Learning a Rendezvous Task with Dynamic Joint Action Perception

Nancy Fulda and Dan Ventura

***Abstract*— Groups of reinforcement learning agents interacting in a common environment often fail to learn optimal behaviors. Poor performance is particularly common in environments where agents must coordinate with each other to receive rewards and where failed coordination attempts are penalized. This paper studies the effectiveness of the Dynamic Joint Action Perception (DJAP) algorithm on a grid-world rendezvous task with this characteristic. The effects of learning rate, exploration strategy, and training time on algorithm effectiveness are discussed. An analysis of the types of tasks for which DJAP learning is appropriate is also presented.**

I. Introduction

When dealing with multiagent reinforcement learners, compatible individual goals are no guarantee of successful group behavior. Agents frequently settle into suboptimal equilibria: local maxima in the joint reward space. This problem is especially common when a high degree of coordination between agents is required to obtain maximum payoff and failed coordination attempts are penalized. Under such conditions, standard reinforcement learners will often learn to avoid actions that lead to penalties before successful coordination patterns can be established [1], [2].

One common means of addressing this problem is to allow each agent to perceive the action selections of its counterparts, thus allowing it to discriminate between the rewards and penalties received for successful and failed coordination, respectively. This is the basic premise behind joint action learning [1], the Nash Q-learning algorithm [3], and sharing of instantaneous information [4].

The primary drawback of such algorithms is that the size of the joint action space to be learned grows exponentially with the number of agents in the system. This both increases system overhead for storing utility estimates and slows down learning because there is no generalization across the joint action space. In a system with more than two or three agents, this can significantly increase the necessary training time for the algorithm.

The Dynamic Joint Action Perception (DJAP) algorithm addresses this issue of scalability by allowing each agent to dynamically learn which other agents affect its rewards. The DJAP algorithm has been shown to out-perform standard reinforcement learners and nearly match the performance of hand-coded joint action learners on a variant of the matching pennies game [5]. The algorithm has also been examined within the larger context of multiagent learning and has been shown to address the problem of action shadowing discussed in [6]. Action shadowing occurs when individual actions contributing to optimal joint policies appear undesirable to the agent because of the consequences of failed coordination attempts. This paper extends previous research by providing an analysis of the DJAP algorithm's learning capabilities and the effects of several parameters on algorithm performance.

We briefly review the Q-learning framework in Section II and then detail a multi-agent learning task that requires agent coordination and demonstrate its difficulty for existing algorithms in Section III. In Section IV we discuss the DJAP algorithm in the context of this task, and in Section V we make some concluding remarks.

Nancy Fulda and Dan Ventura are with the Computer Science Department, Brigham Young University, Provo, UT 84602, USA (email: nancy@fulda.cc, ventura@cs.byu.edu).

II. Reinforcement learning and Q-Learning

Reinforcement learners attempt to learn the expected average reward (often called the *utility*) of each possible state-action pair based on a series of experimental interactions with the environment. Currently, the DJAP algorithm uses the Q-learning update equation [7] to estimate the utility $Q(s_t, a_t)$ of performing action a_t in state s_t:

$$\Delta Q(s_t, a_t) = \alpha_t(r(s_t, a_t) + \gamma \text{max}_a\{Q(s_{t+1}, a)\} - Q(s_t, a_t))$$

where $r(s_t, a_t)$ is the reward received for performing action a_t in state s_t, $0 < \alpha \leq 1$ is the learning rate and $0 \leq \gamma < 1$ is the discount factor. The learning rate may be decayed over time according to the equation

$$\alpha_t = \rho \alpha_{t-1}$$

where $0 < \rho \leq 1$. Note that the value of ρ can have a significant effect on the behavior of the algorithm, and we will have more to say about this later.

At each time step, a reinforcement learner may either exploit its knowledge of the environment by performing the action with the highest estimated utility or explore its environment by selecting some other action. For the experiments used in this paper, each agent exploits its environment with some probability p and selects a random action (which may or may not be optimal) with probability $1 - p$.

