Learning and Transferring Roles in Multi-Agent Reinforcement

Aaron Wilson and **Alan Fern** and **Soumya Ray** and **Prasad Tadepalli**
School of Electrical Engineering and Computer Science
Oregon State University, USA

Abstract

Many real-world domains contain multiple agents that play distinct roles in achieving an overall mission. For instance, in tactical battle scenarios, a tank fulfills a much different role than a foot soldier. When learning an overall multi-agent strategy, it is clearly advantageous to have knowledge of the agent roles and the typical behavioral characteristics associated with each role. The goal of this paper is to learn and transfer information about agent role structure in the setting of multi-task reinforcement learning. We present a hierarchical Bayesian framework for representing and learning a distribution over the unknown set of agent roles and policies from a sequence of related multi-agent RL problems. This knowledge is then transferred to new problems by treating the distribution as an informative prior over the roles and policies of individual agents. In particular, in this work, the prior is used to initialize policy-gradient search. We demonstrate the effectiveness of this role transfer method on a real-time strategy game in the context of a sequence of tactical battles each involving different numbers and types of individual units. The results show that our approach is able to find a set of underlying roles and use them to significantly speed up learning on novel problems.

Introduction

In most real-world domains, there are multiple units 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. Restaurants have cooks, waiters, and cashiers with similarly well-defined roles. In this paper, we consider the general problem of discovering the roles of different agents through reinforcement learning and transferring that knowledge to accelerate learning in other similar tasks.

We consider the setting of multi-task reinforcement learning, where we are faced with solving a sequence of related learning tasks by taking actions and receiving reinforcement. Just as in the real-world, no two tasks are exactly the same, although they share some similarities. In this work, we assume that the roles of the units in all of these tasks are drawn from the same mixture distribution where there is one mixture component for each role, so that units of the same role have similar policies. Importantly, we further assume that the role assigned to a unit by this distribution can depend on observable unit features. For example, unit features might reflect intrinsic unit properties such as speed and power, along with contextual unit properties such as physical location and proximity to other entities in the initial state. Intuitively the availability of this mixture distribution could speed up learning, e.g. by intelligently initializing unit policies by drawing role assignments and corresponding policy parameters from the mixture distribution based on the unit features in the initial state. Our goal in this work is to learn such a mixture distribution from previous tasks and to then transfer it to new tasks in order to obtain such speedups.

It is important to note that the underlying unit roles are never observed by our system, but must be discovered through learning in prior tasks. Furthermore, we do not assume knowledge of the number of underlying unit roles. For the purpose of learning and transferring an unknown number of roles we utilize an Dirichlet Process model. Given the set of unit policies from previous tasks the model automatically clusters the policies into an appropriate number of role components and learns a distribution over policies for each component. For example, the model might learn that units with "tank-like" unit features typically have policies that focus on battling heavily-armored ground units, and vehicles that fly are good for rescue missions in hard-to-reach locations. Given such a model and a new problem, the units can either be assigned to existing unit roles, or to a new role in cases where a unit does not appear to fit any component. Thus, the use of the infinite mixture model allows us to be robust to the discovery of new roles as tasks are encountered, rather than forcing all units in future problems to fit into the fixed set of previously discovered roles.

Our experiments are based on tactical battle scenarios in a real-time strategy domain called Wargus. In this domain each side owns certain types of units such as towers, archers, knights and lumber mills and tries to destroy the other side's units. The learning agent controls one of the sides, while the other side is controlled by the game engine's built-in AI. In one of the experiments we trained the learner on a simpler

scenario and transferred the knowledge to a more complex scenario with many more units on each side. In the second experiment we also included a new type of unit which is not present in the source task. Each agent was trained using policy-gradient descent, and in the transfer case had the policy parameters initialized according to the transferred role mixture model. In both experiments, the system was able to discover useful roles for different units and successfully transfer them to the target scenario with significant jump start in performance.

