JMLR: Workshop and Conference Proceedings 27:217–227, 2012 Workshop on Unsupervised and Transfer Learning

Transfer Learning in Sequential Decision Problems: A Hierarchical Bayesian Approach

Aaron Wilson WILSONAA@EECS.OREGONSTATE.EDU
Alan Fern AFERN@EECS.OREGONSTATE.EDU
Prasad Tadepalli TADEPALL@EECS.OREGONSTATE.EDU
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331

Editor: I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver

Abstract

Transfer learning is one way to close the gap between the apparent speed of human learning and the relatively slow pace of learning by machines. Transfer is doubly beneficial in reinforcement learning where the agent not only needs to generalize from sparse experience, but also needs to efficiently explore. In this paper, we show that the hierarchical Bayesian framework can be readily adapted to sequential decision problems and provides a natural formalization of transfer learning. Using our framework, we produce empirical results in a simple colored maze domain and a complex real-time strategy game. The results show that our Hierarchical Bayesian Transfer framework significantly improves learning speed when tasks are hierarchically related.

Keywords: Reinforcement Learning, Markov Decision Processes, Transfer Learning, Hierarchical Bayesian Framework

1. Introduction

The goal of the current paper is to explain transfer in sequential decision-making domains using the hierarchical Bayesian framework. Unlike the standard supervised setting, in sequential decision making, the training examples are not independently identically distributed. When learning behaviors from demonstrations, examples consist of states and desired action sequences, which is similar to the supervised setting. However, in the more typical reinforcement learning setting, the target action sequences are not provided. The learner must explore action sequences to determine which are are the most rewarding. Prior knowledge informing the agent's exploration strategy would considerably enhance the speed of convergence. Transfer is especially attractive in reinforcement learning since both the discovery of good policies and their generalization would be potentially impacted by learning from related domains.

There are opportunities to learn different kinds of knowledge through transfer in reinforcement learning. We describe two kinds of knowledge transfer: transfer of domain model knowledge, and transfer of policy knowledge. In the *model-based* approach, model learning is improved by incorporating an informative hierarchical Bayesian prior learned from experience in early tasks. The prior represents knowledge about the world. In the *policy search* approach, a prior is placed on the policy parameters themselves. The prior represents direct knowledge of what the agent should do. In each approach, it is assumed that there is a small but unknown number of component types which can be learned and transferred to new tasks. When the discovered components can be reused Transfer is successful.

2. Sequential Decision Problems and Hierarchical Transfer Hypothesis

We formulate the transfer learning problem in sequential decision making domains using the framework of Markov Decision Processes (MDPs). An MDP M is defined by a 6-tuple (S, A, C, T, I, F), where S is a set of states and A is a set of actions. $C(s, a)$ is the immediate cost of executing action a in state s. Function $T(s, a, s')$ defines the probability of reaching state s' given that action a is taken in state s. I is a distribution of initial states and F is a set of final or terminating states. A policy π for M is a mapping from S to A. The expected cost of a policy π starting from a state s is the cumulative sum of action costs starting from state s, and ending when the agent reaches a terminating state. A solution to an MDP is an optimal policy that minimizes the total expected cost from all states. The minimum expected cost from state s is the unique solution to the following Bellman equations:

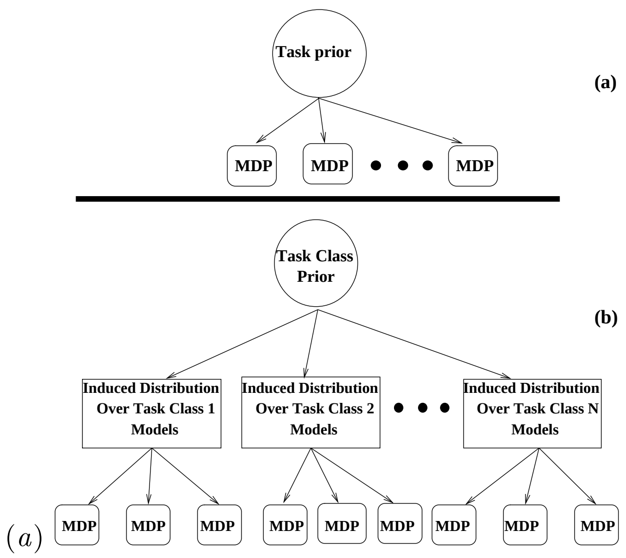
$$V^*(s) = \begin{cases} 0 \text{ if } s \text{ is a terminal state} \\ \min_a C(s,a) + \sum_{s'} (T(s,a,s')V^*(s')), \text{ else} \end{cases}$$

The actions that minimize the right hand side of this equation constitute the optimal policy.

