Multiagent Transfer Learning via Assignment-based Decomposition

Scott Proper
Scool of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331-3202, USA
Email: proper@eecs.oregonstate.edu

Prasad Tadepalli
Scool of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331-3202, USA
Email: tadepall@eecs.oregonstate.edu

***Abstract*—We describe a system that successfully transfers value function knowledge across multiple subdomains of real-time strategy games in the context of multiagent reinforcement learning. First, we implement an assignment-based decomposition architecture, which decomposes the problem of coordinating multiple agents into the two levels of *task assignment* and *task execution*. Second, a hybrid model-based approach allows us to use simple deterministic action models while relying on sampling for the opponents' actions. Third, value functions based on parameterized relational templates enable transfer across sub-domains with different numbers of agents.**

***Keywords*-reinforcement learning; markov decision processes; assignment problem; coordination; transfer learning**

I. Introduction

Most real world learning and planning problems such as fire and emergency response in a city, vehicle routing and product delivery, and real-time strategy games need to represent and reason with multiple objects, agents, tasks, and the relations between them. Such representational richness is both a blessing and a curse. It is a blessing because the knowledge may be expressed and learned in a form that generalizes naturally and transfers easily across domains with different numbers of objects of different types. Unfortunately, having to reason about multiple objects of different types and relations between them raises an enormous scaling problem for planning and learning. In this paper, we show how we can exploit the representational richness of these domains for transfer across multiple subdomains, while taming the complexity of planning and learning in a richer representation through a judicious mixture of architectural and algorithmic choices.

Multiagent domains like real-time strategy games face multiple scaling problems or "curses of dimensionality" [1], [2]. First, both the number of states and the number of actions are exponential in the number of objects such as computer characters or units in the game. Second, the action choices of the opponent makes the number of possible outcomes for a given state-action pair exponential in the number of enemy agents.

In the assignment-based decomposition (ABD) architecture, the action space is divided into a task assignment level and a task execution level. At the task assignment level, each task is assigned a group of agents. At the task execution level, each agent is assigned primitive actions to perform based on its task. The current approach extends the model-free ABD architecture of [3] to a hybrid model-based and model-free architecture and replaces the hill climbing search for the task assignment with a more powerful bipartite graph search. Since the task execution is mostly local, we learn a low-dimensional relational template-based value function for it. While this two-level decomposition resembles that of hierarchical multiagent reinforcement learning, e.g., [4], in contrast, we do not store a value function at the root level which must be over the joint state-space over all the agents in the worst case, and hence intractable.

Our hybrid model-based and model-free approach addresses the problem of exponential outcome space by learning an afterstate-based value function that takes into account the effects of the actions of all the friendly agents but not the enemy agents. The advantage of this hybrid approach is that unlike the full model-based approach, it does not need to learn or use the models for the actions of the opponent. Like Q-learning, it lets the distribution of the next states be sampled by the actions of the enemy agent, and use these samples to update the value of the afterstate. Since our own agents' actions are usually deterministic, by modeling them, it prevents the use of action-based value function as in Q-learning, which does not scale when the action space is exponentially large.

Finally, the use of relational templates or parameterized tables to approximate the lower level value function helps mitigate the scaling problem [5]. More importantly, it enables transfer between domains of different sizes simply by appropriately initializing the value function of one domain from the value function of a previously learned domain in a manner similar to the *behavior transfer* of [6]. It also generalizes the work on class-based value functions of [7] to tuples of objects which are characterized by a set of relational features. Our approach transfers the value function knowledge from 3 to 6 and then 12 agents in a real-time strategy game on a 10×10 grid, a feat that was not successfully demonstrated until now.

The rest of the paper is organized as follows. Section 2 introduces the multiagent assignment MDP (MAMDP)

framework. Section 3 describes our hybrid value function learning approach to solving MAMDPs. Section 4 describes the value function approximation based on relational templates. Section 5 describes the results of transfer learning in different sub-domains of real-time strategy games with different numbers of agents and tasks, which is followed by Section 6 that concludes the paper.

II. Multiagent Assignment Markov Decision Processes

In this section, we first introduce Markov Decision Processes (MDPs), and generalize them to multiagent settings.