The use of roles has been explored in several multi-agent domains including robot soccer and wargames ((Martinson and Arkin 2003; Stone and Veloso 1999; Marsella et al. 1999)). It is noted in (Marsella et al. 1999) that individual agents in robocup soccer tend to specialize their policies towards specific roles. Our own work seeks to find the underlying set of roles describing the various kinds of specializations in order to speed learning in new scenarios. Much work in both soccer domains and war domains is dedicated to the problem of discovering a switching policy between known roles. In (Stone and Veloso 1999) agents communicate directly during play to choose a set of roles for play, and in (Martinson and Arkin 2003) a Q-learning algorithm is employed to learn a dynamic role assignment function. In both of these cases roles can be reassigned during play. In this work role assignment only takes place at the beginning of a task. Thereafter, individual agents are left to adapt their action selection decisions to the new environment. We consider it an important area of future work to learn and transfer such dynamic role switching knowledge.

Multi-Task Reinforcement Learning

In this section, we introduce the Multi-Task Reinforcement Learning problem, which is based on the framework of Markov Decision Processes (MDPs).

An MDP M is defined by a 5-tuple (S, A, C, T), 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, 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 for M is a stochastic mapping from S to A. Every policy π executed from a starting state incurs a sequence of costs. In this paper, we seek to optimize the long-term average cost of the policy over the infinite horizon. In particular, we seek to find a least average-cost policy among a set of parameterized policies. Under reasonable conditions, the average cost is independent of the starting state and is only a function of the policy parameters.

We are interested in multi-agent MDPs (MMDPs), which consist of multiple agents or units taking actions and incurring costs. This can be formalized by treating the action space A in the MDP framework as the product space $A_1 \times \ldots \times A_m$, where A_j is the action set of the j^{th} agent and there are m agents. The immediate costs C and the transition probabilities T are now functions of all agents. We consider the policy space defined by choosing policy parameters π_u for each unit u. Since, the overall policy and its expected average cost is determined by the policy parameters of all units, the goal is to find the set of policy parameters, π_u, that minimize the average cost per time step. In this work, we take the policy gradient approach, which consists of starting with some initial parameter values and adjust them in the direction of the gradient to decrease the cost.

In this work, we focus on the problem of **multi-task reinforcement learning** (MTRL). An MTRL problem is defined by a distribution D over a potentially infinite set of MMDPs. Given an MTRL problem a multi-task learner L is provided with a potentially infinite random sequence of MMDPs $M_1, M_2, \ldots$ drawn i.i.d. from D and is allowed to act in it.

Intuitively our objective is to develop a multi-task learning algorithm that is able to leverage experience in previous MMDPs $M_1, \ldots M_n$ to more quickly learn an optimal policy in a newly drawn MMDP M_{n+1} compared to a algorithm that ignores the previous MMDPs. Naturally, if the MMDPs drawn from a multi-task distribution are completely unrelated, a multi-task learner will have no advantage over simply solving each MMDP in isolation. However, if the MMDPs share common structure, a multi-task learner might be able to learn and exploit this structure, to outperform learners that solve each MMDP independently.

In our work, we consider a class of MTRL problems over MMDPs, where the multiple agents to be controlled have a shared role structure and units of the same role have similar policies. In particular, we assume that the units in each MMDP are associated with observable unit features that reflect intrinsic and contextual properties of a unit and that these features are related to the role assignment of a unit and hence its policy. Our goal is to learn a model from previous tasks that captures the underlying set of unit roles, along with information about how unit features are related to role assignments and what types of policies are associated with each role. This model can then be used for future tasks to speed up learning, e.g., by providing a way to intelligently initialize unit policies. In our setting, the number of unit roles is unknown and must be inferred by the multi-task learner. Furthermore it is not assumed that each MMDP contains units from each of the possible roles. Thus, our multi-task learner must remain open to the possibility that the current MMDP has units with roles that were not encountered in prior MMDPs. Our infinite component Dirichlet process model described in the following sections provides a Bayesian framework for dealing with this uncertainty.

Roles in Multi-Agent Reinforcent Learning

We are interested in cooperative multiagent RL where agents can be thought to fulfill particular roles as part of fulfilling global objectives. Such domains include many sports such as soccer or football where players have specific objectives during play. Playing fullback in soccer is a much different role than center forward, which is even more strongly contrasted by the role played by the goalie. Similarly, in football, players follow carefully predesigned plays each having a different role in completing the overall objective. Examples arise naturally in most team sports. Here we take a role based view Wargus; we are interested in direct tactical conflicts in the Wargus domain.