While most work in RL is concerned with solving a single unknown MDP (task), here we are concerned with transfer across MDPs. Our objective is to develop a lifelong learning algorithm that is able to leverage experience in previous MDPs $M_1, \ldots M_n$ to more quickly learn an optimal policy in a newly drawn MDP M_{n+1}. If each task faced by an agent is arbitrarily different from the previous tasks, the agent has no way to benefit from having solved the previous tasks. To explain the apparently fast learning of people, and duplicate this learning speed in machines, we explicitly formulate the *hierarchical transfer hypothesis*. It is the claim that tasks have a latent hierarchical relationship, which can be explicated and exploited to transfer knowledge between tasks. We propose a hierarchical Bayesian framework to capture the task similarities. We apply this framework to both model-based RL and to model-free policy search. In model-based RL, the hierarchy represents similarities between MDP models. In the policy search approach, the hierarchy represents similarities between policies. Below we discuss each of these approaches in more detail.

3. Hierarchical Model-Based Transfer

In this section, we outline our hierarchical Bayesian approach to model-based transfer for RL problems. Figure 1(a)(bottom) illustrates our fundamental assumptions of the task generation process. Classes of MDPs are generated from a task prior distribution, and each task generates an MDP. The hierarchical structure assumes that each MDP M_i is an iid sample from one of the unobserved classes. We propose a hierarchical prior distribution, based on the Dirichlet Process (DP), in order to capture this hierarchical structure. We aim to exploit this structure when exploring new tasks.


Task prior
MDP
MDP
MDP
(a)
Task Class Prior
(b)
Induced Distribution Over Task Class 1 Models
Induced Distribution Over Task Class 2 Models
Induced Distribution Over Task Class N Models
MDP
(a)

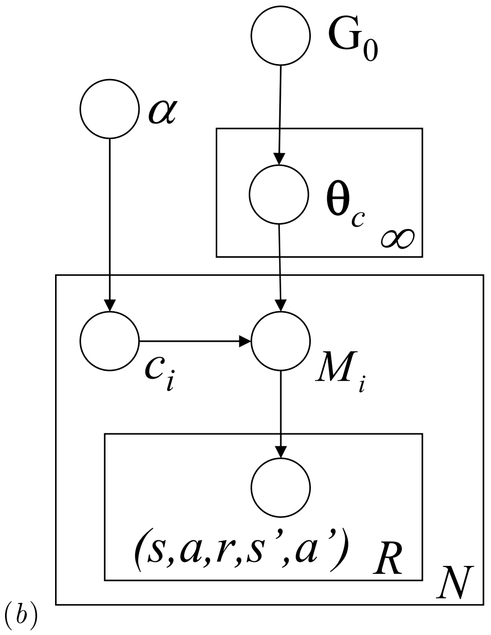


Figure 1: (a) (top) Single task Bayesian RL vs. (bottom) Hierarchical Model-Based Transfer. The number N and parameters of the induced distributions are learned from data. (b) The corresponding Infinite Mixture Model (Parameter set Ψ). The task prior is represented by G_0, task class distributions have parameters θ, classes are indexed by c, M_i represents the i^{th} MDP model parameters, the tuples (s, a, r, s', a') are the observed transition and reward data, and α is the Dirichlet Process concentration parameter.

