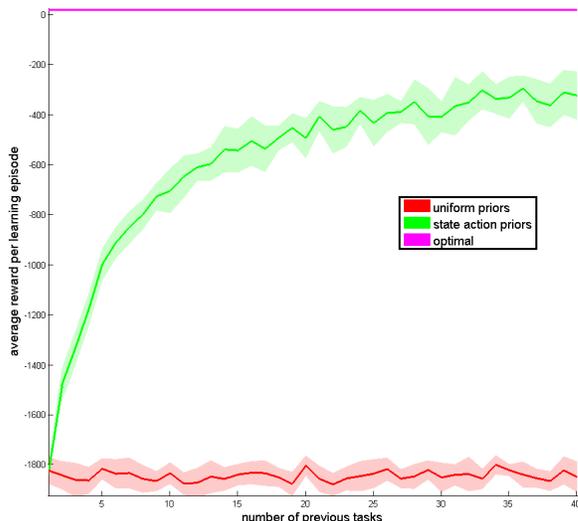


Figure 3: Comparative performance between Q-learning with uniform priors, and accumulating state based action priors from an increasing number of tasks. The number of prior tasks ranges from 0 to 40. These curves show the average reward per learning episode averaged over 10 runs, where the task was to assemble 1 random component. The shaded region represents one standard deviation. The “optimal” line refers to average performance of an optimal policy, which will necessarily be higher than the per episode reward of a learning algorithm.

a prior composed from 40 tasks, it has only experienced a small fraction of the possible tasks in this domain. Despite this, the agent experiences very similar benefits to those seen in the single item case.

Results with Observation Action Priors

In order to demonstrate the effect of using the observation action priors, we present a modification of the factory domain wherein the factory floor layout changes for each task. The map consists of a 3×3 lattice of zones, each of which is 6×6 cells. There is an outer wall, and walls in between every two zones, with random gaps in some (but not all) of these walls, such that the entire space remains connected. Additionally, each zone contains randomly placed internal walls, again maintaining connectivity. Two zones are randomly chosen as procurement zones, and two zones as assembly zones. Each of these chosen zones has either four or five of the appropriate work points placed at random. Examples of this modified factory domain are shown in Figure 5.

This modified domain has a different layout for every task, and so every task instance has a different transition function T . This is in contrast to the original factory domain, where each task differed only in reward function R . State based action priors can therefore not be expected to provide the same benefits as before. We instead use observation priors and discuss four particular feature sets.

Figure 6 demonstrates the improvement obtained by using

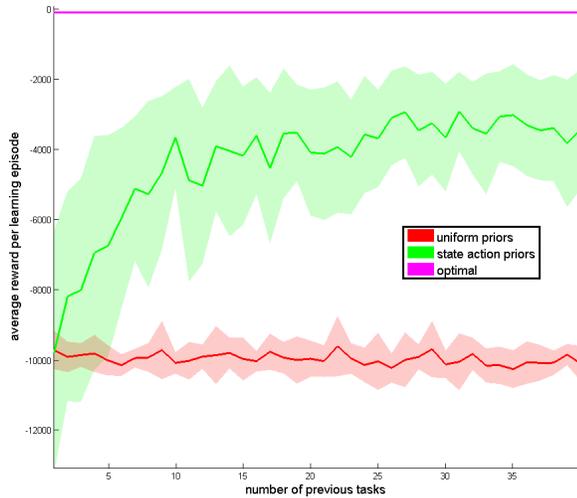


Figure 4: Comparative performance between Q-learning with uniform priors, and accumulating state based action priors from an increasing number of tasks. The number of prior tasks ranges from 0 to 40. These curves show the average reward per learning episode averaged over 10 runs, where the task was to assemble 4 random components. The shaded region represents one standard deviation. The “optimal” line refers to average performance of an optimal policy, which will necessarily be higher than the per episode reward of a learning algorithm.

observation priors over state priors in this modified domain. Note here that the state priors still provide some benefit, as many of the corridor and wall placings are consistent between task and factory instances. Figure 6 shows the effect of four different observation priors:

- ϕ_1 : Two features – the type of the terrain in the cell occupied by the agent (in $\{free, wall, procure-station, assembly-station\}$), and a ternary flag indicating whether any items still need to be procured or assembled.
- ϕ_2 : Four features – the types of terrain of the four cells adjacent to the cell occupied by the agent.
- ϕ_3 : Six features – the types of terrain of the cell occupied the agent as well as the four cells adjacent to that, and a ternary flag indicating whether any items need to be procured or assembled. Note that the features in ϕ_3 are the union of those in ϕ_1 and ϕ_2 .
- ϕ_4 : Ten features – the types of terrain of the 3×3 grid of cells around the agent’s current position, and a ternary flag indicating whether any items need to be procured or assembled.