Roles arise naturally in stratagus because the so called "units" in the game, our agents, have different characteristics which determine how apt they are to a particular task. Each unit has a set of attributes measuring such things as their speed, the range and power of their attack, and their durability. Like real people in a sports match these attributes predispose units to particular uses. For instance, Knights in the game are good at attacking ranged units due to their speed, and the catapult is best used to siege powerful defensive structures thanks to its extensive range. Other features such as the units inital locations and distances to targets introduce contextual attributes which also influence which role an agent elects to take. For instance, nearness can determine which initial targets are the best.

In the cooperative multiagent setting each agent follows a parameterized policy π which it optimizes independent of the other agents decisions. We represent the joint probabilistic policy as the product of individual agent policies as given by Equation 1.

$$P(A|s) = \prod_u P(a_u|g_u(s), w_u) \quad (1)$$

There are a number of ways to parameterize each agents policy. We choose the simple representation shown in Equation 2. Here $g_u(s, a)$ are a set of features for state s and action a, and w_u are the policy parameters.

$$P(a_u|g_u(s), w_u) = \frac{exp(w_u \cdot g_u(s, a))}{\sum_{a'} exp(w_u \cdot g_u(s, a'))} \quad (2)$$

Several issues arise. First, as proposed above we expect units to optimize their behavior towards specific and independent roles. Given a set of learned policies π_u and features representing the agents attributes and initial context, f_u, $\{(\pi_u, f_u)\}$, we want to discover a set of roles that describe the policies, and transfer to new scenarios. For instance, that there are roles for destroying particular mobile units like archers, and roles for sieging large defensive structures like towers. Second, we want to learn how to assign appropriate units (or players) to each role. Fast mobile ranged units should be used to handle knights, very long range units should be used to kill defensive structures like towers, and so on. We describe how this is done in the next section.

Learning Role-Based Distributions over Policies

After learning in the training scenerio we have a collection of data $\{(\pi_u, f_u)\}$ where each π_u are the converged policy parameters for agent u, and each f_u is a set of unit features for u. Each f_u includes unit attributes and contextual features based on the initial state (such as distances to targets). The goal of our inference procedure will be to discover a small set of roles which explain the set of observed policies. For each policy we encode to which role it belongs in a variable c_u. We say that an agent fulfilled role j, (defense, siege, etc.) when $c_u = j$. All knowledge about roles is stored in two sets of parameters: 1) A set of parameters θ_j where θ_j is the parameters over policies for the j'th role, and 2) A set of parameters ϕ that determines the probabilistic assignment of agents to roles based on unit parameters. Parameters θ_j are a mean vector μ and covariance matrix Σ of a multi variate Gaussian distribution. We define a Gaussian Wishart prior distribution from which all θ_j are drawn. The parameters ϕ are assumed to have a Gaussian prior distribution. Furthermore, we require that the function of ϕ has knowledge about which unit features strongly influence role assignment. For instance, with two underlying roles, siege and defense, we would like this function to learn that the unit feature "range" strongly determines whether a unit should be used for siege or defense. Knowledge of these important agent similarities are stored in a kernel function parameterized by assignment parameters ϕ.

We presume that agent roles are drawn from an infinite mixture of Gaussian components. Similar to our previous work we represent this mixture as a Dirichlet Process (DP) (Neal 2000). However, the key difference between the standard is the introduction of the assignment parameters ϕ. In the standard DP model the probability with which a role belongs to a particular component depends only on the number of data points already assigned to the component. In this work this distribution will depend on f_u and ϕ; a necessary change for effective transfer. Below we introduce our modifications and outline the inference algorithm.

