Feature Selection for Domain Knowledge Representation through Multitask Learning

Benjamin Rosman
Mobile Intelligent Autonomous Systems
CSIR, South Africa
BRosman@csir.co.za

***Abstract*—Representation learning is a difficult and important problem for autonomous agents. This paper presents an approach to automatic feature selection for a long-lived learning agent, which tackles the trade-off between a sparse feature set which cannot represent stimuli of interest, and rich feature sets which increase the dimensionality of the space and thus the difficulty of the learning problem. We focus on a multitask reinforcement learning setting, where the agent is learning domain knowledge in the form of behavioural invariances as action distributions which are independent of task specifications. Examining the change in entropy that occurs in these distributions after marginalising features provides an indicator of the importance of each feature. Interleaving this with policy learning yields an algorithm for automatically selecting features during online operation. We present experimental results in a simulated mobile manipulation environment which demonstrates the benefit of our approach.**

I. Introduction

Autonomously learning concise and useful representations of the world is one of the important open questions in artificial intelligence. The representation scheme used by an agent has major ramifications for not only its capabilities, but also the computation time required for solving problems. Despite this fact, most autonomous agents use hand-crafted and carefully engineered feature sets. This approach presents major drawbacks, in both being costly for a human to produce and is thus not scalable, and also inherently biased towards the sensing and experience of the human designer, which may be suboptimal for the artificial agent.

In this paper we are interested in the problem of a learning agent being able to select its own features for representing domain knowledge. This does not solve the entire problem of representation learning, but relieves a human designer from the complete task of feature engineering. Instead, the human need only specify a large set of features, and the agent can then select the useful ones from this set.

It is beneficial to not use more features than necessary, particularly if the agent is learning behaviours or dynamics models. A larger feature set implies a higher dimensional representation, inducing sparsity in the experiences of the agent, and complicating learning through the curse of dimensionality.

Our goal is to have a concise representation of behavioural domain knowledge which can be used to facilitate faster completion and learning of future tasks. This can be used to learn templates coupling observations to local behaviour patterns [1], [2]. These local patterns could then be used as the basis for stimulus based reactive control schemes [3].

The simplest way for an agent to learn an appropriate representation of the domain is through its own experience. However, learning from a single task is likely to overfit to that task, and so instead we focus on a multitask setting. Our approach is thus to present an agent with a collection of different tasks, which it solves using a standard reinforcement learning (RL) method. Each task then provides the agent with an additional optimal policy which can be used to refine its representation and model of the domain.

For this purpose we define a task as a Markov decision process (MDP). We use a recent formalism, *action priors* [4], to encode general domain knowledge. These are observation-based distributions over the action set of the agent, learnt from a collection of policies, which represent the task-independent probability of an action being optimal in a particular context. Through these local distributions, we analyse the impact of each feature on the agent's behaviour, and in doing so prune away the least important features.

Our proposed approach seeks to reduce the total entropy in the action prior distributions. The intuition is that the state representation which provides the most informative (peaked) distributions over the action set, given a set of previous optimal policies, is the representation which provides the agent with the most decisive model of behaviour in the domain.

Feature selection is an important and long-standing question in the operation of autonomous and learning agents and has been studied in many different contexts [5]. This issue has commonly arisen in the construction of basis function approximations, typically of continuous value functions in RL [6]. Notable approaches for feature selection are through regularisation [7] and incrementally building a basis to reduce the Bellman error [8]. Furthermore, if states have factored representations then conditional independencies between features can be extracted from a dynamic Bayesian network [9].

In relational contexts, the relevance of an object can be decided based on whether that object affects the outcomes or rewards of actions [10], or deictic references can be used to only refer to objects needed in the pre- or post-conditions of actions [11]. This has a similar effect to feature selection.

More generally, the importance of features can be determined by studying properties of solution trajectories. For

example, a feature may be considered irrelevant if it does not affect the policy passing through any state [12], or if there is little conditional mutual information between the return and that feature [13]. Alternatively, the importance of a feature set can also be ascertained by evaluating the performance of classifiers trained on different feature sets [14].

We instead leverage the fact that we are operating in a multitask setting, where feature selection is required for minimising the dimensionality of a domain model. In this way, our novel contribution is to perform feature selection based on commonalities between multiple policies for different tasks in the same domain.

