Dynamic Joint Action Perception for Q-Learning Agents

Nancy Fulda and Dan Ventura
owens@cs.byu.edu, ventura@cs.byu.edu
Department of Computer Science
Brigham Young University

April 18, 2003

Abstract

Q-learning is a reinforcement learning algorithm that learns expected utilities for state-action transitions through successive interactions with the environment. The algorithm's simplicity as well as its convergence properties have made it a popular algorithm for study. However, its non-parametric representation of utilities limits its effectiveness in environments with large amounts of perceptual input. For example, in multiagent systems, each agent may need to consider the action selections of its counterparts in order to learn effective behaviors. This creates a joint action space which grows exponentially with the number of agents in the system. In such situations, the Q-learning algorithm quickly becomes intractable. This paper presents a new algorithm, Dynamic Joint Action Perception, which addresses this problem by allowing each agent to dynamically perceive only those joint action distinctions which are relevant to its own payoffs. The result is a smaller joint action space and improved scalability of Q-learning to systems with many agents.

Keywords: Q-learning, Reinforcement Learning, Multiagent Systems

1. Introduction

Q-learning is a temporal differencing algorithm in which the agent learns expected time-discounted rewards $Q(s, a)$ for each state-action pair [16]. The basic Q-learning update algorithm is

$$\Delta Q = \alpha(r(s_t, a_t) + \gamma argmax_a\{Q(s_{t+1}, a)\} - Q(s_t, a_t))$$

where $r(s_t, a_t)$ is a numerical reward (also called a payoff) received for performing action a in state s at time t, $0 \leq \alpha \leq 1$ is the learning rate and $0 \leq \gamma \leq 1$ is the discount factor. Q-learning is guaranteed to converge to the theoretically optimal Q-values with respect to the discount factor under specified conditions [15].

This algorithm requires that an agent maintain $|S| * |A|$ distinct Q-values, where $|S|$ is the size of the state space and $|A|$ is the size of the action space. This representation both slows the learning speed of Q-learning systems with large state or action spaces and limits the tractability of Q-learning as state- or action-space size increases. One area where this problem arises is in the realm of distributed problem solving and multiagent systems where many agents work together to accomplish a common goal. Such applications generally require a strong coupling between specific agents: the actions of one agent affect the payoffs received by one or more of its counterparts. Hence, each agent must take the behavior of its companions into account when estimating Q-values. Otherwise, effective system convergence cannot be achieved.

A common approach for applying Q-learning to multiagent systems is to allow each agent in the system to perceive the action selections of its counterparts [2, 4, 6]. This has proven quite effective for systems with only a few agents. However, the size of the joint action space to be represented grows exponentially with the number of agents in the system. Each agent in a system of n agents with $|A|$ distinct actions and $|S|$ distinct states must store $|S| * |A|^n$ Q-values.

Some of this combinatorial explosion can be avoided through careful planning of agent couplings. As the number of agents in a system increases, the chance that all agents have an equally strong effect on each other decreases. System designers can capitalize on this tendency by allowing agents to perceive only the action selections of counterparts who significantly affect their payoffs. In a traffic light control system for city streets, for example, each agent might be allowed to perceive the color of lights at nearby intersections, but not the color of lights across town.

Such design strategies can decrease the size of each agent's perceived joint action space, but they are inapplicable in many situations for two reasons. First, the strength and structure of agent couplings may not be intuitively apparent at design time. Second, gradual change in a real-world environment may invalidate some agent couplings and generate new ones.

This paper presents Dynamic Joint Action Perception (DJAP), a Q-learning system which allows each agent to construct its own joint action space dynamically from a set of available agent actions, thus reducing the size of the joint action space without intervention from the system's designer. In the DJAP algorithm, action selections of other agents are modeled as part of an agent's individual state. A tree structure is then used to create a variable-resolution partitioning of this augmented state space. New state space distinctions are created as the agent locates percepts (actions of other agents) which have a significant effect on its payoffs.

