Bayesian Policy Search for Multi-Agent Role Discovery

Aaron Wilson and **Alan Fern** and **Prasad Tadepalli**
School of EECS, Oregon State University

Abstract

Bayesian inference is an appealing approach for leveraging prior knowledge in reinforcement learning (RL). In this paper we describe an algorithm for discovering different classes of roles for agents via Bayesian inference. In particular, we develop a Bayesian policy search approach for Multi-Agent RL (MARL), which is model-free and allows for priors on policy parameters. We present a novel optimization algorithm based on hybrid MCMC, which leverages both the prior and gradient information estimated from trajectories. Our experiments in a complex real-time strategy game demonstrate the effective discovery of roles from supervised trajectories, the use of discovered roles for successful transfer to similar tasks, and the discovery of roles through reinforcement learning.

Introduction

In most real-world domains, there are multiple agents or agents that play different roles in jointly accomplishing a task. For example, in a military battle, a tank might engage the enemies on the ground while an attack aircraft provides the air cover. In a typical hospital, there are well-delineated roles for the receptionists, nurses, and the doctors. In this paper, we consider the general problem of discovering the roles of different agents and transferring that knowledge to accelerate learning in new tasks.

We approach the role learning problem in a Bayesian way. In particular, we specify a non-parametric Bayesian prior (based on the Dirichlet Process (DP)) over multi-agent policies that is factored according to an underlying set of roles. We then apply policy search using this prior. Drawing on work by (Hoffman et al. 2007), our approach to the policy search problem is to reduce policy optimization to Bayesian inference by specifying a probability distribution proportional to the expected return which is then searched using stochastic simulation. This approach is able to leverage both prior information about the policy structure and interactions with the task environment. We define such a distribution for our multi-agent task, and show how to sample role-based policies from it. Thus, our work can be viewed as an instance of Bayesian RL, where priors are learned and specified on multi-agent role-based policies.

Previous work on Bayesian RL has considered priors on the domain dynamics in model-based RL (Dearden, Friedman, and Andre 1999; Strens 2000; Poupart et al. 2006; Wilson et al. 2007) and priors on expected returns, action-value functions, and policy gradients in model-free RL (Dearden, Friedman, and Russell 1998; Engel, Mannor, and Meir 2005; Ghavamzadeh and Engel 2007). Our work is contrasted with these past efforts by being both model-free and by placing priors directly on the policy parameters. In our role-based learning formulation, this kind of policy prior is more natural, more readily available from humans, and is easily generalizable from experience in related tasks.

The Bayesian approach allows us to address a variety of role learning problems in a unified fashion. First, we demonstrate learning roles in the supervised setting where examples of optimal trajectories are provided by an expert. We then show that priors learned in one task can be transferred via Bayesian RL to a related task. Finally we show that roles can be discovered along with policies through Bayesian RL. All of our experiments are done in the real-time strategy (RTS) game of Wargus, focusing on tactical battle problems between groups of friendly and enemy agents.

In summary, the key contributions of this paper are as follows: 1) Introducing a novel non-parametric Bayesian model over role-based multi-agent policies based on the Dirichlet process, 2) Introducing a novel model-free Bayesian RL algorithm that can accept priors over policy parameters, 3) An evaluation of the approach in the real-time strategy game of Wargus, showing its effectiveness for supervised learning, transfer, and pure RL.

Problem Formulation

Multi-Agent MDPs. A Markov decision process (MDP) is a 4-tuple (S, A, R, T), where S is a set of states, A is the action space, $R(s, a)$ is the immediate reward of executing action a in state s, and the transition function $T(s, a, s')$ is the probability of reaching state s' given that action a is taken in state s. A policy π is a stochastic mapping from S to A. Every policy π executed from a starting state incurs a reward sequence. In this paper, we will assume that all policy executions reach a terminal state and the goal is to find a policy that maximizes the total expected episodic reward.