III. A multiagent rendezvous

The learning task studied in this paper is defined as a 4-tuple, $T = \{n, m, s, V\}$, where n defines the size of a square grid, m is the number of agent groups, s is the size of each group, and V is a set of possible rendezvous points. The agents are arbitrarily divided into m groups G_i of s agents each, so that

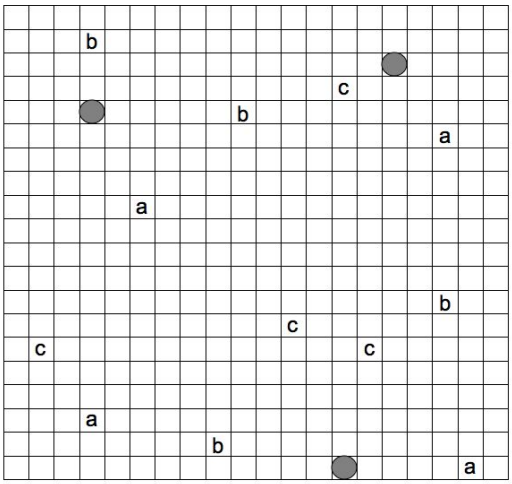

Fig. 1. An example starting configuration for the multiagent rendezvous task with $n = 20$, $m = 3$, $s = 4$ and $|V| = 3$. Rendezvous points are represented by filled circles and agents by alphabetic characters. Different alphabetic characters represent different agent groups that must learn to coordinate which rendezvous point they choose.

$$|G_i| = s, 1 \leq i \leq m$$
$$G_i \cap G_j = \emptyset, i \neq j$$
$$A = \bigcup_i G_i$$
$$|A| = ms$$

The locations of V are randomly generated and fixed for a particular instantiation of the task. We measure a set A of agents' ability to learn a particular instantiation of the task by allowing 5000 iterations of agent learning and then averaging the agent rewards received when all agents exploit with probability $p = 1$. Results reported here are the average of 30 such experiments with $n = 30$, $m = 6$, $s = 5$ and $|V| = 3$ fixed and with the locations of $v \in V$ randomized at the beginning of each new experiment. Note that even with these modest values, the size of the joint action space is 2^{31}, rendering joint action learners such as those discussed in Section I computationally infeasible.

At the beginning of each iteration, each agent is randomly assigned a location on the grid, and multiple agents may share the same coordinates. During each iteration, each agent may execute one of four possible actions: travel from its starting coordinates to one of the rendezvous points, or stay put. Each agent that chooses to move to a rendezvous point does so in a single step, agent rewards are calculated, the agents perform any learning, and the iteration is complete.

If all members of a group select the same rendezvous point, then each agent in the group receives a reward of 10, otherwise they receive a reward of 0. The agents are given no *a priori* information about the size of the groups or which group they are in. This models real-world tasks in which some agents are more tightly coupled than others, but for which the couplings may not be determinable in advance (see Figure 1 for an example initial configuration).

In addition to the potential reward for successful coordination, each agent a also incurs a cost for the distance it must travel to reach its chosen rendezvous point $v \in V$. This cost is the ratio of the Manhattan distance the agent travels to the maximum possible travel distance. This cost is distributed across the entire group, so that for each group G of agents the group penalty p_G that each agent receives is given by

$$p_G = \sum_{a \in G} \frac{|a_x - v_x| + |a_y - v_Y|}{(30 + 30)}$$

If we define the predicate $ren(a, v)$ to be true if agent a chooses to rendezvous at point v, then the group reward r_G can be expressed as

$$r_G = \begin{cases} 10 - p_G & \text{if } \forall a \in G, ren(a, v) \\ 0 - p_G & \text{otherwise} \end{cases}$$

Since the penalty is received regardless of whether the agents successfully coordinate their actions or not, the agents may be tempted not to rendezvous and to simply remain where they are (which incurs no cost). However, if the agents learn to cooperate, they can (on average) obtain a reward of about 7.5. (10 for rendezvousing, -0.5 for each agent to get there, $10 - 2.5 = 7.5$). In fact, this estimate is somewhat pessimistic, because the agents have a choice of 3 rendezvous points and can choose the one that minimizes total travel cost for all agents.