These four observation priors can all be seen to contain information relevant to the domain, and all provide an improvement over the baselines. There is however a significant performance difference between the four feature sets.

Surprisingly, Figure 6 shows that the most beneficial feature set is ϕ_3 , with ϕ_2 performing almost as well. The fact that the richest feature set, ϕ_4 , did not outperform the others seems counterintuitive. The reason for this is that using these

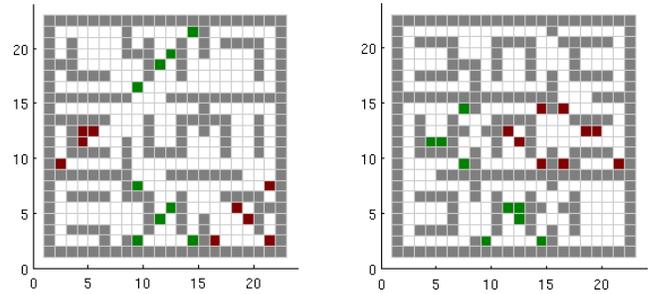


Figure 5: Two instances of the modified factory domain. Grey cells are obstacles, white cells are free space, green cells are procurement points, and red cells are assembly points. The procurement and assembly points count as traversable terrain.

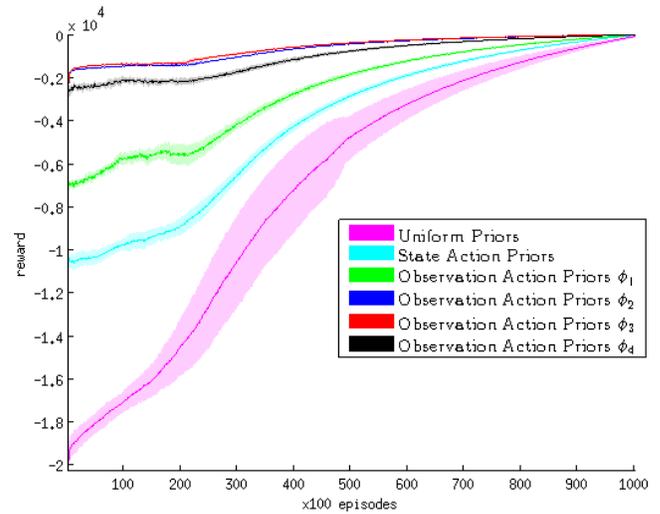


Figure 6: Comparative performance in the modified factory domain between Q-learning with uniform priors, state based action priors, and four different observation based action priors: ϕ_1 , ϕ_2 , ϕ_3 and ϕ_4 . These curves show the average reward per episode averaged over 10 runs, where the task was to assemble 4 random components. In each case the prior was obtained from 80 training policies. The shaded region represents one standard deviation.

ten features results in a space of $4^9 \times 3$ observations, rather than the $4^5 \times 3$ of ϕ_3 . This factor of 256 increase in the observation space means that for the amount of data provided, there were too few samples to provide accurate distributions over the actions in many of the observational settings.

These results indicate the importance of having an informative yet sparse set of observation features for maximal benefits in transfer.

Related Work

Recent work on learning policy priors has similar aspirations to our own (Wingate et al., 2011). This involves an MCMC-based policy search algorithm that learns priors over the

problem domain, which can be shared among states. For example, the method can discover the dominant direction in a navigation domain, or that there are sequences of motor primitives which are effective and should always be prioritised during search. Inference is on (π, θ) , where π is the policy and θ the parameters, by casting this search problem as approximate inference over which priors can be specified or learnt. Our work differs in that we do not assume a known model or kernel over policy space, and as such cannot sample from a generative model as is typical in Bayesian reinforcement learning.

Information from previous tasks can also be transferred as priors for a new task through model-based approaches. Sunmola (2013) proposes one such approach by maintaining a distribution over all possible transition models which could describe the current task and environment, and updating this belief every time an action is taken. Transfer is achieved by using the experience of state transitions in previous tasks to update beliefs when the agent first encounters each state in the new task, before anything is known about the transition probabilities from that state. Local feature models are also used to facilitate generalisation.