2. Related Work

Several researchers have demonstrated the effectiveness of allowing agents in a multiagent system to perceive the action selections of their counterparts. This technique is frequently called *joint action learning*. Littman's Minimax-Q algorithm [6], Hu and Wellman's multiagent Q-learning algorithm [3], and Claus and Boutilier's joint action learners [2] are all examples of this technique applied to Q-learning systems. Other approaches for encouraging optimal behavior in multiagent reinforcement learning systems include policy search [13], optimistic updating techniques [5], agent modeling [14, 11], and the establishment of social conventions [12, 7].

Work on state space partitioning and variable-resolution state space representations includes McCallum's U-Tree algorithm [9] and Utile Suffix Memory algorithm [8], Munos and Moore's Parti-game algorithm [10], and Chapman's G-algorithm [1]. Each of these algorithm selectively distinguishes only those aspects of the state space which are useful in accomplishing the given task.

The research presented in this paper differs from previous research in dynamic state space partitioning because it applies the partitioning concepts to a new application: joint action learning in multiagent environments. The research extends previous research in joint action learning by addressing the issue of scalability.

3. Implementation: the Dynamic Joint Action Perception Algorithm

Dynamic Joint Action Perception (DJAP) is a new algorithm designed to improve the tractability of Q-learning in systems with large numbers of agents. The algorithm achieves this by making three fundemental assumptions: 1) it assumes that the agents all share a common goal, 2) it assumes that some agents have a greater effect on each others' payoffs than others, and 3) it assumes a first-order correlation between the behavior of other agents and the observed payoff distributions.

These assumptions are, admittedly, restrictive. However, within the bounds of these assumptions Dynamic Joint Action Perception is able to learn effective strategies in environments with many interacting agents. Improvements to the algorithm may enable a relaxation of these requirements.

The Dynamic Joint Action Perception algorithm uses a decision tree to create a variable resolution representation of the joint action space. This process is similar to that used by Andrew McCallum's U-Tree algorithm [9]. The primary distinction is that U-Tree uses a statistical test to determine which percepts are relevant, while DJAP uses expected average increase in payoff. This simplification in the DJAP algorithm makes it less resource intensive.

3.1. DJAP Tree Structure

In the DJAP algorithm, action selections of other agents are modeled as potential percepts which may be used when determining the agent's individual state. The DJAP algorithm begins execution with a tree consisting of a single leaf node. This leaf node represents a single state of the DJAP agent in which the actions of other agents are ignored. The leaf node contains a set of Q-values representing the expected utility of executing each possible action given the current state.

The leaf node contains a set of child fringe nodes indexed by the set of unused percepts (i.e. the action selections of other agents in the system). Each fringe node contains a set of joint Q-values which represent the expected utilities of each action selection given the current state (as indicated by the parent leaf node) and by the observed value of the unused percept to which the fringe node corresponds. An example of this structure for two unused percepts is shown in Figure 1.

The agent is allowed to interact with the environment until each fringe node Q-value has been updated approximately k times, where k is a user-defined param-

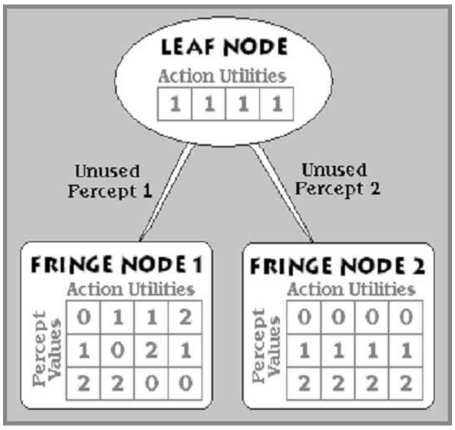

Figure 1. Structure of leaf and fringe nodes in the Dynamic Joint Action Perception Algorithm. Leaves expand along the unused percept which offers the greatest average increase in reward for the agent.

eter. (For the experiments documented in this paper, a value of 50 was used for k.) At that point, one of the unused percepts is selected as the basis for an expansion of the tree. The selection criterion is based on the increase in expected reward obtainable by the agent if the unused percept in question were incorporated as part of the agent's state.