We assume that the learner's environment is modeled by a Markov Decision Process (MDP), defined by a 4-tuple $\langle S, A, p, r\rangle$, where S is a discrete set of states, and A is a discrete set of actions. Action u in a given state $s \in S$ results in state s' with some fixed probability $p(s'|s,u)$ and a finite immediate reward $r(s,u)$. A *policy* μ is a mapping from states to actions. Here, we seek to optimize the total expected reward received until the end of the episode starting from state s. This is denoted by $v(s)$ and satisfies the following Bellman equation:

$$v(s) = \max_{u \in A} \left\{ r(s,u) + \sum_{s' \in S} p(s'|s,u) v(s') \right\} \quad (1)$$

The optimal policy chooses actions maximizing the right hand side of this equation. We can use the above Bellman equation to update $v(s)$. However this is often computationally expensive because of high stochasticity in the domain. Thus, we want to replace this with a sample update as in model-free reinforcement learning. To do this, we base our Bellman equation on the "afterstate," which incorporates the deterministic effects of the action on the state but not the stochastic effects [8], [2]. Since conceptually afterstate can be treated as the state-action pair, it strictly generalizes model-free learning. We can view the progression of states/afterstates as $s \xrightarrow{a} s_a \rightarrow s' \xrightarrow{a'} s'_{a'} \rightarrow s''$. The "$a$" suffix used here indicates that s_a is the afterstate of state s and action a. The stochastic effects of the environment create state s' from afterstate s_a with probability $p(s'|s_a)$. The agent chooses action a' leading to afterstate $s'_{a'}$ and receiving reward $r(s', a')$. The environment again stochastically selects a state, and so on. This variation of afterstate total-reward learning is called "ATR-learning" [5]. We define the afterstate-based value function of ATR-learning as $av(s_a)$, which satisfies the following Bellman equation:

$$av(s_a) = \sum_{s' \in S} \left\{ p(s'|s_a) \left[\max_{u \in A} \{ r(s',u) + av(s'_u) \} \right] \right\} \quad (2)$$

We use sampling to avoid the expensive calculation of the expectation above. At every step, the ATR-learning algorithm updates the parameters of the value function in the direction of reducing the temporal difference error (TDE), i.e., the difference between the r.h.s. and the l.h.s. of the above Bellman equation:

$$TDE(s_a) = \max_{u \in A} \{ r(s',u) + av(s'_u) \} - av(s_a) \quad (3)$$

A *Multiagent Assignment MDP* (MAMDP) extends the above framework to a set of n agents $G = \{g\}$ ($|G| = n$). Each agent g has its own set of local state S_g and actions A_g. We also define a set of tasks $T = \{t\}$, each associated with a set of state variables S_t that describe the task. The set of tasks (and corresponding state variables required to describe them) may vary between states. The joint action space is the Cartesian product of the actions of all n agents: $\mathbf{A} = A_1 \times A_2 \times ... \times A_n$. The joint state space is the Cartesian product of the states of all agents and all tasks. The reward is decomposed between all n agents, i.e., $R(s,a) = \sum_i^n R_i(\mathbf{s}, \mathbf{a})$, where $R_i(\mathbf{s}, \mathbf{a})$ is the agent-specific reward for state $\mathbf{s}$ and action $\mathbf{a}$.

$\beta : T \rightarrow G^k$ is an assignment of agents to tasks; here k indicates an upper bound on the number of agents that may be assigned to a particular task. $\beta(t)$ indicates the set of agents assigned to task t. We let $s_{\beta(t)}$ denote those state variables describing task t and joint states of all agents assigned to t, and $a_{\beta(t)}$ denote the joint actions of all agents assigned to task t. $s_{a_{\beta(t)}}$ indicates the task-specific afterstate of $s_{\beta(t)}$ and $a_{\beta(t)}$. The total expected utility $v(\mathbf{s})$ depends on the states of all tasks and agents in $\mathbf{s}$.