This paper is structured as follows. We first present some reinforcement learning preliminaries in Section II. The learning and using of action priors is then described in Section III. Our approaches to offline and online feature selection using action priors appear in Section IV. Finally, we demonstrate our methods empirically in Section V.

II. Preliminaries

In keeping with the standard formalism of reinforcement learning (RL), an environment is specified by a Markov Decision Process (MDP): (S, A, T, R, γ), for S a finite set of states, A a finite set of actions which can be taken by the agent, $T : S \times A \times S \to [0, 1]$ the state transition function where $T(s, a, s')$ gives the probability of transitioning from state s to state s' after taking action a, $R : S \times A \to \mathcal{R}$ the reward function, where $R(s, a)$ is the reward received when taking action a in state s, and $\gamma \in [0, 1]$ a discount factor.

Now define a domain $D = (S, A, T, \gamma)$, and a task as the MDP $\tau = (D, R)$. This factorises the environment such that the state set, action set and transition functions are fixed for the domain, and each task varies only in the reward function.

A policy $\pi : S \times A \to [0, 1]$ for an MDP is a probability distribution over state-action space. 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 at step k. The goal of an RL 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 [15]. Q is typically learnt by iteratively updating values using the rewards obtained by simulating trajectories through state space, e.g. Q-learning [16]. A greedy policy can be obtained from a value function defined over $S \times A$, by selecting the action with the highest value for the given state.

Let a feature f_i be a mapping from the current state of the agent to a set of values $\{f_i^1 \cdots f_i^{K_i}\}$. Let ϕ be a set of these features. We now abuse notation slightly and for a particular feature set ϕ_i we enumerate the possible settings of all its constituent features, such that $\phi_i = \phi_i^j$ means that the features in ϕ_i are set to configuration j, where these configurations are uniquely ordered such that $j \in [1, K_{\phi_i}]$, where $K_{\phi_i} = \prod_q K_q$, q runs over the features of ϕ_i, and K_q is the number of settings for feature q. Observations at a state s are then a vector given by $\phi(s)$. Except where ambiguous, we use ϕ to refer to $\phi(s)$.

III. Priors over Action Selection

A set of tasks, defined as MDPs, are assumed to exist in a single domain $\mathcal{D}$, which provides common infrastructure, relating the tasks. We now seek to learn a model of domain knowledge, and in particular a model of "normal" behaviour in the domain. This is learnt from multiple tasks, giving the domain's *action priors* [4]. It is the representation of this model for which we select features.

For an unknown but arbitrary set of tasks $\mathcal{T} = \{\tau\}$, with different reward functions and corresponding optimal policies $\Pi = \{\pi_\tau\}$, we learn for each observation ϕ a distribution $\theta_\phi(A)$ over the action set A. This distribution is the action prior, and represents the probability of each action in A being used by a solution policy traversing through a state s with observations $\phi(s)$, aggregated over all states giving the same observations, and tasks $\mathcal{T}$. These action priors model the local invariant effects of the transition functions drawn from $\mathcal{D}$.

From a set of policies, each selecting actions in the same state s according to different distributions, a state-based model is learnt of typical behaviour marginalised over all known tasks in the domain. Thus, given a state s, if one action is favoured by all trajectories through s, then that action should be preferred by any new trajectory exploring through s. Conversely, any action which is not selected by any trajectory passing through s is likely to have negative consequences, and should be avoided. By studying the policies from multiple tasks, a model of the structure of the underlying domain can be learnt, in terms of identifying the set of local behaviours which are invariant across all tasks in the domain.

A. Learning Action Priors

Consider a setting in which multiple tasks have been solved in $\mathcal{D}$, each providing an optimal solution policy. For each observation $\phi \in \Phi$, we model the action priors $\theta_\phi(a), \forall a \in A$ using a Dirichlet distribution, by maintaining a count $\alpha_\phi(a)$ for each action $a \in A$. The initial values of $\alpha_\phi(a) = \alpha_\phi^0(a)$ are the hyperprior, initialised to any value (zero in this case).