We describe the generative process for MDP models using Figure 1(b). The generative model is shown in plate notation. Rectangles indicate probability distributions that are replicated a certain number of times. The number of replications is shown in the bottom-right corner of each plate. The figure shows a distribution over N MDPs. In each MDP the agent has made R observations. Each class $C = c$ is associated with a vector of parameters θ_c. These parameter vectors define distributions from which individual MDPs will be drawn. The parameters c_i indicate the class of MDP M_i. G_0 is a distribution over classes that corresponds to the Task Class Prior in Figure 1(a)(bottom). Finally the parameter α is the concentration parameter of the Dirichlet Process (DP) model. Ψ refers to the tuple of parameters $\Psi = (\theta, c, \alpha)$. The DP generalizes the case of a known finite number of MDP classes to a infinite number of classes. This non-parametric model allows us to (a) model a potentially unbounded number of classes, (b) adapt to and make inferences about classes we have not seen before, and (c) learn the shared structure that constitutes each class, rather than predefining class specifications.

The key steps in our approach to exploiting this model are outlined in Algorithm 1. The Hierarchical DP model is used as a prior for Bayesian RL in a new MDP. Before any MDPs are experienced, the hierarchical model parameters Ψ are initialized to uninformed values. The Sample function (line 5) uses the DP model to generate a sample of several MDPs from

Algorithm 1 Hierarchical Model-Based Transfer Algorithm HMBT

```
1:  Initialize the hierarchical model parameters $\Psi$
2:  for each MDP $M_i$ from $i = 1, 2, \ldots$ do
3:      $O_i = \emptyset$ //Initialize the set of observation tuples (s,a,r,s',a') for $M_i$
4:      while policy $\pi$ has not converged do
5:          $\hat{M}_i \leftarrow$ Sample($P(M \mid O_i, \Psi)$)
6:          $\pi =$ Solve($\hat{M}_i$) //Solve the sampled MDP using value iteration.
7:          Run $\pi$ in $M_i$ for $k$ steps.
8:          $O_i = O_i \cup \{$observations from $k$ steps$\}$
9:      end while
10:     $\Psi \leftarrow$ Sample($P(\Psi \mid \hat{M}_1, \ldots, \hat{M}_i)$) //Given fixed estimates of the MDP model param-
        eters generate samples from the posterior distribution and select the best.
11: end for
```

$\Pr(M_i \mid O_i, \Psi)$. From this sample, the $\hat{M}_i$ with the highest probability is selected. To select an action we solve $\hat{M}_i$ using value iteration (line 6) and then follow the greedy policy for k steps (line 7). Observations gathered while executing the greedy policy are added to the collection of observation tuples O_i for the current MDP. Next, the posterior distribution is updated, and a new sample of MDPs is taken from the updated posterior distribution. This loop is repeated for MDP M_i until the policy has converged. After convergence, we update the hierarchical model parameters Ψ based on the parameter estimates $\hat{M}_i$. This is done using the SampleMap procedure. SampleMap uses the machinery of the DP to assign (possibly new) classes to each observed MDP$\hat{M}_1, \ldots, \hat{M}_i$. Given the assignments, we compute the parameters associated with each class distribution. This process is iterated until convergence. Intuitively, as the agent gains experience with more tasks, the samples generated by this procedure will tend to represent the hierarchical class structures generating the MDPs. Learning speed will improve if a new task is usefully described by the sampled hierarchical structure. Details for this algorithm, including specifications of the posterior sampling algorithm, are found in Wilson et al. (2007).

Experimental Results. To test our hypotheses, we consider a simple colored maze domain. An example maze is shown in Figure 2. The goal of the agent is to traverse the least cost path from the start location to the final goal location. To do so, the agent may navigate to adjacent squares by executing one of four directional actions. The agent's reward function is a linear function combining the costs of all squares adjacent to the agent's location on the map. The contributed cost of each adjacent square is a function of its color. To do well, the agent must determine which colors have low cost and identify a path minimizing the total cost from start to finish. In the multi-task setting the parameters of the reward function, governing the penalty for traversing certain colors, characterize each task, and are assumed to be generated from a hierarchical structure. To generate a task, the world first generates a class and then generates a vector of color costs from the sampled class. Each map is randomly colored before the agent begins its interaction.