III. Model-based Assignment-based Decomposition of MAMDPs

Assignment-based decomposition has previously been implemented in a model-free setting [3]. In this section, we describe a model-based implementation, which has considerable advantages. In addition to requiring many fewer parameters than model-free methods, the time required to calculate the assignment is greatly reduced, as we will show.

Assignment-based decompositions splits the action selection step of a reinforcement learning algorithm into two levels: the upper assignment level, and the lower task execution level. At the assignment level, agents are assigned to tasks. Once the assignment decision is made, the lower level action that each agent should take to complete its assigned task is decided by reinforcement learning in a smaller state space. This two-level decision making process occurs each time-step of the algorithm, taking advantage of the opportunistic reassignments.

At the assignment level, we ignore interactions between the agents assigned to different tasks. This action decomposition exponentially reduces the number of possible actions that need to be considered at the lowest level, at a cost of increasing the number of possible assignments that must be considered. Because each agent g need only consider its local state and task s_g to come to a decision, this method can greatly reduce the number of parameters that are necessary to store. This reduction is possible because rather

than storing separate value functions for each possible agent and task combination, we can share a single value function between multiple agent-task assignments.

The value $v(s_\beta(t))$ denotes the maximum expected total reward for a task t and assigned set of agents $\beta(t)$ starting from the joint state $s_{\beta(t)}$ by following their best policy and assuming no interference from other agents. Similarly we define $av(s_{a_{\beta(t)}})$ as the value of the afterstate of s due to the actions $a_{\beta(t)}$. The temporal difference error (TDE) of the afterstate-based value function for an assigned subset of agents using ATR-learning is as follows:

$$TDE(s_{a_{\beta(t)}}) = \max_{u_{\beta(t)} \in A_{\beta(t)}} \left\{ r_{\beta(t)}(s', u_{\beta(t)}) + av(s'_{u_{\beta(t)}}) \right\} - av(s_{a_{\beta(t)}}) \quad (4)$$

The assignment problem described is nontrivial – there are an exponential number of possible assignments in the number of agents. We can search over the space of possible assignments by defining a value $y_{\mathbf{g}}$ of a task and set of agents $\mathbf{g}$. This value is derived from the underlying state-based value function $v(s_{\beta(t)})$:

$$y_{\mathbf{g}} = v(s_{\mathbf{g}}) \quad (5)$$

The value function we use for ATR-learning is based on afterstates, so the value $v(s_{\mathbf{g}})$, being based on states, is learned separately. This is based on the temporal difference error given below:

$$TDE(s_{\beta(t)}) = r_{\beta(t)} + \max_{u_{\beta(t)} \in A_{\beta(t)}} \left\{ r_{\beta(t)}(s', u_{\beta(t)}) + av(s'_{u_{\beta(t)}}) \right\} - v(s_{\beta(t)}) \quad (6)$$

Here $r_{\beta(t)}$ is the immediate reward received for task t and agents $\beta(t)$. This equation may re-use the calculation of the max found in equation 4. This max is the long-term expected total reward for being in afterstate s_a and thereafter executing the optimal policy. This afterstate value does not account for the immediate reward received for being in state s and taking action a, and so we must add it in here.

To find the best assignment of tasks to agents over the long run, we need to compute the assignment that maximizes the sum of the expected total reward until the task completion plus the expected total reward that the agents could collect after that. Unfortunately this leads to a global optimization problem which we want to avoid. So we ignore the rewards after the first task is completed, and try to find the assignment that maximizes the total expected reward accumulated by all agents for that task. It turns out that this approximation is not so drastic, because the agents get to reassess the value of the task assignment after every step and opportunistically exchange tasks.

Fixed assignment: This is the simplest possible option: no assignment search at all. Assignments are arbitrarily set at the start of an episode and never change.

Exhaustive search: One straightforward method that guarantees optimal assignment is to exhaustively search for the mapping β that returns the maximum total value for all tasks $\max_\beta \sum_t y_{\beta(t)}$. However, for many agents, this search could become intractable. A faster approximate search technique is necessary, which we introduce next.