Figure 2 shows the performance of standard Q-learning agents on this task for several possible values of ρ. The agents have clearly not learned to coordinate their actions, since even assuming the maximum possible costs for traveling to the rendezvous point, the agents should receive a total reward of at least 5 for successful coordination. Varying the amount of exploitation does not significantly improve the system's performance, nor does increasing the number of training iterations. Note that a more classical treatment of the

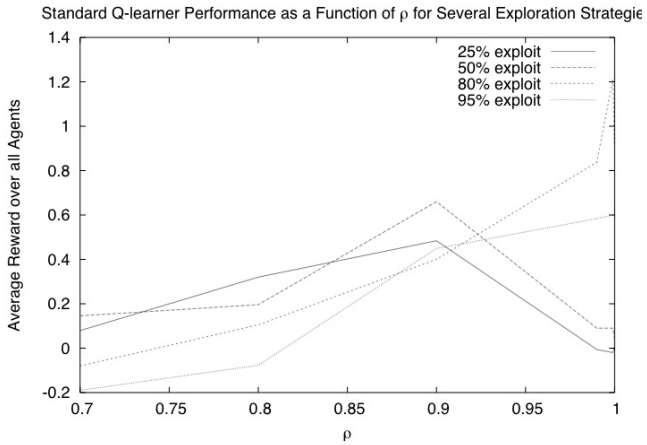

Fig. 2. Performance of standard reinforcement learners on the rendezvous task for varying values of ρ and p. The agents were trained for 5,000 time steps and the results of 30 experimental runs were averaged. The standard deviation of each datapoint was less than 1.1.

learning rate decay (e.g., $\alpha = 1/(1 + visits(s, a))$ produces similar results (at a much slower rate) because the standard Q-learners will never learn to consistently cooperate.

IV. The Dynamic Joint Action Perception algorithm

The Dynamic Joint Action Perception algorithm was originally introduced in [5]. The reader is referred to that paper for a more extensive description.

The DJAP algorithm uses a decision tree to create a variable resolution partitioning of the joint action space. The algorithm begins execution with a tree consisting of a single leaf node containing estimated utilities for performing each possible action given the current state. The leaf node contains a set of child fringe nodes indexed by the other agents in the system. Each fringe node contains a set of joint utilities which represent the expected reward for performing each action given the current state and given the observed action selection of the agent to which the fringe node corresponds. An example of this structure is shown in Figure 3. Notice that agent a cannot discriminate its actions without considering the actions of other agents (the estimated utilities are 1 for all three actions in the leaf node).

The agent is allowed to explore the environment (updating the Q-values in the leaf and fringe nodes) until the leaf node has been visited qk times, where q is the average number of Q-values per associated fringe node and k is a user-defined parameter. At that point, the leaf node is expanded along the fringe node which maximizes the agent's ability to obtain reward. In the example in Figure 3, considering the joint action space with agent c or agent d does not help a discriminate the effect of its actions (see the fringe nodes for c and d). However, the joint action space that includes agent b's actions does provide useful information (see the fringe node for b in the figure). To take into account this new information, the leaf node is replaced by a branch node

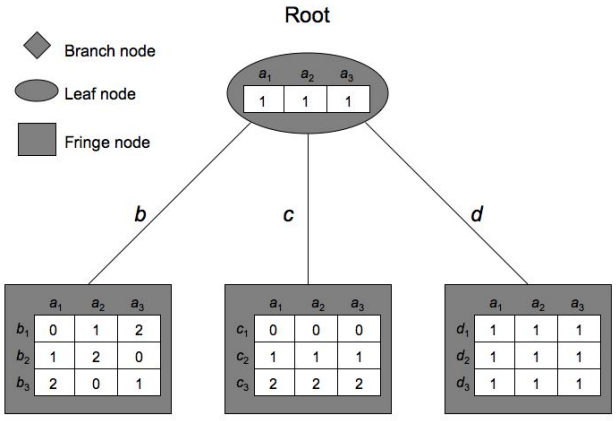

Fig. 3. An example root leaf node and associated fringe nodes for agent a in a system of four agents, denoted as a, b, c, and d, each of which has three possible action selections.

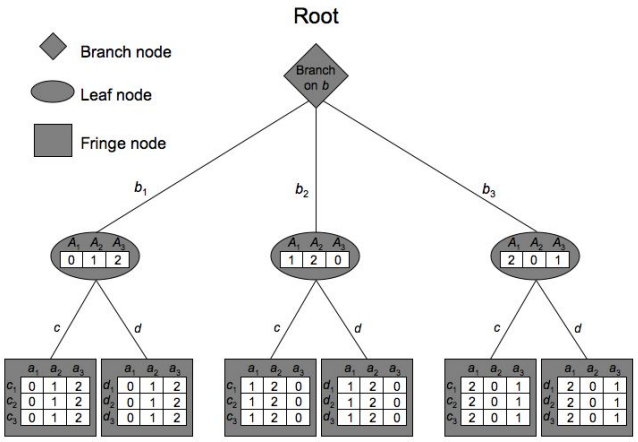

Fig. 4. The expansion of the tree in Figure 3 along the fringe node associated with agent b's action selection.

and a new set of leaf nodes is created, one for each possible action selection of the agent represented by the fringe node along which the leaf was expanded (in this case b). This leaf node expansion is shown in Figure 4. Notice how each newly created leaf node corresponds to a row in the joint action Q-value table of fringe node b in Figure 3 and that each new fringe node is initialized with these same Q-values. The process of qk visitations followed by expansion is then continued recursively for each of the newly created leaf nodes.