In the first instance of this domain the goal location is fixed. The agent must traverse from the top left to the bottom right of the map. The objective is to maximize the total reward per episode. Episodes end only after the goal is reached. Map squares have 8

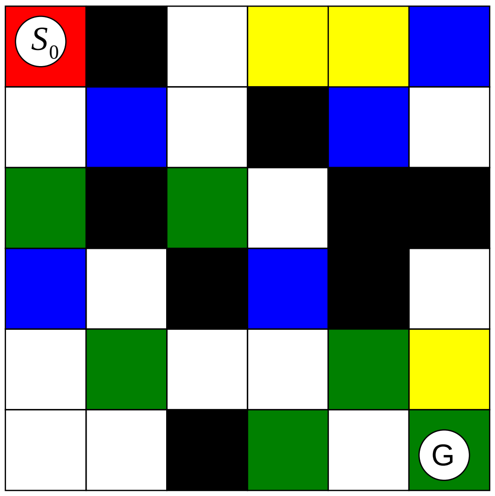


Figure 2: The colored map domain. Tile squares are randomly colored. Each color has a cost. When the agent lands on a square it receives a penalty proportional to the cost of the occupied square and the costs of adjacent squares. The agent's goal is to traverse the least cost path from the top left to the bottom right.

different colors and reward functions are generated from 4 distinct classes. Figure 3 shows results for this setting. Good performance is possible with a small number of steps in each sampled MDP. Even so the benefit of transfer is obvious. Given experience in 16 MDPs the number of exploratory moves is reduced from 2500 to 100 steps – an order of magnitude difference. The curves also indicate that the algorithm avoids negative transfer. Exploring according to the learned prior results in consistently improved performance even when early tasks are not related to the current task.

In order to better illustrate the benefit of transfer, we consider a more difficult problem. We no longer assume that goal locations are fixed. Instead, goal locations are sampled from a hierarchical distribution. This confounds the exploration problem for uninformed agents, which must now identify the goal location and the color costs. Termination is achieved when the agent finds the goal location. We present results for larger maps, 30x30 squares, with 4 underlying classes of goal locations (each class has goal locations distributed around a fixed map square). Results are shown in Figure 4. The graph depicts the average performance in the first episode given a fixed number of prior tasks. By comparison, we show the performance of the algorithm with zero prior tasks. Improvements indicate a reduction in the number of steps needed to find the goal. The observed improvement in average total reward is dramatic. After ten tasks the agent has inferred a reasonable representation of the hierarchical structure that generates the goal locations. Knowledge of this hierarchical structure reduces the exploration problem. The agent only checks regions where goals are likely to exist. Exploiting this hierarchical knowledge reduces the cost of exploration by an order of magnitude.

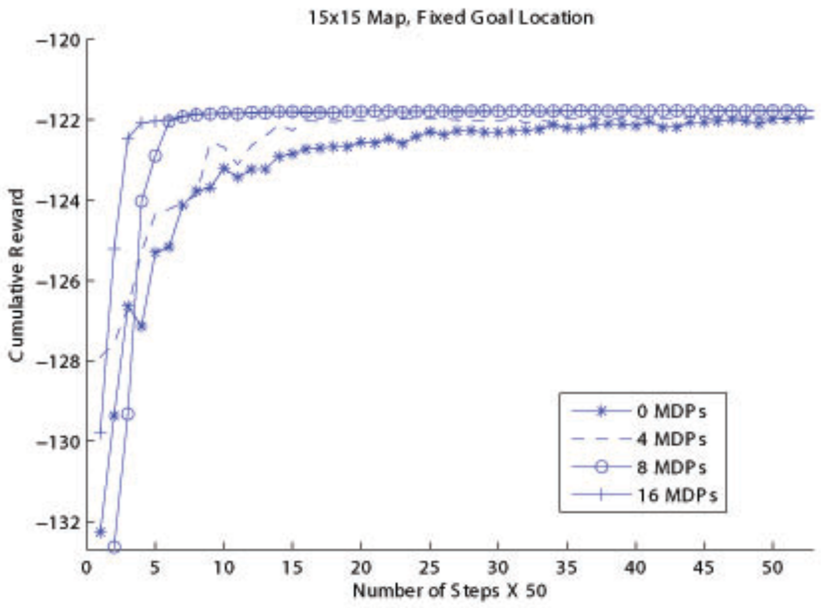


Figure 3: Learning curves for the fixed goal location problem. Performance is evaluated after 0, 4, 8, and 16 tasks have been solved.