The αs are learnt by the agent from previous behaviours [4], and updated for each $(\phi(s), a)$ for a new policy π as

$$\alpha_{\phi(s)}^{t+1}(a) \leftarrow \begin{cases} \alpha_{\phi(s)}^t(a) + 1 & \text{if } \pi(s, a) = \max_{a'} \pi(s, a') \\ \alpha_{\phi(s)}^t(a) & \text{otherwise.} \end{cases} \quad (1)$$

$\alpha_s(a)$ reflects the number of times a was considered an optimal choice of action in state s by *any* policy, added to the hyperprior priors $\alpha_s^0(a)$.

The closed form of Equation (1) given policy set Π is

$$\alpha_{\phi(s)}(a) = \sum_{s \in [s]_\phi} \sum_{\pi \in \Pi} U_{\phi(s)}^\pi(a) + \alpha_{\phi(s)}^0(a), \quad (2)$$

where $U_{\phi(s)}^\pi(a) = \delta[\pi(\phi(s), a), \max_{a' \in A} \pi(\phi(s), a')]$ for $\delta[\cdot, \cdot]$ the Kronecker delta function, and $[s]_\phi = \{s' \in S | \phi(s) = \phi(s')\}$ is the equivalence class of all states with the same observation features as state s. This additional summation

occurs because in the general case, the priors from multiple states will map to the same observation based action priors.

To obtain the action priors $\theta_\phi(A)$, sample from the Dirichlet distribution: $\theta_\phi(A) \sim \text{Dir}(\alpha_\phi)$. An action a is selected by sampling from the action prior: $a \sim \theta_\phi(A)$.

B. Using the Action Priors

Action priors provide knowledge about which actions are sensible in situations in which there are several choices, and so are useful for seeding search in a policy learning process. We demonstrate this as an exploration process with an adaptation of traditional Q-learning [15], called ϵ-greedy Q-learning with Perception-based Action Priors (ϵ-QPAP) [4], which is shown in Algorithm 1. Note, in this algorithm, $\alpha^Q \in [0, 1]$ denotes the learning rate, and should not be confused with the Dirichlet distribution counts $\alpha_\phi(a)$. The parameter $\epsilon \in [0, 1]$ controls the trade-off between exploration and exploitation. Both α^Q and ϵ are typically annealed after each episode.

Algorithm 1 ϵ-greedy Q-learning with Perception-based Action Priors (ϵ-QPAP)

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

The difference between this and standard ϵ-greedy Q-learning can be seen in the action selection step (line 6), consisting of two cases. The first deals with exploiting the current policy stored in $Q(s, a)$ with probability $1-\epsilon$, and the second 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 $\theta_\phi(a)$ to shape the action selection based on what were sensible choices in the past.

The effect 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. This highlights the assumption that there is inherent structure in the domain which can be identified across multiple tasks.

IV. Feature Selection

A. Offline Feature Selection

The choice of observational features f is an open question, depending on the capabilities of the agent. Possible features include the state label (complete state description), or any sensory information the agent may receive from the environment. Furthermore, these features can include aspects of the task description, or recent rewards received from the environment.

As a result, in many domains, there could be a large set of observational features. The size of Φ, the space of possible mappings, is exponential in the number of features. We are interested in identifying the optimal feature set $\phi^* \in \Phi$, which provides abstraction and dimensionality reduction, with a minimal loss of information in the action prior. Finding such a ϕ^* allows for the decomposition of the domain into a set of *capabilities* [2], being recognisable and repeated observational contexts, with minimal uncertainty in the optimal behavioural responses, over the full set of tasks.

We wish to find a feature set which can prescribe actions with the most certainty. To this end, we define the average entropy of an action a for a particular feature set ϕ as

$$\bar{H}_\phi(a) \quad = \quad \sum_{i=1}^{K_\phi} P(\phi^i) H[P(a|\phi^i)], \tag{3}$$

where $P(a|\phi^i) = \theta_{\phi^i}(a)$ is the action prior for a particular set of feature values, $P(\phi^i)$ is the prior probability of those feature values, estimated empirically from the data as $\frac{\sum_a \alpha_{\phi^i}(a)}{\sum_i \sum_a \alpha_{\phi^i}(a)}$, and $H[p] = -p \log_2 p$ is the standard entropy. The prior $P(\phi^i)$ serves to weight each component distribution by the probability of that feature combination arising in the data.