Similar to (Rasmussen and Ghahramani 2002) we seek to represent an infinite mixture of components using a DP model, but our component distributions, which generate policies $P(\pi|\mathbf{c}, \mathbf{f}, \theta)$, are represented as conditional Gaussians with conjugate prior distributions (described above). Also like (Rasmussen and Ghahramani 2002), we define the assignment function $P(\mathbf{c}|\mathbf{f}, \phi)$ by specifying a kernel which encodes the similarity between unit types and contexts (explained below).

$$P(\boldsymbol{\pi}|\mathbf{f}, \Psi) = \sum_{\mathbf{c}} P(\boldsymbol{\pi}|\mathbf{c}, \mathbf{f}, \theta) P(\mathbf{c}|\mathbf{f}, \phi)$$

$$P(\boldsymbol{\pi}|\mathbf{f}, \Psi) = \sum_{\mathbf{c}} [\prod_u P(\pi_u|c_u, f_u, \theta_{c_u}) P(\theta_{c_u})] P(\mathbf{c}|\mathbf{f}, \phi) \quad (3)$$

In Equation 3 π_u is the policy associated with unit u, c_u is a role assignment for unit u's policy, and f_u are a set of unit features. Our goal is to estimate the posterior distribution over the parameters $\{\mathbf{c}, \theta, \phi\}$. To estimate the posterior distribution given a set of $\{(\pi_u, f_u)\}$ pairs we construct a sampling algorithm using a Gibbs sampling procedure described in (Neal 2000) for the parameters $\{\mathbf{c}, \theta\}$, and metropolis updates for the assignment parameters $\{\phi\}$. The requirement for the Gibbs sampling procedure are conditional distributions for the parameters $\mathbf{c}$ and θ. In our case, the conditional distributions for θ are simple (see the algorithm outlined below). However, we have to modify the standard conditional distribution over the role assignment variables.

Sampling Role Assignments: In algorithm 8 of (Neal 2000) the role assignment variables for each unit, c_u, are sampled from the standard conditional distribution Equation 4 which depends on the MCMC state $\mathbf{c}$ at each sampling step.

$$P(c_u = j|\mathbf{c}_{-u}, \alpha) = \begin{cases} \frac{n_{-u,j}}{n-1+\alpha} & |n_{-u,j} > 0 \\ \frac{\alpha}{n-1+\alpha} & |\text{otherwise} \end{cases} \quad (4)$$

Thus, in the standard DP Gibbs sampling framework the conditional probability of the assignment $c_u = j$ depends principally on the number of policies currently assigned to component j after removing u, i.e., $n_{-u,j}$. In this equation it is worth noting that with probability $\frac{\alpha}{n-1+\alpha}$ the algorithm samples a completely new component. In this way a properly constructed sampling procedure can stochastically change the number of inferred components. The dependence solely on the $\mathbf{c}$, and α means that sampling from this distribution will be independent of the available unit features f_u. However, the corresponding conditional distribution in Equation 3, $P(\mathbf{c}|\mathbf{f}, \phi)$, depends on both the unit features and assignment parameters. To make Equation 4 depend on the unit features and assignment parameters we modify the conditional distribution by approximating $n_{-u,j}$ using a parameterized kernel K_ϕ. The approximation is defined in Equation 5.

$$n_{-u,j} = (n-1)\frac{\sum_{u' \neq u} K_\phi(f_u, f_{u'})\delta(c_{u'} = j)}{\sum_{u' \neq u} K_\phi(f_u, f_{u'})} \quad (5)$$

In Equation 5 the numerator determines the weighted sum of distances between the unit in question and all units in role j and divides by the sum of distances between u and all other units. By approximating $n_{-u,j}$ in this way we insure that the conditional distribution, Equation 4, biases the assignment of c_u to roles fulfilled by units similar to u. We choose to use the kernel below which allows the role assignment parameters to determine the relevance of features.

$$K_\phi(f_u, f_{u'}) = exp(-\frac{1}{2}\sum_d (f_{u,d} - f_{u',d})^2/\phi_d^2) \quad (6)$$

This kernel measures the distance between unit feature vectors and weights the relevance of individual features to the role classification problem. We choose this kernel because it can decide when contextual features are irrelevant, and when differences in innate features such as 'range' strongly influence which role it belongs to.

We can use this modified conditional distribution $P(c_u|f_u, \phi, \mathbf{c}_{-u}, \alpha)$ to sample new assignments from $P(\mathbf{c}|\mathbf{f}, \phi)$ in the Gibbs sampling algorithm as is described below.