When selecting actions for execution once the learning phase is complete, the agent simply assumes that all other agents will act to maximize its reward. It therefore selects the individual action which will permit the agent's most-preferred joint action (based on the subset of the joint action space represented by the DJAP tree structure) to be executed.

A. Representational and learning ability

The DJAP tree is capable of representing arbitrary nth-order correlations between agents. But just because these capabilities can be represented does not necessarily mean that they are easy to learn. When the tree is being constructed, the algorithm searches only for first-order correlations between the current tree structure and the agents who have not yet been incorporated into that section of the tree. This means that higher-order correlations can be learned, but only if a first-order correlation chain connects them.

For example, let A be a set of agents and $C_n(a_i, G)$ represent an nth-order correlation between a_i and G, where $a_i \in A$, $G \subset A$, and $|G| = n$. Given an arbitrary ordering on the elements of A, $\{a_i, ..., a_n\}$, the DJAP algorithm can learn correlations which satisfy the condition $C_{k-1}(a_k, \{a_1, ..., a_{k-1}\})$ for all values of k, where $1 < k \leq n$.

Because the DJAP algorithm uses a policy-based test rather than a statistically-based test to determine the best fringe node to use for tree expansion, it will not necessarily find

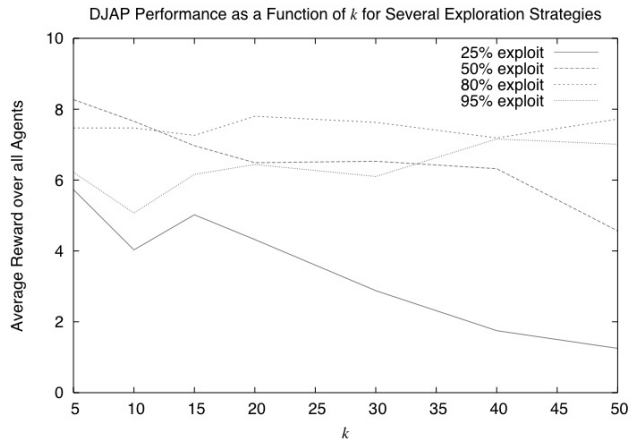

Fig. 5. Performance of the Dynamic Joint Action Perception algorithm for varying values of k and p. The agents were trained for 5,000 time steps and the results of 10 experimental runs were averaged. The standard deviation of each datapoint was less than 1.52.

every statistical correlation between other agents' actions and current rewards – it will expand first along fringe nodes which provide the opportunity to increase rewards (for example, fringe b in Figure 3) rather than those where a statistical correlation exists but cannot be capitalized upon (as with fringe c in Figure 3).

In the limit as expansion continues the tree will eventually represent the entire joint action space. In practical terms, however, this point is reached only in very small systems.

B. Learning rate and the parameter k

In the DJAP algorithm, the rate ρ at which α is decayed is determined as a function of the user-defined parameter k. For fringe nodes, the value of ρ is determined by the equation

$$\rho = e^{\frac{\ln\left(\frac{0.01}{0.1}\right)}{k}}$$

For leaf nodes, the value of ρ is determined by

$$\rho = e^{\frac{\ln\left(\frac{0.001}{\alpha_\mu}\right)}{ck}}$$

where c is the average number of possible percept values per fringe node and α_μ is the average learning rate of the leaf node's initial Q-values (usually about 0.1).

If the value of the parameter k is sufficiently large to allow optimal convergence of the leaf and fringe node Q-values then the DJAP algorithm will always branch along the fringe node that allows maximum increase in expected reward. However, large values of k increase the learning time required by the algorithm.

Figure 5 shows DJAP performance on the rendezvous task for several values of k under varying exploration strategies. The agents were trained for 5,000 time steps in each experimental run. Overall, algorithm performance tends to be superior with smaller values of k, even though there is no guarantee that the tree will split optimally for these values. The likely reason for this is that it is unnecessary to allow the fringe node Q-values to fully converge. They need only to begin converging towards their optimal values in order for the algorithm to differentiate between fringe nodes that do and don't provide a potential increase in expected reward.