By summing the average entropy for each action, we define the entropy of the action set A for a feature set ϕ as

$$H_\phi \quad = \quad \sum_{a \in A} \bar{H}_\phi(a) \tag{4}$$

$$= -\sum_{a \in A} \sum_{i=1}^{K_\phi} \frac{\sum_{a' \in A} \alpha_{\phi^i}(a')}{\sum_{j=1}^{K_\phi} \sum_{a' \in A} \alpha_{\phi^j}(a')} \theta_{\phi^i}(a) \log_2 \theta_{\phi^i}(a). \tag{5}$$

The optimal feature set is the feature set which minimises the action set entropy. In this way, what we seek is analogous to the information invariance which is present in the observations of the agent [17]. The problem with this minimisation is that the number of possible feature sets is exponential in the number of features. We therefore present a method, shown in Algorithm 2, which returns an approximate minimal feature mapping $\tilde{\phi}^*$. The key assumption of this approximation is that each feature f affects the entropy of $\theta_\phi(a)$ independently, and so we need only select the set of features that each individually decreases the total entropy by more than some amount.

For each feature f from the full feature set ϕ_{full}, we marginalise over that feature and compute the entropy of the remaining feature set. Each of these $\|\phi_{full}\|$ individual entropy values is compared to the entropy of the full set $H_{\phi_{full}}$. The

greater the increase in entropy resulting from the removal of feature f, the more important f is as a distinguishing feature in the action prior, as that feature reduces the overall entropy of the action prior distribution. The feature set chosen is thus the set of all features which, when removed, would result in an entropy increase greater than a threshold ω.

Algorithm 2 Independent Feature Selection

Require: feature set ϕ_{full}, entropy increase threshold ω
1: Compute $H_{\phi_{full}}$ by Equation (5)
2: **for** every feature f in ϕ_{full} **do**
3: $\quad \phi_{-f} \longleftarrow \phi_{full} \setminus f$
4: $\quad$ Compute $H_{\phi_{-f}}$ by Equation (5)
5: $\quad \Delta_f \longleftarrow H_{\phi_{-f}} - H_{\phi_{full}}$
6: **end for**
7: $\tilde{\phi}^* \longleftarrow \{f \in \phi_{full} | \Delta_f \geq \omega\}$
8: **return** $\tilde{\phi}^*$

If there is a high entropy in which actions should be taken in the context of a particular observation, then there are two possible reasons for this: 1) it may be the case that different tasks use this context differently, and so without conditioning action selection on the current task, there is no clear bias on which action to select, or 2) it may be that this observation is not informative enough to make the decision, providing scope for feature learning. This distinction can only be made in comparison to using the maximal feature set, as the most informative set of observation features. Without ground truth state information, this can only be ascertained through learning. If the observations are not informative enough, then this suggests that additional features would be useful. This provides the agent with the opportunity for trying to acquire new features.

B. Online Feature Selection

Algorithm 3 describes a complete process for performing feature selection on an online agent. The agent is repeatedly presented with a new task, solves that task, and uses the new policy to refine its feature set.

Algorithm 3 Online Feature Selection

1: Let ϕ_{full} be the full feature set
2: Initialise full action prior θ^0_{full}
3: $\theta^0 \longleftarrow \theta^0_{full}$
4: **for** every new task $t = 1, 2, \ldots$ **do**
5: $\quad$ Learn policy π^t using prior θ^{t-1} and Algorithm 1
6: $\quad$ Update θ^t_{full} using θ^{t-1}_{full} and π^t with Eq. (1)
7: $\quad$ Select features ϕ^t from θ^t_{full} using Algorithm 2
8: $\quad$ Extract θ^t from θ^t_{full}, marginalising over $f \in \phi_{full} \setminus \phi^t$
9: **end for**

Given a new task, the algorithm executes four steps. First, a policy is learnt using the current action prior and Algorithm 1. This new policy is then used to update the full prior. From the full prior, select features using Algorithm 2. Finally, extract the action prior using the selected features by means of marginalising over the excluded features.

As the feature selection is done using Algorithm 2, this requires a choice of ω. This parameter is domain specific, but in our experiments we automate its selection as the mean of the set of Δ_f values, as computed in Algorithm 2.

V. Experiments

A. The Factory Domain

The factory domain is an extended navigation domain that involves a mobile manipulator robot placed on a factory floor. The layout of the factory consists of an arrangement of walls through which the robot cannot move, with some procurement and assembly points placed around the factory. The factory layout changes between tasks (corresponding to different transition functions T), to simulate the effect of equipment being moved, or the robot working in several different buildings. Examples of the domain are shown in Figure 1.