Full Sampling Algorithm: Given conditional distributions for the assignments c_u, and role parameters θ_j we can define a sampling algorithm for posterior inference. The inference algorithm is outlined below:

1. Define the initial MCMC state to be $\{\mathbf{c}, \theta, \phi\}$. Set initial values for these parameters.
2. Update $\mathbf{c}$: Sample an assignment for each unit policy π_u. For each unit $u = 1 : N$ sample from the conditional distribution $P(c_u = j|f_u, \phi, \mathbf{c}_{-u}, \alpha)$. Update the state $\mathbf{c}$ with the new samples (Neal 2000). The new assignment $\mathbf{c}$ is now drawn from $P(\mathbf{c}|\mathbf{f}, \phi)$.
3. Update θ: Sample new parameters θ for each Gaussian component from the posterior $P(\theta_j = (\mu_j, \Sigma_j)|\mathbf{c}, \boldsymbol{\pi})$ using only the policies assigned to component j and update θ (Neal 2000). This is the probability of the jth role parameters given the policies assigned to j. Given fixed role assignment variables $\mathbf{c}$ the component posteriors are independent Gaussian Wishart distributions for which standard sampling algorithms exist.
4. Update ϕ: Use metropolis hastings to update the parameters ϕ given the new state for $\{\mathbf{c}, \theta\}$ using a Gaussian proposal distribution (Rasmussen and Ghahramani 2002). When sampling from the pseudo posterior the kernel parameters play a role in determining the values of the pseudo likelihood.
5. Repeat the sampling routine (steps 2,3,and 4) until samples are drawn from the posterior and then select the sample maximizing the pseudolikelihood $max_c[\prod_u P(\pi_u|c_u, f_u, \theta_{c_u})P(\theta_{c_u})]$.

Additional details for the inference procedure can be found in (Neal 2000; Rasmussen and Ghahramani 2002).

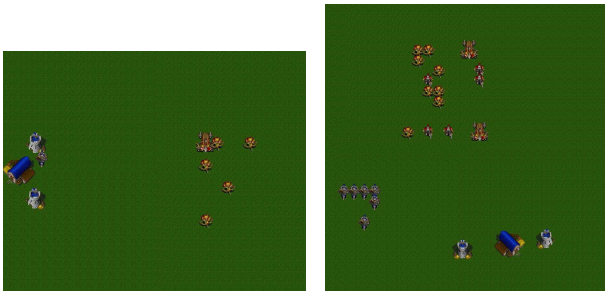
The state $\{\mathbf{c}, \theta, \phi\}$, the fixed prior parameters, and the data are transferred to new tasks. What has been learned is a hierarchical prior distribution over policy parameters that can be used to initialize policies of units in new scenarios.

Role Transfer

Given the posterior sample, $\{\mathbf{c}, \theta, \phi\}$, transfer is a simple matter. Each units policy parameters are sampled, independently, from the new prior distribution. The respective set of policies is then used as a starting point for learning. To sample a policy for an individual agent u we simply sample from the conditional distribution $P(c_{new}|f_{new}, \phi)$ and then generate a policy from the gaussian component associated with label c_{new}, $P(\pi_{new}|c_{new}, f_{new}, \theta_{c_{new}})$. Policies represent the best guess as to the mapping from unit features f_{new} to optimal policies. This highlights how crucial the assignment parameters ϕ are to successful initialization. Without the additional feature information initialization would be matter strictly of popularity. In general the sampled policies may not be optimal for the current task and must be optimized. Each agents behavior is optimized using OLPOMDP an online policy gradient optimization technique (Baxter and Bartlett 1999) using the sampled policy as an initial starting point.

Results

Experiments were run in the Wargus environment. The goal in each scenario is to destroy an enemy target structure, a lumbermill, guarded by a number of enemy forces. In each experiment enemy forces are composed of immobile towers capable of firing on units from a distance, and knights which are strong melee combatants. The agents forces are composed of a mixture of units including archers, footmen, and ballistas. Each agent receives its own reward signal. Reward is recieved for killing another unit. Positive reward for killing an enemy unit and negative reward for killing a friendly unit. Destroying the lumbermill results



(a) Training map (b) Test map 2

Figure 1: a. Training map used in all experiments. b. Second test map: 8 archers, 5 footmen, 2 ballista

in a very large positive reward and represents the completion of an episode (alternatively an episode ends when all friendly units have died). Negative reward is also recieved once the agent dies. Each agents action set includes all live targets on the map including friendly units. Each action has a set of features which includes information about the target (speed, range of attack, current hit points, etc), coordination features (how many units are currently attacking this target, how close are units to this target, etc), and relational features (how far is the agent from the target, etc.). The feature vector has 180 dimensions.