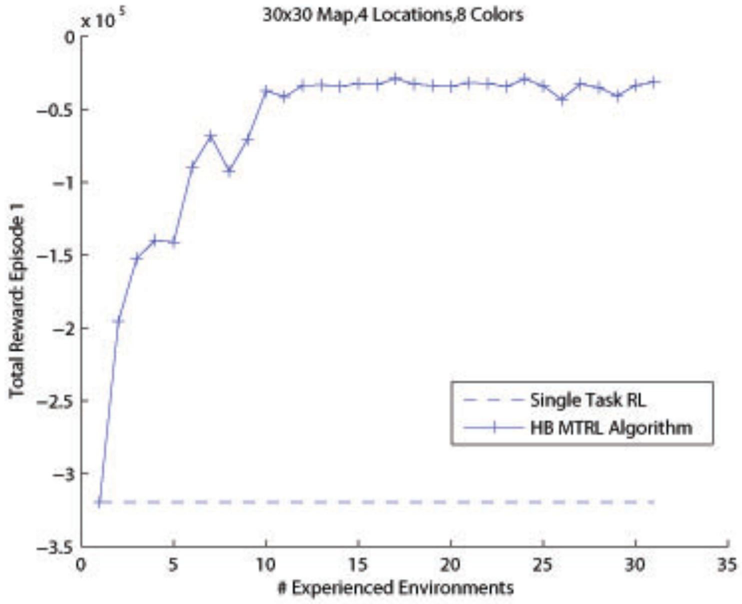
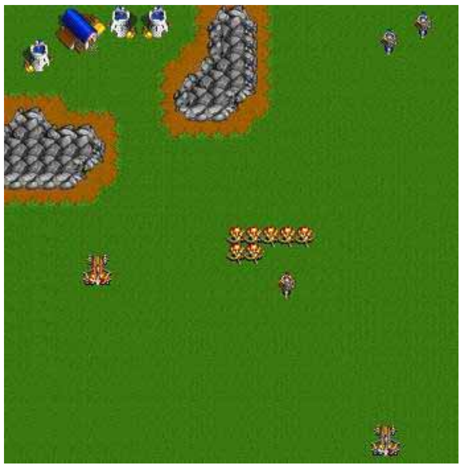


Figure 4: Average cost of finding the hidden goal location in the first episode. Performance is a function of experience in previous tasks.

4. Hierarchical Bayesian Policy Transfer

The premise behind the hierarchical transfer hypothesis, as it applies to policy search, is that policies are hierarchically composed of shared components. Our Bayesian Policy Search

(a) 

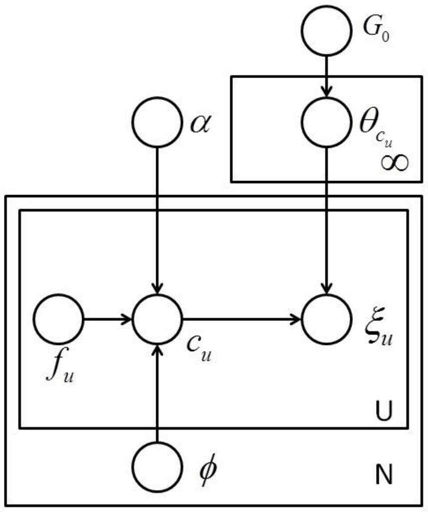
(b) 

Figure 5: (a) A Wargus map. The friendly (red) team is composed of one ballista, seven archers, and one knight. The agent's objective is to destroy the structures and units on the blue team. The agent must select, at each time step, a target for each unit on its team (targets can include friendly units). (b) The hierarchical prior distribution for unit roles. The set of parameters Ψ includes: The prior distribution on role parameters G_0, the set of roles θ_c, the assignment parameters ϕ, unit features f_u, the concentration parameter of the DP model α, the assignment variable for each unit c_u, and the observed trajectory data ξ generated by executing role based policies. The plate notation indicates that there are N tasks and each task is composed of U units.