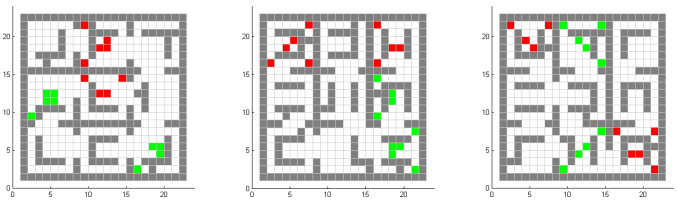

Fig. 1. Examples of the factory domain. Grey cells are obstacles, white cells are free space, green cells are procurement points, and red cells are assembly points.

The robot's action set consists of four movement actions (*North*, *South*, *East* and *West*), each of which will move it in the desired direction provided there is no wall in the destination 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* used at assembly point i constructs component i, provided the robot possesses the materials required for component i. A task is defined as a list of components which must be assembled by the robot. The state describes the position of the robot and the state of assembly of each required component.

The rewards are defined as follows. All movement actions give a reward of -1. Collisions are damaging 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. The learning algorithm parameters were initialised as $\gamma = 0.99$, $\epsilon = 0.9$ and $\alpha^Q = 0.5$.

B. Effect of Different Feature Sets

Figure 2 demonstrates the performance of agents learning using action priors with different feature sets. Note that

uniform priors equate to not using action priors (standard Q-learning). Figure 2 shows the effect of four feature sets:

- ϕ_1: Two features: the terrain type occupied by the agent (in $\{free, wall, procure\text{-}point, assembly\text{-}point\}$), and a ternary flag indicating whether any items need to be procured or assembled.
- ϕ_2: Four features: the types of terrain of the cells adjacent to the cell occupied by the agent.
- ϕ_3: Six features: the types of terrain of the cell occupied by the agent as well as the cells adjacent to that, and a ternary flag indicating whether any items need to be procured or assembled. Note that $\phi_3 = \phi_1 \cup \phi_2$.
- 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.

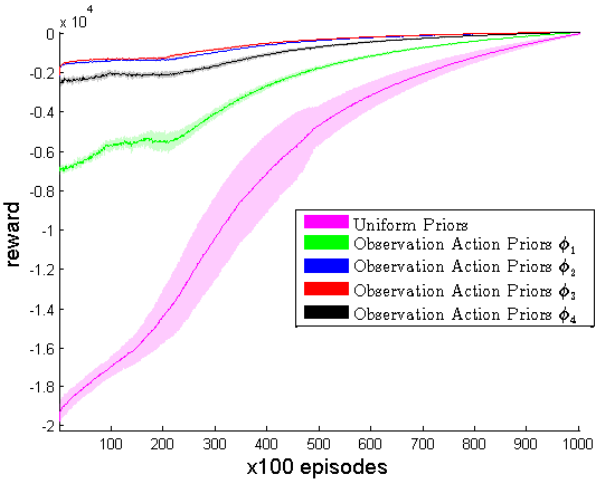

Fig. 2. Comparative performance of Q-learning with uniform priors and four different observation based action priors: ϕ_1, ϕ_2, ϕ_3 and ϕ_4. The curves show average reward per episode averaged over 10 runs, where the task was to assemble 4 random components. In each case the prior was obtained from 80 training policies. The shaded region represents one standard deviation.

As can be seen, these four observation priors all contain information relevant to the domain, as they all provide an improvement over the baseline. There is however a significant performance difference between the four feature sets.

Surprisingly, Figure 2 shows the most beneficial feature sets are ϕ_2 and ϕ_3. The fact that the richest feature set, ϕ_4, did not outperform the others seems counterintuitive. The reason for this is that using these 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. This illustrates the trade-off between a sparse feature set which cannot capture the required information, and a rich feature set which requires more time for learning.

We identify the set of the most useful features using Algorithm 2. These results, given in Figure 3, show that the relative importance for the ten features (all of which are present in ϕ_4) are consistent across the four feature sets. As may be expected, the ϕ_4 results indicate that the values of the cells diagonally adjacent to the current cell occupied by the agent are relatively unimportant, as they are at best two steps away from the agent.