Here we consider a special class of MDPs, multi-agent MDPs (MMDPs), which model the problem of central control of a set of cooperative agents. An MMDP is simply a standard MDP where the action space and rewards are factored according to a set of m agents. In particular, the action space A is the product space $A_1 \times \ldots \times A_m$, where A_j is the action set of the j^{th} agent. The reward function R is similarly defined as the sum of individual agent rewards $R(s,a) = \sum_i R_i(s,a_i)$, where $R_i(s,a_i)$ is the reward received by agent i for taking action a_i in state s. The primary complicating factor of solving MMDPs compared to standard MDPs is that the action space is exponential in the number of agents. This means that the desired policy π is a probabilistic mapping from states to an exponentially large joint action space. The key questions are how to compactly represent such policies and learn them from experience.

Pseudo-Independent Policies. In general, large multi-agent domains like Wargus may require arbitrary coordination between agents. Unfortunately representing and learning coordination of this kind is computationally prohibitive for anything but very small problems. To alleviate the computational burden we focus on learning a restricted class of joint agent policies, which in practice are often sufficient to achieve good performance. In particular, we use a pseudo-independent multi-agent policy representation where agents are assumed to be ordered and the decisions of agents at each time step are made in sequence, with later agents being allowed to condition on the decisions of previous agents. More formally, under our representation the probability of selecting joint action $A = (a_1, \ldots, a_m)$ in state s is,

$$P(A|s) = \prod_u P_u(a_u|s, A_{1,u-1}), \qquad (1)$$

which is a product over individual agent policies P_u, where the policy for agent u conditions on the state s and the actions of previous agents in the ordering represented by $A_{1,u-1}$. This pseudo-independent representation allows for efficient representation and evaluation of a policy, at the expense of some expressive power in coordination structure.

Due to the large state and action spaces it is not possible to represent each agent policy P_u explicitly. Thus, we use a common parametric log-linear representation. For this purpose, we assume the availability of a feature function $g(s, a_u, A_{1,u-1})$ that returns a vector of numeric features for a state s, agent action a_u, and the actions of previously ordered agents. Given this feature function, each agent policy is represented via the following Boltzmann distribution,

$$P_u(a_u|s, A_{1,u-1}) = \frac{\exp(\theta_u \cdot g(s, a_u, A_{1,u-1}))}{\sum_{a'_u} \exp(\theta_u \cdot g(s, a'_u, A_{1,u-1}))}, \qquad (2)$$

where θ_u is the parameter vector for agent u. The parameters $\theta = (\theta_1, \ldots, \theta_m)$ define a joint policy over agent actions.

Role-Based Parameter Sharing. In general, a pseudo-independent policy can specify different policy parameters for each agent. However, doing so does not exploit the fact that agents can often be divided into a small number of possible roles, where agents of the same role have identical policies. For example, in Wargus roles arise naturally because the agents, or units, have different characteristics which determine their suitability to particular tasks (e.g. the agents speed, range, durability, and power of their attack). We can capture such role structure by specifying the following: 1) the number of possible roles, 2) a set of policy parameters θ_c for each role c, 3) for each agent u a role assignment c_u, and 4) a means of assigning agents to roles dependent on agent features f_u (speed,location,etc.). Given this information the joint policy assigns all agents to roles and then agents execute their assigned policies, with parameters $\theta = (\theta_{c_1}, \ldots, \theta_{c_m})$, pseudo-independently. The advantage of this representation is the sharing of parameters when agents belong to the same role. In particular, agents assigned to role c can benefit from the experiences of all other agents in the same role when learning the parameters θ_c. In practice a designer may not know the best role structure for a domain, the number of roles that should exist, or how to assign agents to roles. This motivates our goal of automatically learning and exploiting a prior over the number of roles, the role parameters, and the assignment of roles.

Bayesian Policy Search

Background. We formulate the MMDP learning problem as Bayesian policy search. In particular, we assume the availability of a prior distribution $P(\theta)$ over policy parameters, which might be learned from related problems or provided by a human. In the next section we will specify a particular form for this prior that captures uncertainty about the number and types of roles suitable for a given domain. Here we describe a novel model-free Bayesian policy-search algorithm that can exploit priors in order to speed up learning.