In both experiments we transfer knowledge learned in a single training map (see Figure 1(a)). In the training map the enemy forces are composed of two towers, a knight, and a lumbermill. The agents forces include five archers, and a ballista. The towers are powerful enough to kill the archers, but do not have a range sufficient to hit the ballista. The knight can easily dispatch the ballista (their preferred target), and must be destroyed by the archers for the agents to succeed in destroying the lumbermill. Both the archers and ballista fulfill seperate roles to successfully storm the enemy blockade.

In experiment one we first train the units in the training map and then transfer to a more complicated scenerio. In the new scenerio the number of opposing knights has been increased to two and number of archers on the agents team is increased to 8. Individually the knights are more than a match for the archers. If the archers do not present a unified front they will be quickly overwhelmed. Other unit types remain the same. As verified in Figure 2 the transfer learning algorithm has discovered a set of roles for the underlying set of agent types and initializing with good sampled values gives a substantial jumpstart for learning. The archers are initialized to be biased towards attacking the knights, and the ballista prefers to target the immobile structures. The resulting behavior is that the archers successfully gang up on the knights killing them before they can destroy the ballista, and the ballista moves into position to kill off the unprotected structures.

In experiment two the test map is complicated by a much larger number of archers (8), the number of available ballistas is increased to 2 and we introduce a new unit type footmen (5) (see Figure 1(b)). Additionally, the agents face 6 knights defending the target structure, as well as two towers. Figure 3 illustrates the results. The transfer learning problem is more difficult in this case because the algorithm must generalize across unit types assigning appropriate roles to the footmen for which it has no prior experience. Though the footmen cannot attack at range their other characteristics are much closer to archers than they are to ballistas. Footmen are more mobile than ballistas, attack more frequently, and are more fragile than the ballista. As a result the algorithm guesses that they should be assigned a role similar to archers, aiming to kill the defending knights. Making this generalization is key to success in this scenario because the large number of knights can easily destroy the archers if they do not have support. The generalization is successful, and the transfer algorithm has a large jumpstart in the new map.

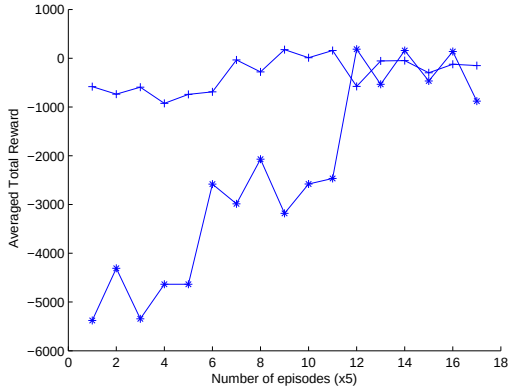


Figure 2: Transfer map 1: Transfer task 8 archers, 1 balissta vs. 3 knights, 2 towers, 1 lumbermill.

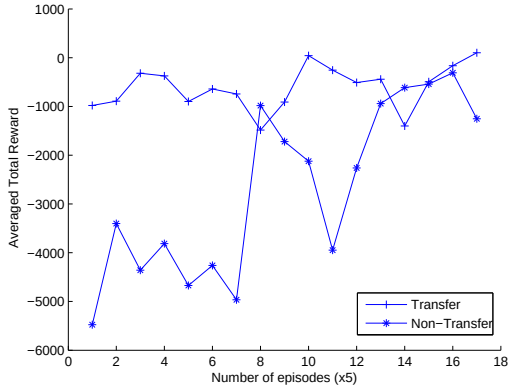


Figure 3: Transfer map 2: Transfer task 8 archers, 5 footmen, 2 balissta vs. 6 knights, 2 towers, 1 lumbermill.