Sherstov and Stone (2005) also address transferring action preferences. In problems with large action sets, they try to either cut down the action set, or bias exploration in learning. The difference in this work is that the reduced action set, based on what they call the relevance of an action, is determined from the training data of optimal policies for the *entire* domain, rather than for each state or observation. This has the effect of pruning away actions that are always harmful throughout the domain, but the pruning is not context-specific.

Options (Precup, Sutton, and Singh, 1998) are a popular formalism of hierarchical reinforcement learning, and are defined as temporally extended actions with initiation sets where they can be invoked, and termination conditions. There are many approaches to learning these, see e.g. Pickett and Barto (2002). Although there are similarities between learning the initiation sets of options and action priors, they are distinct, in that an initiation set defines where the option *can physically be instantiated*, whereas an action prior describes regions where the option is *useful*. This is the same distinction that must be drawn between learning action priors and the preconditions for planning operators (e.g. Mourão et al. (2012)). For example, while pushing hard against a door may always be physically possible, this level of force would be damaging to a glass door, but that choice would not be ruled out by options or planning preconditions. Consequently, action priors not only augment preconditions, but are beneficial when using large sets of options or operators, in that they mitigate the negative impact of exploration with a large action set.

One approach to reusing experience is to decompose an environment or task into a set of subcomponents, learn optimal policies for these common elements through methods such as imitation learning (Argall et al., 2009) or apprenticeship learning (Abbeel and Ng, 2004; Rosman and Ramamoorthy, 2010), and then piece them together (Foster and Dayan, 2002), possibly applying transforms to make the

subcomponents more general (Ravindran and Barto, 2003). This is the philosophy largely taken by the options framework. Our method differs by discovering a subset of reasonable behaviours in each perceptual state, rather than one optimal policy. Our priors can thus be used for a variety of different tasks in the same domain, although the policy must still be learned. As a result, our method is also complementary to this decomposition approach.

The idea of using a single policy from a similar problem to guide exploration is a common one. For example, policy reuse (Fernandez and Veloso, 2006) involves maintaining a library of policies and using these to seed a new learning policy. The key assumption is that the task for one of the policies in the library is similar to the new task. While we acknowledge that this is a very sensible approach if the policies are indeed related, we are instead interested in extracting a more abstract level of information about the domain which is task independent, and thereby hopefully useful for any new task.

Other authors have explored incorporating a heuristic function into the action selection process to accelerate reinforcement learning (Bianchi, Ribeiro, and Costa, 2007), but this does not address the acquisition of these prior behaviours, is sensitive to the choice of values in the heuristic function, and requires setting additional parameters.

Action priors are related to the idea of learning *affordances* (Gibson, 1986), being action possibilities provided by some environment. These are commonly modelled as properties of objects, and can be learnt from experience (e.g. Sun et al. (2010)). The ambitions of action priors are however slightly different to that of affordances. As an example, learning affordances may equate to learning that a certain class of objects is “liftable” or “graspable” by a particular robot. We are instead interested in knowing how likely it is that lifting or grasping said object will be useful for the tasks this robot has been learning. Ideally, action priors should be applied over action sets which arise as the result of affordance learning, making these complementary concepts.

Conclusion

The problem of learning across multiple tasks is an important issue, as much knowledge gained during the learning of previous tasks can be abstracted and transferred to accelerate learning of new tasks. We address this problem by learning action priors from the policies arising from solving a collection of tasks. These describe context-specific distributions over the action set of the agent, based on which actions were used in different optimal policies under the same conditions.

We have shown that by learning priors over actions, an agent can improve performance in learning tasks in the same underlying domain. These priors are learned by extracting structure from the policies used to solve various tasks in the domain. By maintaining these distributions over the action space, exploration in learning new policies is guided towards behaviours that have been successful previously.

Our experiments show that this approach leads to faster learning, by guiding the agent away from selecting actions which were sub-optimal in other policies in the same domain. This approach essentially limits the branching fac-

tor caused by large action sets, and prunes the decision tree of options available to the agent during sequential decision making, based on successful behaviours in other tasks.

In future work we aim to apply this approach to knowledge transfer on real robotic systems, consisting of tasks with higher dimensionality and imperfect sensing, with the aid of perceptual templates for robustness. Additionally, we are investigating the application of action priors to other decision making paradigms, most notably planning.

Acknowledgements

The author gratefully acknowledges the anonymous reviewers for their helpful suggestions.