(BPS) algorithm learns these components in simple tasks and recombines them into a joint policy in more difficult tasks. In some cases, tasks that are impossible to tackle with no prior knowledge become trivial if the necessary component set is learned from simpler tasks.

We pursue this basic idea in the context of a real time strategy game called Wargus. The tactical conflicts we consider include groups of heterogeneous units fighting to destroy one another. An example Wargus map is shown in Figure 5(a). Units controlled by the agent include a ballista, a knight, and several archers. Each unit has a set of physical characteristics including armor, speed, location, and so on. These characteristics naturally predispose units to particular roles in confrontations. For instance, due to its long range the ballista is useful for destroying structures, while units close to key pieces (such as the opposing ballista) may be best employed targeting those pieces. Therefore, we view a policy for Wargus conflicts as a composition of roles (components) and a means of assigning roles to units (combining components). In practice, the agent does not know the number of roles, the type of roles, or the correct method of assigning roles to units. Therefore, the goal of our lifelong learning agents will be to learn (or be taught by example) a collection of roles useful for a variety of confrontations and to adapt its method of assigning units to roles as is appropriate for novel tasks.

Our approach to the role discovery and assignment problem is to define a hierarchical prior distribution for the set of individual agent roles, and then to search this structure using stochastic simulation algorithms adapted to the policy search problem. We first define a set of policy components in the form of a parametric class of policies, then define a hierarchical Bayesian prior in the form of a modified DP model, and finally, we define a means of searching the policy space for solutions.

It is assumed that each unit in the game acts independently of the others given their assigned roles. The distribution of the joint action for a set of agents, conditioned on the assigned roles, is the product of the individual action choices, $P(A|s) = \prod_u P(a_u|s, \theta_{c_u})$, where A is the joint action, c_u is a role index for unit u, and each θ_{c_u} represents the vector of role parameters assigned to unit u.

The hierarchical prior distribution for the role parameters, which describes a joint policy for all agents, is shown in Figure 5(b). The prior illustrates how roles are combined and associated with units in the game. There are N tasks. Each task is composed of U units. Each unit is assigned a vector of role parameters θ_{c_u} indexed by c_u. The variable ξ_u represents a set of trajectories, a sequence of state-action observations, made by agent u when executing its policy. Assignment of units to roles depends on unit-specific features f_u and a vector of role assignment parameters ϕ. The vector of assignment parameters ϕ represent a strategy for assigning the collection of units to roles. An instance of the role assignment strategy is associated with each of the N tasks. The set of available roles θ_{c_u} is shared across all of the N tasks. Taken together, the set of parameters, $\Psi = (\theta, c, \phi, \alpha)$, represents a joint policy for all observed tasks.

For purposes of searching the joint policy space we build on the work by Hoffman et al. (2007). We define an artificial probability distribution,

$$q(\Psi) \propto U(\Psi)P(\Psi) = P(\Psi) \int R(\xi)P(\xi|\Psi)d\xi,$$

proportional to the prior distribution and a utility function. For purposes of applying this idea to the policy search problem the utility function is defined to be the expected return for a policy executed in an episodic environment, $V(\Psi) = E(\sum_{t=1}^{T} R(s_t, a_t)|\Psi)$. This formulation allows us to adapt advanced methods of stochastic simulation, developed for estimation of probability distributions, to the problem of policy search. Essentially, the prior distribution acts as a bias. It guides the search to regions of the policy space believed to have a high return. The utility function plays a similar role. Samples drawn from $q(\Psi)$, will tend to focus around regions of the policy space with high utility.

Given N previously observed tasks, we take the following strategy. We fix the set of previously identified roles and use this role set as prior knowledge in the new task. Discovery of roles is a difficult process. Therefore, role reuse in the new task can dramatically improve learning speed. Within a new task we search for strategies that recombine these existing roles and propose new roles to overcome novel obstacles. Algorithm 2 shown in Figure 2 is designed to implement this strategy by sampling role-based policies given the informed prior.