Our work builds on (Hoffman et al. 2007) which uses prior information on the policy parameters in a model based RL setting. Central to this work is the construction of an artificial distribution, proportional to a utility term and a prior on the policy parameters influencing the utility,

$$q(\theta) \propto U(\theta)P(\theta) = P(\theta) \int R(\xi)P(\xi|\theta)d\xi. \qquad (3)$$

In the RL setting ξ corresponds to finite length trajectories from initial states to terminal states. The conditional distribution $P(\xi|\theta)$ is simply the probability that a policy parameterized by θ generates trajectory ξ. And the utility is defined to be the expected return $E(\sum_{t=0}^{T} R(s_t, a_t)|\theta)$ for the policy parameters. Importantly, samples drawn from this artificial distribution focus on policies with high expected return.

Our approach. Given these definitions one can sample from the augmented distribution $q(\theta)$ to search for a policy maximizing the marginal $\theta^* = \arg\max_\theta U(\theta)P(\theta)$. Unfortunately sampling from q is not practical for our problems. In particular, we do not have access to the domain model necessary for generating sample trajectories from $P(\xi|\theta)$. Moreover generating a large number of sample trajectories based on actual experience is costly. This motivates our adaptation of this approach to the model-free RL setting.

We elect to sample directly from an estimate $\hat{U}(\theta)P(\theta)$ of the target marginal distribution. To evaluate this product at any point θ we propose to use a finite sample drawn from a sequence of policies. To do so we use importance sampling. Importance sampling allows us to evaluate, in off-policy fashion, the expected return of any policy we wish.

Algorithm 1 Bayesian Policy Search

```
1: Initialize parameters: $\theta_0, \xi = \emptyset$
2: Generate $n$ trajectories from the domain using $\theta_0$.
3: $\xi \leftarrow \xi \cup \{\xi_i\}_{i=1..n}$
4: $S = \emptyset$
5: for $t = 1 : T$ do
6:     $\theta_t \leftarrow Sample(\hat{U}(\theta_{t-1})P(\theta_{t-1}))$
7:     Record the samples: $S \leftarrow S \cup (\theta_t)$
8: end for
9: Set $\theta_0 = argmax_{(\theta_t)\in S} U(\theta_t)$
10: Return to Line 2.
```

Algorithm 2 BPS For Role Based Policies

```
1: Initialize all parameters: $(\theta_0, c_0, \phi)$
2: Given the input $\xi$ and the initialized parameters generate samples from $q(\theta, c, \phi)$.
3: $S = \emptyset$
4: for $t = 1 : T$ do
5:     $c_t \leftarrow SampleAssignments(\boldsymbol{\theta_{t-1}}, \mathbf{c_{t-1}}, \boldsymbol{\phi_{t-1}})$
6:     $\theta_t \leftarrow SampleRoleParameters(\boldsymbol{\theta_{t-1}}, \mathbf{c_t}, \boldsymbol{\phi_{t-1}})$
7:     $\phi_t \leftarrow SampleKernelParameters(\boldsymbol{\theta_t}, \mathbf{c_t}, \boldsymbol{\phi_{t-1}})$
8:     $S \leftarrow S \cup (\boldsymbol{\theta_t}, \mathbf{c_t}, \boldsymbol{\phi_t})$
9: end for
```

Please see (Shelton 2001) for details on the weighted importance sampling algorithm used to estimate the expected return and its gradient in this paper. Intuitively, our agent will alternate between stages of action and inference. It generates observed trajectories by acting in the environment according to its current policy, then performs inference for the optimal policy parameters given its experience so far, generates new trajectories given the revised policy parameters, and so on. An outline of our Bayesian Policy Search (BPS algorithm) procedure is given in Algorithm 1. The next section details the prior used in our MMDP setting and the Markov Chain Monte Carlo (MCMC) algorithm used to generate samples from $\hat{U}(\theta)P(\theta)$ (Line 6 of the BPS algorithm).