For example, in Figure 1, the agent could increase its average expected reward from 1 to 2 if it were to incorporate unused percept 1 as part of its internal state, because for each possible percept value, there is an action option for the agent which provides a reward of 2. Unused percept 2, in contrast, does not allow an increase in expected reward. Even when the agent can perceive the values of unused percept 2, it can obtain a reward of 2 only approximately 1/3 of the time, regardless of its action selections. Thus, in this example, the leaf node would be expanded along unused percept 1.

When a leaf node is expanded, it is replaced by a branch node. Each branch node has one child for each possible value of the unused percept which was selected for expansion. Each newly created branch node contains a set of child leaf nodes. The Q-values of these leaf nodes are taken from the corresponding elements in the Q-value table of the fringe node for the percept along which the tree was expanded. Each newly-created leaf node generates a set of fringe nodes based on all of the remaining unused percepts. The initial Q-values for these fringe nodes are generalized from the leaf node Q-values: fringe node Q-values are initialized based on the Q-value for each action selection, regardless of the value of unused percept which the fringe node represents. Fringe node Q-value distinctions based on the percept values will be learned through further interactions with the environment.

Leaf node expansion continues until some user-defined stopping criterion is reached. Examples of potential stopping criteria include a minimum threshold on the increase in expected reward required to qualify a percept for expansion, an upper bound on the depth of the tree, or a limit on the number of nodes in the tree. The version of DJAP used for this paper implements no stopping criterion at all. The tree is continually expanded throughout the training period. Because previously learned Q-values are generalized to newly created fringe nodes, overexpansion is not deteremental to system performance in this case, although it does have a negative impact on resource usage and adaptability to subsequent changes in the environment.

In summary, each branch of the decision tree represents an available percept (i.e. the action selections of another agent). Each branch node has one child for each possible value of the percept in question. Each leaf node of the tree represents a state of the DJAP agent, with each state corresponding to a specific combination of actions of other agents. Each leaf node also maintains a set of fringe nodes, with one fringe node for every available percept which has not been used in that section of the tree. Leaf nodes are expanded along the unused percept which offers the greatest potential increase in average reward. Expansion of the tree continues until a user-defined stopping criterion is reached.

3.2. Learning Rate

A critical factor for any Q-learning algorithm is the learning rate used. In the DJAP algorithm, the objective is for fringe node Q-values to converge to nearly-optimal Q-values before expansion of the parent leaf node occurs. Learning rates are therefore dependent on the user-defined value k, the average number of updates received by each fringe node Q-value before expansion occurs.

In the current implementation of the DJAP algorithm, each leaf node and each fringe node maintains individual learning rates for each Q-value. These learning rates are intialized to 0.1 for fringe Q-values. Newly-created leaf nodes "inherit" the final learning rates of the fringe nodes from which they are created. Normally, the inherited learning rate is approximately 0.01 (The root leaf is an exception. It uses an initialization value of 0.1). The learning rate of each Q-value is decayed by a factor of β each time the Q-value is updated. The objective is to ensure that the learning rate has decreased to a target value of 0.01 for fringe nodes and 0.001 for leaf nodes by the time k updates per fringe node Q-value have occured.

For fringe nodes, the value of β is determined by the equation $\beta = 1/k * ln(0.01/0.1)$. For leaf nodes, the value of β is determined by $\beta = 1/kp * ln(0.001/a)$, where p is the average number of possible percept values per fringe node and a is the average learning rate of the leaf node's current Q-values. (Recall that when a new leaf node is created, it inherits the Q-values and current learning rates of the fringe node which was selected for expansion.) For the root leaf node, $a = 0.1$.

3.3. Determining the Optimal Policy

When selecting actions for execution once the learning phase is complete, the DJAP algorithm encounters a problem. The state space of the agent is partially defined in terms of the action selections of other agents. But these action selections cannot be known until *after* the agent has acted. How can the agent know which action to perform if it does not know what state it is in?