The algorithm has four basic parts. We assume that trajectories of optimal policies are available for the first N tasks. If no prior task is available the trajectory set will be empty. Trajectories are stored to be used during inference. This brings us to the main body of the sampling procedure, which generates samples from the artificial distribution $q(\Psi)$. On line 5, after initializing the state of the Markov chain, the new role assignments are sampled.

Algorithm 2 Bayesian Policy Search

```
1:  Initialize parameters of the role-based policy: $\Psi_0$
2:  Initialize the sets of observed trajectories: $\xi = \emptyset$
3:  Generate $n$ trajectories from the domain using $\Psi_0$.
4:  $\xi \leftarrow \xi \cup \{\xi_i\}_{i=1..n}$
5:  $S = \emptyset$
6:  for $t = 1 : T$ do
7:      //Generate a sequence of samples from the artificial distribution.
8:      $\Psi_t \leftarrow Sample(\hat{U}(\Psi_{t-1})P(\Psi_{t-1}))$
9:      Record the samples:
10:     $S \leftarrow S \cup (\Psi_t)$
11: end for
12: Set $\Psi_0 = argmax_{(\Psi_t) \in S} U(\Psi_t)$ //Select the role-based policy maximizing the expected
    return.
13: Return to Line 2.
```

Each agent is placed into one of the existing components, or (using the mechanics of the DP) a new component generated from G_0. Next, line 6 updates the role parameters using Hybrid Markov Chain Monte-Carlo (Andrieu et al. (2003)). The key to this portion of the algorithm is the use of gradient information to guide the simulation toward roles which yield high expected return. Next, we perform a simple Metropolis-Hastings update for the vector of assignment parameters. Finally, the role-based policy with highest predicted utility is used to generate a new set of trajectories. The concrete implementation of each sampling procedure can be found in Wilson et al. (2010), which includes details of the model-free method of estimating the utility.

Experimental Results. Our algorithm is agnostic about the process generating trajectories. Trajectories may be produced by an expert, or automatically generated by the learning agent. Figure 6 illustrates the results of transfer from a simple map, when given examples of expert play in the simple map, to a more complex map. An expert demonstrates 40 episodes of play in the simple map. The expert controls 6 friendly units. Given the expertly generated trajectories, roles and an assignment strategy are learned using the BPS algorithm. The agent attempts to transfer these learned roles to a more difficult Wargus map where it will control 10 friendly units (see Figure 5(a)). We compare the transfer performance, BPS:Expert, to three baselines. The first two baselines (BPS Independent and BPS Single) represent the BPS algorithm with an enforced role structure. In the Independent case, all units are forcibly assigned to their own role. In the Single case, all units share the same role. In principle, BPS with the independent role structure can learn the correct policy and this fixed role structure may be beneficial when roles are not shared. Similarly, if a single role will suffice, then BPS run with all agents assigned to a single policy will be ideal. Independent and Single baselines cannot make use of the role structure, and therefore they cannot benefit from transfer. Instead, they must learn their policies from scratch. Finally, we compare our performance to OLPOMDP where units independently learn a policy (Baxter et al. (2001)). Results are shown in Figure 6. The BPS algorithm benefits substantially from identifying and transferring the expert roles.

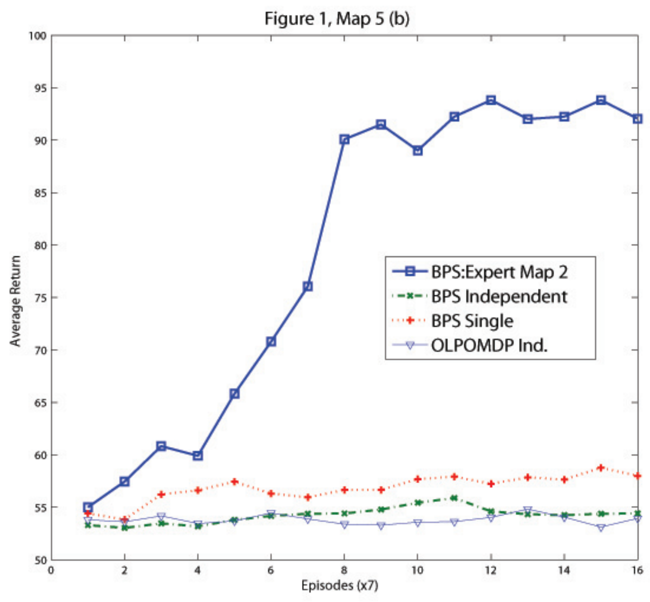


Figure 6: Transfer results of BPS given expert examples of play and tested on a map with 10 units.