Role Learning with a DP prior

A Role Based Prior. To encode our uncertainty about the type of roles (what behaviors we expect), the number of roles, and the assignments of agents to roles we use a modification of the Dirichlet Process (DP) prior distribution which has all the properties we desire. Below we discuss the DP as a generalization of a standard finite mixture model and the modifications of the inference process we employ to sample role based policies.

It is easy to interpret the DP, in the form of an infinite mixture model with concrete assignment variables, in terms of our role based setting. The DP parameterizes the joint policy of all agents with a set of role assignments c, a set of role parameters θ_{c_i}, and a set of expert assignment parameters ϕ. The DP specifies a distribution over all of these parameters. Thus, in our case, inference in the DP is a search through all possible partitions of agents into roles. Though we do not give a full treatment of the DP in this paper we do specify how we sample from the standard conditional distributions used in Gibbs sampling algorithms for the DP (Neal 2000).

With the DP in hand we can define our own artificial probability distribution,

$$q(\theta, c, \phi) \propto [\widehat{U}(\theta)][\prod_j G_0(\theta_j)]P(c|\phi), \qquad (4)$$

which decomposes into the estimated expected return $\widehat{U}(\theta)$, and the prior distribution $P(\theta, c) = [\prod_j G_0(\theta_j)]P(c|\phi)$. The prior further decomposes into two parts. The first factor is the prior over role parameters, $G_0(\theta_j)$, encoding our uncertainty about where we expect to find good solutions in the role parameter space. For our experiments we define G_0 to be a multivariate Gaussian distribution with zero mean and diagonal covariance. This asserts that a priori we believe the parameters of each role tend to be close to zero and that we do not know of correlations within the role parameter vector. The distribution $P(c|\phi)$ captures our uncertainty over role assignments. It is conditioned on a new set of parameters ϕ which influence the probability of assigning an agent to a role as a function of the agents features.

Overview: BPS for Role Based Policies. Algorithm 2 outlined here, is a specialization of line 6 of the BPS algorithm, designed to sample role based policies. At each point the state of the Markov chain will include a tuple of parameters, (θ, c, ϕ), which we will simulate in turn. The algorithm has three basic parts. First, after initializing the state of the chain, the new role assignments are sampled placing each agent into one of the existing components, or (using the mechanics of the DP) a new component generated from G_0. The second part of the algorithm updates the role parameters using Hybrid MCMC (Andrieu et al. 2003). The key to this portion of the algorithm is the use of log gradients to simulate samples from $q(\theta|c, \phi)$, focusing samples on regions of the policy space that are promising. Finally, we perform a simple Metropolis-Hastings update for the vector of kernel parameters. In the following sections we more explicitly define the implementation.

Sampling Role Assignments. The advantage of working with DP models is that they give rise to succinct conditional distributions for the assignment variable. The DP conditional distribution for role assignments is,

$$P(c_u = j|c_{(-u)}, \alpha) = \begin{cases} \frac{n_{(-u),j}}{n-1+\alpha} & |n_{(-u),j} > 0 \\ \frac{\alpha}{n-1+\alpha} & |\text{otherwise} \end{cases}, \qquad (5)$$

encoding the probability of assigning agent u to role j given all the other assignments $c_{(-u)}$. Note that the probability is proportional to the number of agents currently assigned to role j, $n_{(-u),j}$, and the DP focus parameter α (n is the total number of agents). The second term of this distribution assigns some probability to a role which currently has no data assigned to it allocating probability to unseen roles. This distribution can be used naturally to Gibbs sample new role assignments. Before we describe how this is done we modify the distribution.

As currently written the distribution depends only on the proportion of agents already assigned to a component. Ideally this distribution should take into account the similarities between agents when computing the conditional probability.