To address this problem, the algorithm uses an optimistic assumption [5]. The agent simply assumes that all other agents will act to maximize its reward. It therefore selects the action which will permit the agent's most-preferred joint action to be executed. Assuming that all other agents have learned to perceive the same preferred joint action, and that the optimistic assumption holds, the system will exhibit optimal behavior.

4. Test Problem: Multiagent Penny-Matching

The DJAP algorithm was tested on a task structure which is reminiscent of a classic multiagent coordination problem: the matching pennies game.

In the matching pennies game, two agents are asked to pick a side of a penny: heads or tails. If both agents choose the same side, then they receive a payoff of 1. If they choose different sides, they receive a payoff of -1. The objective is for the agents to learn to coordinate their actions to obtain optimal payoff.

The implemented version of the matching pennies game differs from the classic example in several ways.

Group Size: The game is played in groups of n agents. Each group of agents tries to coordinate the actions of all group members.

	A_h/B_h	A_h/B_t	A_t/B_h	A_t/B_t
C_h/D_h	(1, 1, 1, 1)	(-2, 0,-2,-2)	(0,-2,-2,-2)	(0, 0,-2,-2)
C_h/D_t	(-2,-2,-2, 0)	(-2, 0,-2, 0)	(0,-2,-2, 0)	(0, 0,-2, 0)
C_t/D_h	(-2,-2, 0,-2)	(-2, 0, 0,-2)	(0,-2, 0,-2)	(0, 0, 0,-2)
C_t/D_t	(-2,-2, 0, 0)	(-2, 0, 0, 0)	(0,-2, 0, 0)	(0, 0, 0, 0)

Figure 2. Payoff matrix for four agents playing a variant of the matching pennies game. Grid values represent the payoffs for agents A, B, C, and D, respectively.

Multiple Groups: The playing environment consists of m groups of n agents playing the matching pennies game simultaneously. Agents are given no information about the size or number of the playing groups, nor do they know which other agents are in their groups.

Nondeterminism: With 5% probability on every round, someone bumps the virtual playing table and all pennies are flipped to random sides. Thus the agents' rewards are not always correlated with their action selections.

Reward Structure: The agents are not only required to coordinate their actions by selecting a specific penny side, but they must do so in the face of a temptation to "defect". If all agents pick heads, then all agents receive a reward of 1. However, if one or more agents choose tails, then each agent that selected heads receives a reward of -2 and each agent that selected tails receives a reward of 0.

An example payoff matrix of this reward structure for a group size of four is shown in Figure 2. In general, this reward structure is not learnable by reinforcement learning agents unless they are able to see the action selections of their counterparts.

The penny-matching environment shares many characteristics with more situated problems such as robot soccer, formation flying, and rendezvous tasks. Each agent is a member of a much larger global system and must learn to coordinate its actions with the actions of some – but not all – of the other agents in the system in order to achieve desirable results. The agent does not know in advance which members of the system will have a significant effect on its rewards. This subset must be learned.

5. Results

The Dynamic Joint Action Perception algorithm was tested in an environment consisting of 32 agents with 4 agents per group. This creates a system joint action space of 2^{32} distinct action combinations.

Three types of Q-learning agents were compared in this environment: DJAP agents, more traditional joint ac-

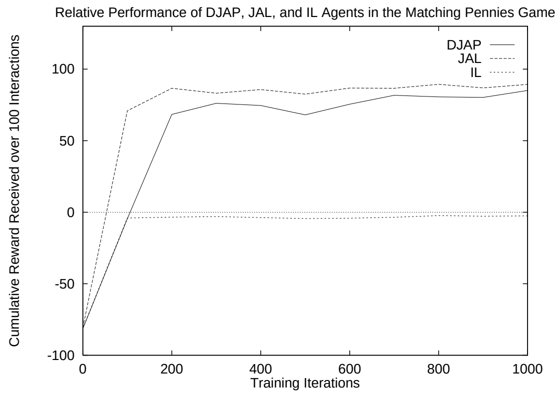

Figure 3. Performance of DJAP, JAL, and IL agents on the matching pennies game. Average of 10 trials.