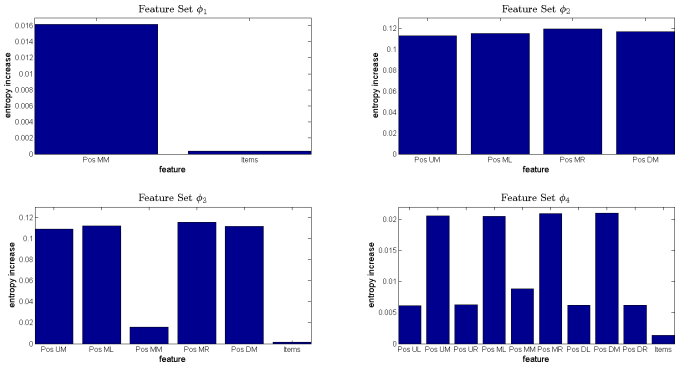

Fig. 3. Feature importance of each feature in the four different observation based action priors: ϕ_1, ϕ_2, ϕ_3 and ϕ_4. The spatial features are labeled $PosYX$, with $Y \in \{(U)p, (M)iddle, (D)own\}$ and $X \in \{(L)eft, (M)iddle, (R)ight\}$. The feature $Items$ is a ternary flag, indicating if the agent still needs to assemble or procure any items.

What is surprising at first glance is that neither the value of the cell occupied by the agent, nor the current state of the assembly carried by the agent are considered relevant. Consider the current state of assembly: the feature *items* is actually already a very coarse variable, which only tells the agent that either a procurement or an assembly is required next. This is very local information, and directly affects only a small handful of the actions taken by the agent. Now consider the cell currently occupied by the agent. This indicates whether the agent is situated above an assembly or procurement point. Again, this is only useful in a small number of scenarios. Note that these features are still useful, as shown by the performance of ϕ_1 relative to uniform priors.

What turns out to be the most useful information is the contents of the cells to the North, South, East and West of the current location of the agent. These provide two critical pieces of information to the agent. Firstly, they mitigate the negative effects that would be incurred by moving into a location occupied by a wall. Secondly, they encourage movement towards procurement and assembly points. These then constitute the most valuable features considered in our feature sets. This observation is confirmed by the fact that ϕ_2 and ϕ_3 are near identical in performance.

C. Online Feature Selection

We now demonstrate the performance of the adaptive priors generated through the use of online feature selection in Algorithm 3. This is shown in Figure 4. Note that in the first episode of learning, the performance of the agent with feature selection is actually slightly better than that of the agent with the full prior, again showing the cost incurred through the use of an overly rich feature set. However, this (minor) improvement comes with a considerable decrease in the number of possible

observational templates (discussed below). The performance of both is considerably better than not using a prior. However, at convergence, all three methods achieve the same performance (shown by the three lines at the top of the figure).

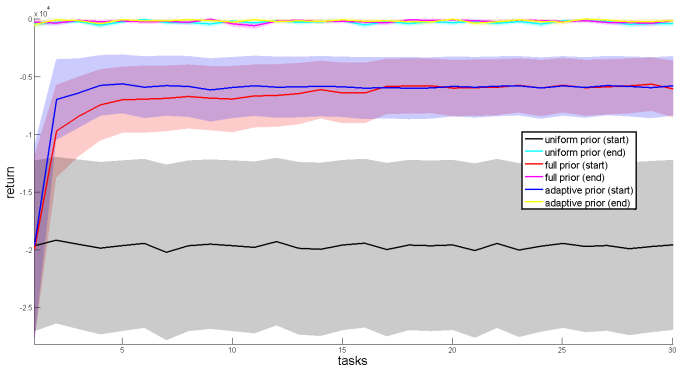

Fig. 4. Performance of learning agent during first (start) and 30^{th} (end) episodes, over thirty tasks. Results compare uniform priors, the full set of 10 features, and the adaptive prior of online feature selection. Results are averaged over 10 runs. The shaded region represents one standard deviation.

Figure 5 shows the features selected by the algorithm as a function of the number of tasks experienced. The result is also a considerable reduction in feature space descriptions. The algorithm is seen to converge to the four features used in ϕ_2.