Hill climbing search: A simple means of improving search time is to use a hill climbing approach. We implemented this approach as follows:

- At step 1 of an episode, start with an arbitrary initial assignment β.
- At subsequent steps, let the starting assignment β be the assignment found in the last time step.
- Create a list of all possible agent-agent assignment "swaps" (i.e., switching the tasks of two agents).
- For each swap, if it improves the total value $\sum_t y_{\beta(t)}$, swap the assignments of the corresponding agents.

This simple algorithm is a fast way of improving an existing assignment. Results can be improved by repeating the final two steps multiple times.

Bipartite search: The Hungarian method [9] is a combinatorial optimization algorithm which solves the assignment problem in polynomial time. We adapted this technique for use in solving the assignment problem faced by assignment-based decomposition. We adapt the Hungarian method (or Kuhn-Munkres algorithm as it is sometimes called) to assign multiple agents to each task by copying each task as many times as necessary to match the number of agents (one copy for each "slot" available to agents for completing a task). This creates an $n \times n$ matrix defining a bipartite graph, which is solved by the Hungarian method in polynomial time. The weight of each edge of the graph is given by y_g, where g is a single task and agent. The solution to the bipartite graph consists of an assignment of each task to a set of agents.

A serious problem with this approach is that each edge of the graph, or entry in the $n \times n$ matrix, cannot contain any information other than that pertaining to the single edge and task of that edge. In other words, we must give up any coordination information when making our assignment decisions. While this could cause some serious sub-optimalities in principle, as we will show in the next section, in practice the approximately optimal assignment found by this method performs very well.

IV. Relational Templates

We use relational templates [5] to create the function approximator used in our experimetns. Relational templates are defined by a set of relational features over shared variables. For example, the template $\langle Distance(A,B), AgentHP(B), TaskHP(A), UnitsInrange(B) \rangle$ contains relational features indicating distance between an agent and task, agent and enemy unit hit points, and a count of the number of agents in range of a task. Each template is instantiated in a state by binding

its variables to units of the correct type. An instantiated template i defines a table θ_i indexed by the values of its features in the current state. In general, each template may give rise to multiple instantiations in the same state. The value $v(s)$ of a state s is the sum of the values represented by all instantiations of all templates.

$$v(s) = \sum_{i=1}^{n} \sum_{\sigma \in \mathcal{I}(i,s)} \theta_i(f_{i,1}(s,\sigma), \ldots, f_{i,m_i}(s,\sigma)) \quad (7)$$

where i is a particular template, $\mathcal{I}(i,s)$ is the set of possible instantiations of i in state s, and σ is a particular instantiation of i that binds the variables of the template to units in the state. The relational features $f_{i,1}(s,\sigma), \ldots, f_{i,m_i}(s,\sigma)$ map state s and instantiation σ to discrete values which index into the table θ_i. All instantiations of each template i share the same table θ_i, which is updated for each σ using the following equation:

$$\begin{aligned} &\theta_i(f_{i,1}(s,\sigma), \ldots, f_{i,m_i}(s,\sigma)) \leftarrow \\ &\theta_i(f_{i,1}(s,\sigma), \ldots, f_{i,m_i}(s,\sigma)) + \alpha(TDE(s,\sigma)) \end{aligned} \quad (8)$$

where α is the learning rate.

V. Experimental Results

We performed all experiments on several variations of a real-time strategy game (RTS) simulation first introduced in [5]. The RTS simulation operates on a 10x10 gridworld. The grid is presumed to be a coarse discretization of a real battlefield, and so units are permitted to share spaces. The number of starting agents varied from 3-12, and the number of enemy towers (tasks) varied from 1-4. Units, either enemy or friendly, were defined by several features: position (in x and y coordinates, hit points (0-6), and type (either archer or tower) We also defined relational features such as distance between agents and the assigned enemy unit, and aggregation features such as a count of the number of opposing units within range. In addition, each unit type was defined by how many starting hit points it had, how much damage it did, the range of its attack (in manhattan distance), and whether it was mobile or not. Enemy towers are more powerful than the agent-controlled archers, and so require coordination to defeat.