C. Exploration and training time

Figure 6 shows the response of both DJAP agents and standard Q-learners to varying values of the exploitation parameter p. Because standard Q-learners never learn an acceptable group behavior for the rendezvous task, their performance is not significantly affected by the value of p. The DJAP algorithm, however, shows markedly improved performance when $p > 30$. The reason is that the increased exploitation on the part of each agent helps to concentrate leaf node visitations in useful areas of the joint action space. This causes those areas of the tree to branch earlier and helps the agents to identify more members of their group before training is done.

Figure 7 shows system performance as a function of training time for both DJAP and standard Q-learning agents. Again, the standard Q-learners are not significantly affected by the amount of training because they never learn an acceptable policy. DJAP agents using $k = 5$ and $p = 50$ learn a reasonable policy relatively quickly, within about 2,000 time steps, and approach an optimal policy by 5,000 time steps.

D. Applicability

Like most algorithms, the Dynamic Joint Action Perception algorithm is better-suited to some tasks than to others. Tasks which are well-suited for the DJAP algorithm have the following characteristics:

- **Agents share common goals.** The DJAP algorithm relies on an optimistic assumption for action selection. Each agent assumes that the other agents will act to maximize its reward.

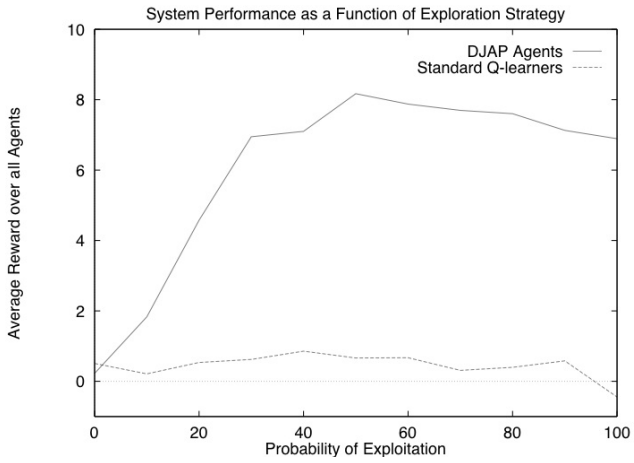

Fig. 6. System performance as a function of the exploitation parameter p. For DJAP agents, $k = 5$. For standard Q-learners, $\rho = 0.9$. Agents were trained for 5,000 time steps and the results of 30 experimental runs were averaged. The standard deviation of each datapoint was less than 2.02 for DJAP agents and less than 1.1 for standard Q-learners.

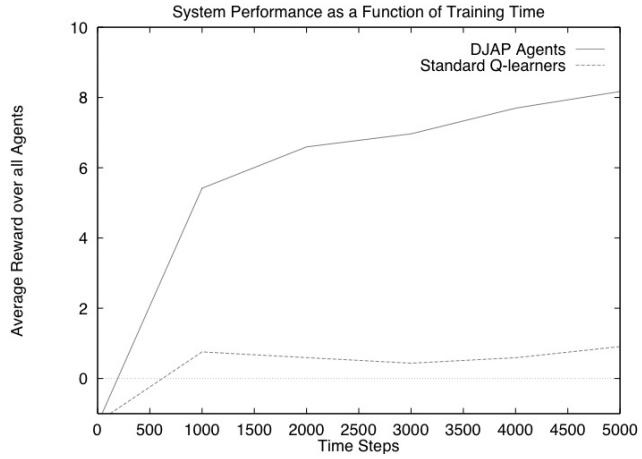

Fig. 7. System performance as a function of training time. Each agent used an exploitation parameter of $p = 50$. For DJAP agents, $k = 5$. For standard Q-learners, $\rho = 0.9$. The results of 30 experimental runs were averaged. The standard deviation for each datapoint was less than 1.08 for DJAP agents and less than 0.96 for standard Q-learners.