To incorporate this idea we modify Equation 5, adapted from (Rasmussen and Ghahramani 2002), to depend on the agent features by replacing $n_{-u,j}$ with the value of a parameterized kernel which we define to be,

$$n_{-u,j} = (n-1)\frac{\sum_{u' \neq u} K_\phi(f_u, f_{u'})\delta(c_{u'} = j)}{\sum_{u' \neq u} K_\phi(f_u, f_{u'})}. \qquad (6)$$

Equation 6 determines the weighted sum of distances between the agent in question and all agents in role j and divides by the sum of distances between u and all other agents. We choose to use the kernel below which encodes a similarity metric between agent features, and allows the role assignment parameters to determine feature relevance,

$$K_\phi(f_u, f_{u'}) = exp(-\frac{1}{2}\sum_d (f_{u,d} - f_{u',d})^2/\phi_d^2). \qquad (7)$$

We choose this kernel because it can decide when contextual features are irrelevant, and when differences in innate features such as 'range' strongly influence to which role an agent belongs. Principally the modified conditional distribution can substantially bias role assignments when the system is confronted with new tasks. This change was crucial to success in our experiments.

The assignment of agents to roles must be conditioned on their performance in the selected role. We encode this dependence in Equation 8. The equation has two primary factors. The first factor is new and encodes that the probability of assignment is proportional to the impact the role parameters have on the expected return $\widehat{U}(\theta)$. The second factor is the prior probability of the role parameters which we have already discussed. Additionally, due to our modification, simulation will tend to assign agents with similar agent features f_u to the same role.

$$P(c_u = j | c_{-(u)}, \theta, \phi, \alpha) = \begin{cases} b\frac{n_{-(u),j}}{n-1+\alpha}\widehat{U}(\theta)G_0(\theta_j) | n_{-u,j} > 0 \\ b\frac{\alpha}{n-1+\alpha}\widehat{U}(\theta)G_0(\theta_{new}) | \text{otherwise} \end{cases}. \qquad (8)$$

Importantly, we employ an auxiliary role method to allow for new roles to be automatically generated by the prior. This means that we always store the set of roles which have some agents already assigned to them, and in addition store an auxiliary set of roles with parameters initialized from the prior G_0 as needed. In other words, the first case of Equation 8 corresponds to the existing roles and the second case governs the auxiliary roles. The Sample Assignments function samples a new assignment for each unit in turn exiting once all units have new assignments. In the next section we use hybrid MCMC to sample new role parameters.

Sampling Role Parameters. To update the role parameters θ we use Hybrid MCMC. Hybrid MCMC biases new samples using information about the log gradient of the target distribution. Doing so helps guide the sampling process for high dimensional distributions. In our case the target distribution, given the fixed assignments for all other parameters is $P(\theta|c,\phi) \propto \widehat{U}(\theta)\prod_j G_0(\theta_j)$. So long as we can compute the log gradient of this function we can use the hybrid rejection method. The expected return is estimated using importance sampling, and a treatment of the gradient we compute for the first term can be found in (Meuleau, Peshkin, and Kim 2001). The second term is simply the log gradient of a product of multi-variate normal distributions. Naturally, this can be computed. With the necessary gradient information in hand we can make use of Hybrid MCMC to construct a probing distribution for $P(\theta|c,\phi)$. Samples returned from Hybrid MCMC are used to update the state of θ

Sampling Kernel Parameters. We use Metropolis-Hastings to update the kernel parameters given the sampled state for (c,θ), using a Gaussian proposal distribution. Here we use the pseudo-likelihood, proportional to $[\widehat{U}(\theta)][\prod_j G_0(\theta_j)][\prod_u P(c_u|\phi)]$, to determine acceptance.

Acting in the Environment. After drawing a large sample using this simulation procedure we wish to select a policy, a tuple of parameters (θ_t, c_t, ϕ_t), to execute in our task. To do so we select the policy maximizing,

$$(\theta^*, c^*, \phi^*) = argmax_{(\theta_t, c_t, \phi_t) \in S} U(\theta_t). \qquad (9)$$

The sampled policy is then executed in the domain to generate new trajectories (Line 2 of the BPS algorithm).