Agents had six actions available in each time step: move in one of the four cardinal directions, wait, or attack its assigned target (if in range). Towers cannot move, but may choose whom to attack. Towers followed predefined policies, attacking the unit closest to death and within range. An attack at a unit within range always hits, inflicting damage to that unit and killing it if it is reduced to 0 hit points. Thus, the number of agents (and tasks) are reduced over time. Eventually, one side or the other is wiped out, and the battle is "won". We also impose a time limit of 20 steps. Due to the episodic nature of this domain, total reward reinforcement learning is suitable. We gave a reward of $+1$ for a successful kill of an enemy unit, a reward of -1 if an agent is killed, and a reward of $-.1$ each time step to encourage swift completion of tasks. Thus, to receive positive reward, it is necessary for agents to coordinate with each other to quickly kill enemy units without any losses of their own.

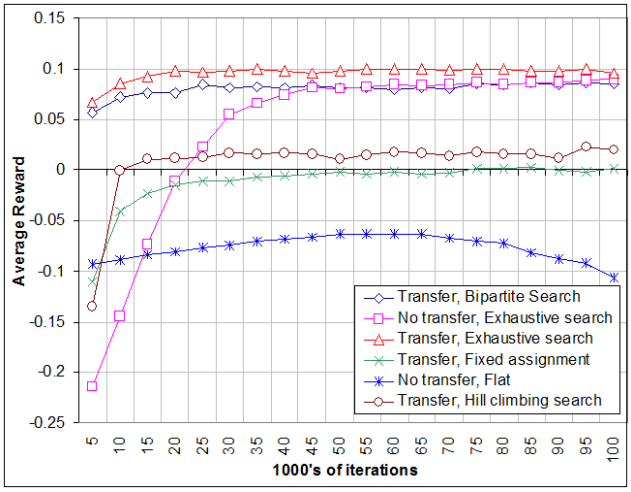

Figure 1. Comparison of 6 agents vs 2 task domains.

In [5], we studied the effectiveness of relational templates for transferring knowledge between different subdomains of three agents and one enemy unit. Transfer learning across subdomains is very helpful, but transfer learning may also provide an additional benefit when combined with assignment-based decomposition: we can transfer knowledge learned in a small domain (such as the 3 vs 1 domains discussed in [5]) to a larger domain, such as the 6 vs. 2 or 12 vs. 4 domains discussed here.

To transfer results from the 3 vs. 1 domain to the 6 vs. 2 domain, we must use assignment-based decomposition within the larger domain. This domain has two enemy units (tasks) and six agents. Each time step, we assign agents to tasks using any of a variety of search algorithms, and allow the task execution level to decide how each group of agents should complete its single assigned task. Thus, we can adapt the relational templates used to solve the 3 vs. 1 domain to this larger problem.

If we adapt the templates used in the 3 vs. 1 domain directly (one of which is shown in section IV), performance will suffer due to interference (being shot at) by enemy units other than those assigned to each agent. To prevent this, we create a new aggregation feature $TasksInrange(B)$ (which counts the number of enemies in range of agent B), and define a *behavior transfer function* [6] $\rho(\pi)$ which initializes the new relational templates which include this feature by transforming the old templates which do not. We do this simply by copying the parameters of the old templates into those of the new for all possible values of the additional dimensions. We also used the template $\langle UnitX(A), UnitY(A), UnitX(B), UnitY(B) \rangle$ to allow

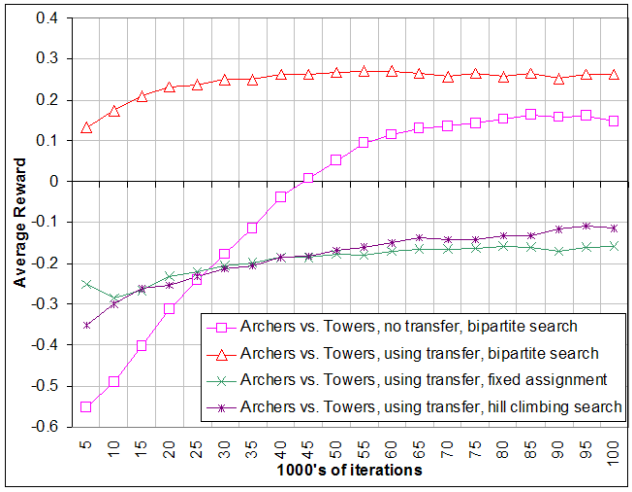

Figure 2. Comparison of 12 agents vs 4 task domains.