References

- Abbeel, P., and Ng, A. Y. 2004. Apprenticeship Learning via Inverse Reinforcement Learning. *International Conference on Machine Learning*.
- Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5):469–483.
- Bianchi, R. A. C.; Ribeiro, C. H. C.; and Costa, A. H. R. 2007. Heuristic Selection of Actions in Multiagent Reinforcement Learning. *International Joint Conference on Artificial Intelligence* 690–695.
- Dimitrakakis, C. 2006. Nearly optimal exploration-exploitation decision thresholds. In *Artificial Neural Networks–ICANN 2006*. Springer. 850–859.
- Fernandez, F., and Veloso, M. 2006. Probabilistic policy reuse in a reinforcement learning agent. *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*.
- Foster, D., and Dayan, P. 2002. Structure in the Space of Value Functions. *Machine Learning* 49:325–346.
- Gibson, J. J. 1986. *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Associates, Inc., 2nd edition.
- Harre, M.; Bossomaier, T.; and Snyder, A. 2012. The Perceptual Cues that Reshape Expert Reasoning. *Scientific Reports* 2(502).
- Huys, Q. J.; Eshel, N.; O’Nions, E.; Sheridan, L.; Dayan, P.; and Roiser, J. P. 2012. Bonsai trees in your head: how the Pavlovian system sculpts goal-directed choices by pruning decision trees. *PLoS computational biology* 8(3):e1002410.
- Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1-2):99–134.
- Konidaris, G. D., and Barto, A. G. 2006. Autonomous shaping: Knowledge transfer in reinforcement learning. *Proceedings of the 23rd International Conference on Machine Learning* 489–496.
- Mourão, K.; Zettlemoyer, L. S.; Petrick, R.; and Steedman, M. 2012. Learning strips operators from noisy and incomplete observations. *arXiv preprint arXiv:1210.4889*.
- Ortega, P. A., and Braun, D. A. 2013. Generalized thompson sampling for sequential decision-making and causal inference. *arXiv preprint arXiv:1303.4431*.
- Pickett, M., and Barto, A. G. 2002. PolicyBlocks: An Algorithm for Creating Useful Macro-Actions in Reinforcement Learning. *International Conference on Machine Learning* 506–513.
- Precup, D.; Sutton, R. S.; and Singh, S. 1998. Theoretical results on reinforcement learning with temporally abstract options. *European Conference on Machine Learning*.
- Ravindran, B., and Barto, A. G. 2003. Relativized Options: Choosing the Right Transformation. *Proceedings of the Twentieth International Conference on Machine Learning*.
- Rosman, B. S., and Ramamoorthy, S. 2010. A Game-Theoretic Procedure for Learning Hierarchically Structured Strategies. *IEEE International Conference on Robotics and Automation*.
- Rosman, B. S., and Ramamoorthy, S. 2012a. A Multitask Representation using Reusable Local Policy Templates. *AAAI Spring Symposium Series on Designing Intelligent Robots: Reintegrating AI*.
- Rosman, B. S., and Ramamoorthy, S. 2012b. What good are actions? Accelerating learning using learned action priors. *International Conference on Development and Learning and Epigenetic Robotics*.
- Rosman, B. S. 2014. *Learning Domain Abstractions for Long Lived Robots*. Ph.D. Dissertation, The University of Edinburgh.
- Sherstov, A. A., and Stone, P. 2005. Improving Action Selection in MDP’s via Knowledge Transfer. *AAAI* 1024–1029.
- Sun, J.; Moore, J. L.; Bobick, A.; and Reh, J. M. 2010. Learning Visual Object Categories for Robot Affordance Prediction. *The International Journal of Robotics Research* 29(2-3):174–197.
- Sunmola, F. T. 2013. *Optimising learning with transferable prior information*. Ph.D. Dissertation, University of Birmingham.
- Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.
- Taylor, M. E., and Stone, P. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research* 10:1633–1685.
- Thompson, W. R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 285–294.
- Watkins, C. J., and Dayan, P. 1992. Q-Learning. *Machine Learning* 8:279–292.
- Wingate, D.; Goodman, N. D.; Roy, D. M.; Kaelbling, L. P.; and Tenenbaum, J. B. 2011. Bayesian Policy Search with Policy Priors. *International Joint Conference on Artificial Intelligence*.
- Wyatt, J. 1997. *Exploration and inference in learning from reinforcement*. Ph.D. Dissertation, University of Edinburgh.