Results

We evaluate our algorithm on problems from the game of Wargus. RTS games involve controlling multiple agents in activities such as: resource gathering, building military infrastructure/forces, and engaging in tactical battles. We focus on tactical battles where we must control a set of friendly agents to destroy a set of enemy buildings and agents controlled by the native Wargus AI.

Decision cycles occur at fixed intervals, where all agents select an action. The possible actions include attacking any of the friendly or enemy units, or doing nothing. Attack actions cause an agent to pursue the selected target and attack when in range. At the end of each decision cycle the state is updated and the agents receive rewards, which include a small reward for damaging a target plus a bonus reward for killing the target plus a large reward if the team has won the game (rewards are negative when attacking and killing friendly units). A game continues until one side is destroyed, or for a maximum of 100 decision epochs.

Our learning algorithm requires us to provide agent specific features f_u to facilitate role assignment. For this purpose, we use as features the basic attributes of agents as described in the Wargus configuration file including information such as armor strength, speed, and attack range. We also include features that capture properties of an agent's initial state such as distance between agent and enemy targets, and distance to friendly agents. Thus f_u captures both inherent physical capabilities of an agent as well as contextual information in the initial state. Our log-linear policy representation also requires that we provide a feature set $g(s, a_u, A_{1,u-1})$ that captures properties of the current state s, candidate agent action a_u, and other agent actions. We hand-coded a set of 20 features that capture information such as whether the range of the target exceeds that of the agent, whether the target is mobile, and how many agents are currently attacking the target.

	Friendly			Enemy		
	Archers	Ballista	Knights	Towers	Ballista	Knights
Map 1	4	1	0	2	0	1
Map 2	4	1	1	2	1	1
Map 3	16	1	2	4	2	5
Map 4	7	1	0	2	0	2
Map 5	8	1	1	4	1	2

Table 1: Unit compositions for each map.

We test our algorithm on a variety of Wargus maps with varying locations and compositions of units. The basic compositions of units for each map are shown in Table 1.

Learning from an Expert. An important advantage of our learning approach is that it naturally benefits from expert examples. Here we test the ability of our algorithm to discover the inherent role structure observed in expert play in Map 1, Map 2, and a much harder map Map 3. In each case, the algorithm is provided with 40 episodes of observed performance of high-quality (expertly coded) policies, which were based on natural role structures for each task. To test the performance of our algorithm we allow BPS to learn a policy given the expert samples as input. The performance of the learned policy is then assessed on the target maps. In all three cases the agents win 100 percent of games after learning (prior to learning no games are won). We assess how well the discovered roles match those used in the expert policies by examining the decomposition of units in the sampled assignment vector c^*. In each case BPS learns role structures consistent with the expert policy, including the appropriate role assignments and role parameters. Two roles are found for the first map, which correspond to archers defending the ballista and the ballista attacking the enemy base. The algorithm discovers that the knight deserves its own role in map 2 (it is good at killing the enemy ballista). Likewise, in the difficult third map the sampled policies partition the agents appropriately.

Prior Transfer. Based on the expert demonstrations our algorithm is able to learn a role structure, role assignment function, and role parameters. We now evaluate the ability of BPS to transfer this learned knowledge, in the form of a prior distribution over parameters (θ, ϕ), to speedup RL on new, but related, tasks. In particular, we conduct two experiments. The first uses the posterior distribution learned from expert demonstrations on Map 1 as a prior for RL in Map 4. The second experiment is identical but uses the posterior learned from expert play in Map 2 for RL in Map 5. We compare the transfer performance to three baselines. The first two baselines (BPS Independent and BPS Single) represent running the BPS algorithm with an enforced role structure. In the independent case all agents are forcibly assigned to their own component and in the single case all agents share the same policy. In principle BPS with the independent role structure can learn the correct policy and this fixed assumption may be beneficial if the role structure is unimportant. Similarly, if a single role will suffice, then BPS run with all agents assigned to a single policy will be optimal. Because the Independent and Single baselines cannot make use of the role structure they cannot benefit from transfer. Instead they must learn their policies from scratch. Furthermore, we compare our performance to OLPOMDP where agents independently learn their own policies (Baxter, Bartlett, and Weaver 2001). In principle OLPOMDP should be able to find a locally optimal policy.