Table I
EXPERIMENT DATA AND RUN TIMES.

Size	Transfer	Search type	Seconds
3 vs 1	no	flat	28
3 vs 1	yes	flat	29
6 vs 2	no	exhaustive	34
6 vs 2	yes	exhaustive	60
6 vs 2	yes	bipartite	60
6 vs 2	yes	hill climbing	60
6 vs 2	yes	fixed	57
6 vs 2	no	flat	2567
12 vs 4	no	bipartite	76
12 vs 4	yes	bipartite	122
12 vs 4	yes	hill climbing	156
12 vs 4	yes	fixed	114
12 vs 4	yes	bipartite	108

coordination between pairs of agents.

We have an additional consideration when transferring from the 3 vs. 1 to 6 vs 2 domains: as we are using assignment-based decomposition in the 6 vs. 2 domains, can we (or should we?) transfer the state-based value functions? While this is possible to do (by learning the function in the 3 vs. 1 domain) empirical results have shown that it is not necessary, and performance suffers very little if we learn the state-based value function from scratch each time, and so this is what we have done for all results in this paper.

All experiments shown here (Figures 1 and 2) are averaged over 30 runs of 10^5 steps each, averaging the results of each run together. We used the ATR-learning algorithm with assignment-based decomposition for most of the experiments. Runs were divided into 40 alternating train/test phases, with $\epsilon = .1$ or $\epsilon = 0$ respectively. For the 6 vs. 2 domain, we used $\alpha = .1$. For the 12 vs. 4 domain, we used $\alpha = .01$.

We compared results with and without transfer from the 3 vs. 1 domain to the 6 Archers vs. 2 Towers domain. These results show that using transfer is significantly better than not using it at all. In addition, we tested several different forms of assignment search: exhaustive, hill climbing, bipartite, and fixed assignment. As expected, fixed assignment performs quite poorly. Bipartite search, while performing slightly worse than exhaustive, still does very well. The performance of hill climbing varies between that of fixed assignment and bipartite search, depending on how many times the hill climbing algorithm is used to improve the assignment. Shown are results for only one iteration of the hill climbing algorithm, which is only a modest improvement upon fixed assignment.

We also tested the "flat" (no assignment-based decomposition) 6 vs. 2 domain without transfer learning. As expected, this performed very poorly, which is due to the difficulty of creating an adequate function approximator for 6 agents and 2 tasks. We arrived at using only a single, simplified, relational template – $\langle Distance(A,B), AgentHP(B), TaskHP(A), UnitsInrange(B), TasksInrange(B)\rangle$ – after experimentation with several other alternative templates. Even with α set very low (.008) parameters of the value function slowly continue to grow, causing performance to peak and eventually dip. This points to the inadequacy of traditional methods to solve such a large problem: we need to decompose problems of this size in order to attempt to solve them. "Flat" ATR-learning is also very slow on such problems, taking almost 43 times more computation time to finish a single run than the decomposed versions! (Table I)

Our tests on the 12 vs. 4 domain have similar results. Here, we transferred from the 6 vs. 2 domain, which requires no additional relational features. Results (Figure 2) show that using transfer provides an enormous benefit. Most 12 vs. 4 results use bipartite search (as an exhaustive search of the assignment space is unacceptably slow) and this performs very well, especially compared to no assignment search at all.

Finally, we examine the performance of the various algorithm/domain combinations (Table I). The columns list domain size, use of transfer learning, assignment search type ("flat" indicates no assignment search), and average time to complete a single run. From these results, we can see that the computation time required to solve a problem using assignment-based decomposition scales linearly in the number of agents and tasks. This is a considerable improvement over "flat" approaches, which require an exponential amount of time in the number of agents/tasks to solve. As expected, not searching at all is very fast. Exhaustive search is the slowest search technique – so slow as to be unusable in the 12 vs. 4 domain. Bipartite search outperforms hill climbing in both speed and quality. Interestingly, methods that used no transfer learning were faster than those that did. This is most likely because more agents died during these runs,

resulting in less time to compute each time step.