Discussion

The model does two things successfully. First, it automatically discovers the number and kind of underlying roles that agents fulfill. The model allows us to remain agnostic about the types of roles that we might find. It is not immediately obvious that this is necessary in the Wargus environment. Wargus agents might, uninterestingly, fall into a role completely determined by the unit type. In our relatively simple experiments the algorithm found more roles than there were unit types in the training set reinforcing the utility of remaining agnostic of the underlying set of roles. This has the additional benefit of allowing the library of roles to grow as the agent has experiences in new maps. Second, the parameterized kernel allows the assignment function to generalize across the unit parameter space. With its experience limited to the policies of archers and catapults our algorithm learned a strong bias towards using the catapults to attack fortifications and using archers to overcome mobile defenders. Generalizing the archer role to footmen could not have been done properly without the similarity metric.

There are two crucial aspects of the algorithm that require additional attention. First, currently each agents policy is drawn independently from the generative model. Agents do not explicitly coordinate. Second, given a sampled policy we optimize the agents behavior using olpomdp.

The first limitation, independent action selection, limits success to domains where a carefully designed reward function can be constructed to encourage cooperative play and requires carefully encoded action features important for assisting coordination (one such feature is how many agents are currently attacking a target). When a good encoding of the reward function is not available or the feature set is not obvious some form of explicit communication is necessary.

In the Wargus experiments we are interested in optimal coordination at the action level is not possible due to the large number of agents. Potentially this problem could be reduced by considering coordinating at the role level. By jointly considering an assignment of roles we need only consider $|R|^{|A|}$ such role combinations for $|A|$ agents. This can be a considerable savings. In our Wargus domain the action set grows with the number of opposing targets which was as large as nine in the second experiment. Whereas the number of discovered roles was only three. Still there remains an exponential number of potential assignments and it would be worthwhile, in future work, to find additional structure which can be exploited to approximate the optimal assignment.

Our second issue is limited application of the revised prior to perform initialization in a new environment. This does well in our experiments, but we would like to exploit the available information during exploration of new maps. Such a policy prior could be used to both guide a sequence of experiments, sampled policies from the posterior distribution, while simultaneously optimizing the expected returns. To do so we need a distribution characterizing the influence of the policy parameters on the expected return. Ideally such a distribution would encode the relationship between similar policies so that discovering the true optimal does not require evaluating all policies in the set. These requirements suggest that a Gaussian Process prior may be a natural candidate for capturing the functional relationship between policies. We would like to explore composing our policy prior with a learned mapping from policies to expected returns to do online policy search using the bayesian framework. Such a policy search algorithm would replace OLPOMDP so that the learning algorithm continues to exploit the prior distribution as it explores.

Conclusion

We have introduced a simple bayesian framework for role transfer in multi-agent multi-task reinforcement learning. Our approach is to learn a hierarchical bayesian prior distribution and use this learned distribution to initialize the agents roles in new environments. Our algorithm successfully learns individual agent policies, automatically discovers a good set of roles describing the space of learned policies, and initializes agent roles based on unit features.

We provide a successful demonstration of the proposed transfer algorithm in large Wargus conflicts. Up to 15 units, when initialized using the learned prior successfully perform the target task of destroying the lumbermill. We show that our algorithm discovers a useful set of roles and successfully transfers role information to new Wargus scenarios. The first experiment shows transfer is successful when the number of units in the cooperating forces increases. The algorithm can find roles that are specific to the unit types experienced in the training scenario. The second experiment illustrates generalization across unit types. The expert function correctly identifies footman units as being similar to archer units. Initializing based on that assumption leads to a substantial jumpstart on learning.

References

Baxter, J., and Bartlett, P. 1999. Direct gradient-based reinforcement learning. In *Technical report, Research School of Information Sciences and Engineering, Australian National University, July 1999.*

Marsella, S.; Adibi, J.; Al-Onaizan, Y.; Kaminka, G. A.; Muslea, I.; and Tambe, M. 1999. On being a teammate: experiences acquired in the design of RoboCop teams. In Etzioni, O.; Müller, J. P.; and Bradshaw, J. M., eds., *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, 221–227. Seattle, WA, USA: ACM Press.

Martinson, E., and Arkin, R. C. 2003. Learning to role-switch in multi-robot systems. In *ICRA*, 2727–2734. IEEE.

Neal, R. M. 2000. Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics* 249– 265.

Rasmussen, C. E., and Ghahramani, Z. 2002. Infinite mixtures of gaussian process experts. In *Proceedings Neural Information Processing Systems*.

Stone, P., and Veloso, M. 1999. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*.