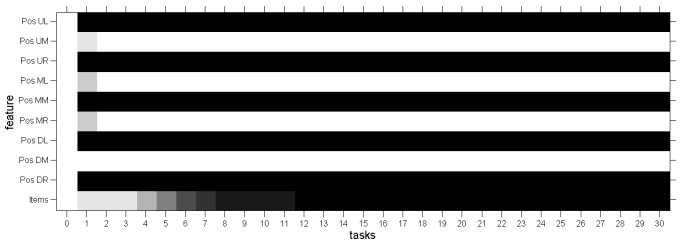

Fig. 5. Features selected during online feature selection over 30 tasks, averaged over 10 runs. The brightness of each square indicates the percentage of runs for which that feature was selected. The spatial features are labeled $PosYX$, with $Y \in \{(U)p, (M)iddle, (D)own\}$ and $X \in \{(L)eft, (M)iddle, (R)ight\}$. The feature $Items$ is a flag, indicating if the agent still needs to assemble or procure any items.

The effect is that a reduced set of four features is shown to be preferable to the initial full set of ten which were proposed for this domain in ϕ_4. The benefit is that the number of possible feature configurations has been reduced from $4^9 \times 3$ down to 256: requiring about 0.033% of the storage space (observational templates). Learning behaviours for these configurations is far simpler, and provides the agent with a conveniently small repertoire of useful local behaviours.

VI. Conclusion

In this paper, we presented a novel approach for using experience from multiple tasks to select observation features for a concise representation of domain knowledge. This algorithm frees a system designer from providing an agent with the best representation scheme for the domain and tasks with which it will operate, and instead allows the agent to select these features itself over the course of its operational lifetime.

This automatic selection mechanism negotiates the problems of having a feature set which is either too sparse (meaning the agent cannot adequately describe the tasks at hand) or too rich (which, thanks to the curse of dimensionality, requires considerable time to learn and encounter each possible setting).

By reducing the dimensionality of the feature space, the agent reduces the number of configurations which need to be learnt in a domain. This provides the agent with a concise set of scenarios which describe local behaviour, and thus domain knowledge. The result is that this approach can be used for reactive control, where selecting the feature sets corresponds to learning a set of stimuli, and the action priors provide appropriate local controllers.

Acknowledgements

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

References

[1] M. Stolle and C. G. Atkeson, "Knowledge transfer using local features," *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 26–31, 2007.

[2] B. S. Rosman and S. Ramamoorthy, "A Multitask Representation using Reusable Local Policy Templates," *AAAI Spring Symposium Series on Designing Intelligent Robots: Reintegrating AI*, 2012.

[3] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.

[4] B. S. Rosman and S. Ramamoorthy, "What good are actions? Accelerating learning using learned action priors," *International Conference on Development and Learning and Epigenetic Robotics*, November 2012.

[5] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[6] G. Konidaris, S. Osentoski, and P. S. Thomas, "Value function approximation in reinforcement learning using the fourier basis." in *AAAI*, 2011.

[7] J. Z. Kolter and A. Y. Ng, "Regularization and feature selection in least-squares temporal difference learning," pp. 521–528, 2009.

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

[9] M. Kroon and S. Whiteson, "Automatic feature selection for model-based reinforcement learning in factored mdps," in *Machine Learning and Applications, 2009. ICMLA'09. International Conference on*. IEEE, 2009, pp. 324–330.

[10] T. Lang and M. Toussaint, "Relevance Grounding for Planning in Relational Domains," *European Conference on Machine Learning*, 2009.

[11] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling, "Learning symbolic models of stochastic domains," *Journal of Artificial Intelligence Research*, vol. 29, no. 1, pp. 309–352, May 2007.

[12] N. K. Jong and P. Stone, "State Abstraction Discovery from Irrelevant State Variables," *International Joint Conference on Artificial Intelligence*, pp. 752–757, August 2005.

[13] H. Hachiya and M. Sugiyama, "Feature selection for reinforcement learning: Evaluating implicit state-reward dependency via conditional mutual information," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 474–489.

[14] C. Diuk, L. Li, and B. R. Leffler, "The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 249–256.

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

[16] C. J. Watkins and P. Dayan, "Q-Learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[17] B. R. Donald, "On information invariants in robotics," *Artificial Intelligence*, vol. 72, no. 1, pp. 217–304, 1995.