The results for the first experiment are shown in Figure1(a). The curve (BPS:Expert Map 1) illustrates the advantages of transfer using the prior distribution learned from expert samples on Map 1. The graph shows the average reward per episode for the learned policy as a function of the amount of experience (number of episodes) used to learn the policy. The algorithm benefits substantially from the improved prior distribution as it focuses samples on policies with good role structures. We see similar results for the more complex Map 5 illustrated by the curve (BPS:Expert Map 2) in Figure1(b). In this case a larger number of roles needs to be correctly mapped to new agents making the problem substantially more difficult when roles are not transferred. Our learning algorithm, with the improved prior distribution, substantially outperforms the baselines. It is worth noting the performance of BPS Single in Figure1(b). On this map the baseline has discovered a simple but suboptimal policy accumulating a small amount of reward by sending all agents to attack a tower. This results in damaging the tower, but insures that all friendly units will die. The restricted policy representation, coupled with the large number of friendly agents, helps this baseline find this suboptimal policy quickly. The baselines were unable to find the optimal policy before we terminated the experiments.

Autonomous Role Discovery. Learning in both of the test maps becomes much more difficult when examples of expert play are not provided. This can be seen by the poor performance of the baselines, which cannot benefit from prior role knowledge. Here we consider whether our algorithm is able to start from an uninformative prior and autonomously learn the role structure during the actual RL process, and benefit from that structure as it is being learned. The BPS curve in Figure 1(c) shows results of our agent with an uninformed prior. Given the less informed prior many more episodes are required before an optimal policy is found. This is a substantial benefit over the baseline algorithms which do not win games. This shows that learning the role structure during the actual RL process can speed up RL even when the initial prior is uninformative. The primary reason for this is that experience between agents of the same role is shared leading to faster learning of role policies once good role assignments are discovered. Qualitatively the algorithm did find distinct and natural roles for the archers and ballista. Results on Map 5 are not good for any of the non-transfer algorithms. We conjecture that it would require any RL agent without significant prior knowledge a very long time to discover a winning policy. In fact, when run with a random policy no win was recorded after 500 games. As shown in Figure 1(d) BPS without transfer performs no better than the baselines on this problem. However, as illustrated above, BPS can successfully make use of learned priors to significantly speed learning in new tasks. Thus, we first allow BPS to learn a useful role structure in Map 1 and then make use of the learned prior distribution in Map 5. The results are shown in Figure 1(d), BPS:BPS Map 1; the prior learned in the simpler problem make the harder problem tractable. It is a substantial advantage of the BPS