- **Not all agents affect each other equally.** The DJAP algorithm is able to avoid the system overhead incurred by complete joint action learning because it represents only a small subset of the joint action space. If the entire joint action space must be represented to achieve optimal performance, then the DJAP algorithm has no benefits over complete joint action learning.
- **Failed coordination attempts are penalized.** When there is no penalty for failed coordination, standard reinforcement learners are often as effective as agents that perceive the joint action space. In this case, the extra complexity of the DJAP algorithm is unnecessary.
- **Correlations are first-order or linked by a first-order correlation chain.** If the DJAP algorithm cannot find any first-order correlations in the available fringe nodes, it will randomly select a fringe node for expansion. This might lead to the accidental discovery of higher-order correlations that do not fit the constraints described in section IV-A, but there is no guarantee that this will happen.

V. Conclusion

The Dynamic Joint Action Perception algorithm is able to learn effective policies for coordination tasks by allowing each agent to dynamically observe a subset of the joint action space. This prevents the high overhead associated with algorithms that learn the complete joint action space while still producing effective performance on appropriate tasks. This paper has used a 30 agent rendezvous task as the basis for studying the performance of the DJAP algorithm under different values of user-defined parameters. The algorithm performs best for small values of k, values of $p > 30$, and at least 2,000 time steps of training. The algorithm is capable of learning higher-order correlations as long as they are connected by a first-order correlation chain.

Future work in this area should concentrate on expanding the range of learning situations to which the DJAP algorithm is applicable. In situations where the optimistic assumption is violated, the use of a minimax assumption for zero-sum games or a fictitious play implementation for general-sum games would be desirable. Alternative methods for determining which node to split on should be investigated. For example, a statistical test for significance of a particular split might provide a more principled method for splitting nodes and may also result in a natural stopping criterion (i.e. when no splits are likely to produce statistically significant improvement in the learned policy). A means of dynamically determining an appropriate value for k should also be developed.

As defined, the task is stateless and episodic. However, state can easily be introduced as the locations of $v \in V$, and the task could be further complicated by using agent locations to augment this state. Also, agents could be given a simpler action set (up, down, right, left) and forced to re-evaluate their decisions every step on the way to the rendezvous point. Competition could be introduced amongst agent groups, the task could be made recurring, etc. This would allow for a richer set of agent interactions, and would create an environment for extending the DJAP algorithm to allow for the possibility of agents employing signaling mechanisms, threats, reputation, etc.

One of the difficulties of learning in multi-agent settings is the non-stationarity of the environment (due to the fact that the other agents are changing their behaviors as they learn). This is problem can be ameliorated, to a degree, by allowing agents to learn at different rates – the environment for the fast learning agent appears relatively stationary (c.f. the WOLF family of algorithms [8], [9]). It may be that a similar approach will further improve DJAP learning performance as well.

DJAP style learning produces a graphical representation that in essence factorizes the joint action space, allowing learning in situations where the full space is too large to treat explicitly. Recent work on coordination graphs [10], [11] presents efficient algorithms for agent coordination given a graphical factorization of the joint action space. Combining these two approaches may lead to an elegant approach to the general multi-agent cooperation/coordination problem.

References

[1] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *AAAI/IAAI*, 1998, pp. 746–752.

[2] S. Kapetanakis and D. Kudenko, "Improving on the reinforcement learning of coordination in cooperative multi-agent systems," in *Second AISB Symposium on Adaptive Agents and Multi-Agent Systems*, 2002.

[3] J. Hu and M. Wellman, "Nash q-learning for general-sum stochastic games," *Journal of Machine Learning Research*, to appear, 2003.

[4] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative learning," in *Readings in Agents*. San Francisco: Morgan Kaufmann, 1997, pp. 487–494.

[5] N. Fulda and D. Ventura, "Dynamic joint action perception for q-learning agents," in *Proceedings of the International Conference on Machine Learning and Applications*, Los Angeles, Ca, 2003, pp. 73–78.

[6] ——, "Predicting and preventing coordination problems in cooperative q-learning systems," in *AAAI*, in submission, 2006.
[7] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.
[8] M. H. Bowling and M. M. Veloso, "Multiagent learning using a variable learning rate," *Artificial Intelligence*, vol. 136, no. 2, pp. 215–250, 2002.
[9] M. Bowling, "Convergence and no-regret in multiagent learning," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. Cambridge, MA: MIT Press, 2005, pp. 209–216.
[10] C. Guestrin, D. Koller, and R. Parr, "Multiagent planning with factored mdps," in *14th Neural Information Processing Systems (NIPS-14)*, 2001, pp. 1523–1530.
[11] C. Guestrin, S. Venkataraman, and D. Koller, "Context specific multiagent coordination and planning with factored mdps," in *AAAI*, 2002, pp. 253–259.