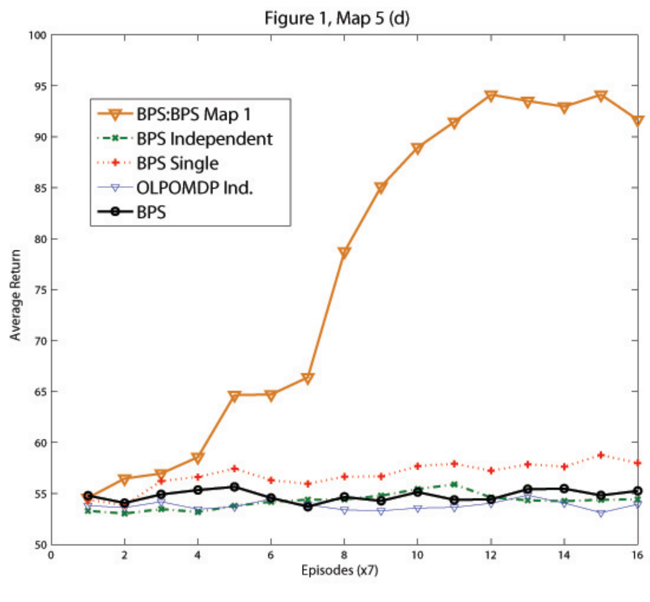


Figure 7: Transfer results of BPS algorithm given training in a simple map with 6 units and tested on a map with 10 units.

Experts are not always available. Unfortunately, learning in our test map is more difficult when examples of expert play are not provided. On the test map used above, which represents a large multi-agent system, it was difficult to learn a good policy using BPS without prior knowledge. To overcome this problem without resorting to expert help we first allow BPS to learn a useful role structure in a simpler map (by observation of its own behavior). Subsequently, we use the learned prior distribution in the larger 10 unit map. The results are shown in Figure 7, BPS:BPS Map 1. The roles learned in the simpler problem make learning the harder task tractable. By comparison, BPS without the prior knowledge takes much longer to identify a policy.

5. Conclusion

This work focuses on a hierarchical Bayesian framework for transfer in sequential decision making tasks. We describe means of transferring two basic kinds of knowledge. The HMBT algorithm learns a hierarchical structure representing classes of MDPs. The HMBT algorithm transfers knowledge of domain models to new tasks by estimating a hierarchical generative model of domain model classes. We illustrated the effectiveness of learning the MDP class hierarchy in a simple colored maze domain. Results show that transferring the class structure can dramatically improve the quality of exploration in new tasks. Furthermore, the special properties of the HMBT algorithm prevent transfer of incorrect knowledge. The BPS algorithm learns a hierarchical prior for multi-agent policies. It assumes that policies for each domain are composed of multiple roles, and the roles are assigned to units in the game via a hierarchical strategy. Our empirical results show that tactical conflicts in Wargus exemplify this hierarchical structure. Distinct roles can be discovered and transferred across tasks. To facilitate this transfer, the BPS algorithm can learn roles from expert examples as well as observing its own play in simple tasks. By learning in simple task instances first the algorithm can tackle increasingly sophisticated problems.

References

Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1):5–43, January 2003. doi: 10.1023/A:1020281327116.

Jonathan Baxter, Peter L. Bartlett, and Lex Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15(1):351–381, 2001. ISSN 1076-9757.

Mathew Hoffman, Arnaud Doucet, Nando de Freitas, and Ajay Jasra. Bayesian policy learning with trans-dimensional MCMC. *NIPS*, 2007.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *ICML*, pages 1015–1022, 2007.

Aaron Wilson, Alan Fern, and Prasad Tadepalli. Bayesian role discovery for multi-agent reinforcement learning. In *AAAI*, 2010.