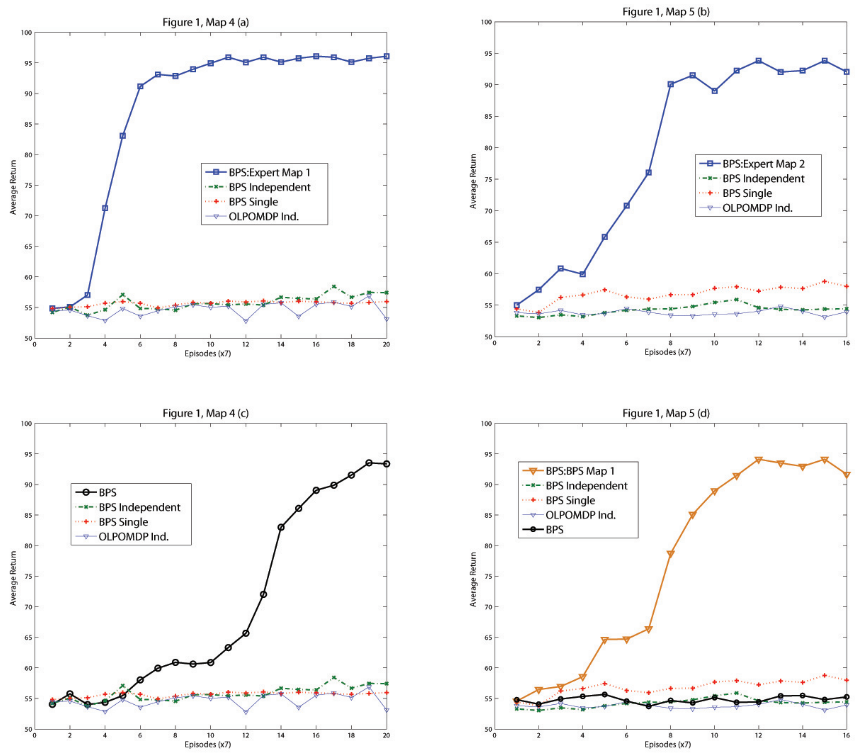

Figure 1: Results for Map 4 and Map 5. Top row: Results of RL after learning prior from expert examples. Bottom row: Learning roles using RL. Each graph shows the average total reward per episode (average over 20 runs).

algorithm that it can naturally leverage experience in simple tasks to learn quickly in hard tasks.

Conclusion

We have introduced a Bayesian Policy Search algorithm for the model-free multi-agent RL problem. Our approach, based on stochastic simulation methods, uses a prior distribution on the policy parameters and gradient information to improve the speed of learning. The resulting simple framework was demonstrated to effectively learn multiple roles from supervised optimal trajectories. Furthermore, we demonstrated that learned roles can be successfully transferred to similar tasks. The framework was also used successfully in the RL setting to autonomously discover new roles that were useful for a task and learn how to map agents to those roles based on their observable properties.

Acknowledgements. We gratefully acknowledge the support of the Army Research Office under grant W911NF-09-1-0153.

References

Andrieu, C.; de Freitas, N.; Doucet, A.; and Jordan, M. I. 2003. An introduction to MCMC for machine learning. *Machine Learning* 50(1):5–43.

Baxter, J.; Bartlett, P. L.; and Weaver, L. 2001. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research* 15(1):351–381.

Dearden, R.; Friedman, N.; and Andre, D. 1999. Model based Bayesian exploration. In *UAI*.

Dearden, R.; Friedman, N.; and Russell, S. 1998. Bayesian Q-learning. In *AAAI*.

Engel, Y.; Mannor, S.; and Meir, R. 2005. Reinforcement learning with Gaussian processes. In *International Conference on Machine Learning*, 201–208.

Ghavamzadeh, M., and Engel, Y. 2007. Bayesian policy gradient algorithms. In *NIPS*.

Hoffman, M.; Doucet, A.; de Freitas, N.; and Jasra, A. 2007. Bayesian policy learning with trans-dimensional MCMC. *NIPS*.

Meuleau, N.; Peshkin, L.; and Kim, K.-E. 2001. Exploration in gradient-based reinforcement learning. *Technical Report*.

Neal, R. M. 2000. Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics* 9(2):249–265.

Poupart, P.; Vlassis, N.; Hoey, J.; and Regan, K. 2006. An analytic solution to discrete Bayesian reinforcement learning. In *ICML*.

Rasmussen, C. E., and Ghahramani, Z. 2002. Infinite mixtures of Gaussian process experts. In *NIPS*, 881–888.

Shelton, C. R. 2001. *Importance Sampling for Reinforcement Learning with Multiple Objectives*. Ph.D. Dissertation, MIT.

Strens, M. J. A. 2000. A Bayesian framework for reinforcement learning. In *International Conference on Machine Learning*, 943–950.

Wilson, A.; Fern, A.; Ray, S.; and Tadepalli, P. 2007. Multi-task reinforcement learning: a hierarchical Bayesian approach. In *ICML*.