VI. Discussion

We have shown how relational templates and assignment-based decomposition combine to transfer knowledge from a domain with only a few units to domains with many units. Although the addition of one or more relational features may be required, the decomposed value function used in this technique allows a straightforward transfer of knowledge from smaller to larger domains. In combination with fast approximate search techniques such as bipartite search, we can achieve a linear increase in required computation times, as opposed to the exponential amount of time conventional RL approaches require.

An interesting area of possible future work in model-based assignment-based decomposition is the introduction of coordination graphs. Coordination graphs [10], [11] are a means of coordinating multiple agents so as to avoid conflicts and jointly take actions between multiple agents. Coordination graphs are not sufficient to coordinate assignment decisions [3], however they are useful for coordinating between agents at the task-execution level, for example to avoid collisions. The RTS domain introduced here does not model collisions, and so there is no need for low-level coordination between tasks as there is in [3]. Introducing collisions to this RTS domain would be straightforward, and would require adapting the use of coordination graphs to a model-based RL algorithm.

Another possibility for future work is to combine the work on hierarchical RL with assignment-based decomposition. Currently, our assignment search method ignores the calculation of the expected future reward after the current tasks are completed by the corresponding agent teams. One approach to take this into account is to learn an assignment-level value function as in the work of [4]. However, this value function is likely to be highly nonlinear and hard to approximate when a large number of agents are present. Better ways of exploiting the structure of the domain to either search this larger space or represent its value function more compactly would be an interesting research avenue.

Acknowledgments

We gratefully acknowledge the support of the Defense Advanced Research Projects Agency under grant number FA8750-05-2-0249.

References

[1] W. B. Powell and B. Van Roy, "Approximate Dynamic Programming for High-Dimensional Dynamic Resource Allocation Problems," in *Handbook of Learning and Approximate Dynamic Programming*, J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, Eds. Wiley-IEEE Press, Hoboken, NJ, 2004.

[2] S. Proper and P. Tadepalli, "Scaling model-based average-reward reinforcement learning for product delivery." in *ECML '06: Proceedings of the 17th European Conference on Machine Learning*, ser. Lecture Notes in Computer Science, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4212. Springer, 2006, pp. 735–742.

[3] S. Proper and P. Tadepalli, "Solving multiagent assignment markov decision processes." in *AAMAS '09: Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2009, pp. 681–688.

[4] R. Makar, S. Mahadevan, and M. Ghavamzadeh, "Hierarchical multi-agent reinforcement learning," in *AGENTS '01: Proceedings of the 5th International Conference on Autonomous Agents*. Montreal, Canada: ACM Press, 2001, pp. 246–253. [Online]. Available: citeseer.ist.psu.edu/makar01hierarchical.html

[5] S. Proper and P. Tadepalli, "Transfer learning via relational templates." in *ILP 2009: Proceedings of the 19th International Joint Conference on Inductive Logic Programming*, 2009.

[6] M. E. Taylor and P. Stone, "Behavior transfer for value-function-based reinforcement learning," in *AAMAS '05: The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, Eds. New York, NY: ACM Press, July 2005, pp. 53–59.

[7] C. Guestrin, D. Koller, C. Gearhart, and N. Kanodia, "Generalizing plans to new environments in relational mdps," in *IJCAI '03: In International Joint Conference on Artificial Intelligence*, 2003, pp. 1003–1010.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. MIT Press, 1998.

[9] H. Kuhn, "The Hungarian Method for the assignment problem," *Naval Research Logistic Quarterly*, vol. 2, pp. 83–97, 1955.

[10] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *ICML '02: Proceedings of the 19st International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann, July 2002.

[11] X. Zhang, D. Aberdeen, and S. V. N. Vishwanathan, "Conditional random fields for multi-agent reinforcement learning," in *ICML '07: Proceedings of the 24th International Conference on Machine Learning*. New York, NY, USA: ACM, 2007, pp. 1143–1150.