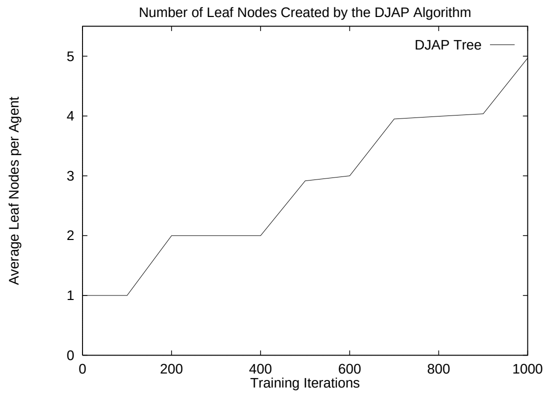

Figure 4. Number of leaf nodes created per agent by the DJAP algorithm when learning the matching pennies game. Average of 10 trials.

tion learning (JAL) agents, and independently learning (IL) agents. The DJAP agents use the Dynamic Joint Action Perception algorithm described in this paper to learn a variable-resolution joint action space. The JAL agents use a hand-designed joint action space: each agent was allowed to see the action selections of the three other agents in its playing group, creating a total of 2^3 perceived joint actions and 2^4 Q-values. The independently learning agents execute a normal Q-learning algorithm, without regard for the behavior (or even the existence) of the other agents in the system.

During the learning period, each agent executed the action with the highest Q-value (or the action which enabled the maximal joint action, in the case of DJAP and JAL agents) with 80% probability. A random action selection was executed with 20% probability.

Figure 3 shows the results for each of these algorithms in the matching pennies environment. As one might expect, the JAL agents learn the task most quickly, as they do not have to spend any time learning which percepts are relevant to their rewards. The DJAP agents also learn surprisingly quickly. Their overall performance is slightly impaired, however, because the algorithm is not guaranteed to split along the correct percepts. Thus, although most agents learn optimal policies, some playing groups do not learn to obtain optimal rewards. Additional training time can help to improve performance, but because each split in the DJAP tree increases the total number of states, an incorrect split early in training may take a very long time to overcome, even if the correct split is taken later on.

Figure 4 shows the number of leaf nodes created by the DJAP agents as a function of the number of training iterations. Analysis of this graph presents a surprising result. At 200 interactions, the point at which DJAP cumulative reward jumps to 68 in Figure 3, the DJAP agents have only two leaf nodes each: they are only perceiving the actions of one other agent. This is less information than one would expect the agents to be able to learn an efficient policy with. However, the probabilistic exploration algorithm used by the agents allows them to learn the task with less information than they would require if completely random exploration were used. This result is significant because it demonstrates that even in simple tasks, the DJAP algorithm can achieve close to the same performance as a hand-designed algorithm, but with fewer state distinctions.

One difficulty that arose with the DJAP algorithm during testing is its sensitivity to the exploration strategy used by the agents. In many cases, the 80%-20% exploration strategy used to generate Figure 3 was effective. In other cases, however, particularly when the number of agents interacting in the environment was small, this exploration strategy failed to produce a desirable policy. A completely random exploration strategy was similarly irregular in effectiveness. This sensitivity to the exploration pattern used represents a significant area for future research.

6. Conclusion

The Dynamic Joint Action Perception (DJAP) algorithm allows Q-learning agents to dynamically create joint action spaces in environments with large numbers of interacting agents. This is of value because hand-coding agent couplings for joint action learning systems is often impractical. The empirical results presented in this paper indicate that, at least for some problems, DJAP agents can learn successful policies with a relatively small joint action space. In some cases, DJAP can achieve reasonable performance with a smaller joint action space than that used by a hand-coded set of joint action learners.

The DJAP algorithm offers several potential avenues for future research. The sensitivity of the DJAP algorithm to the exploration strategies used by the agents has already been mentioned. A better understanding of this sensitivity and the means by which it may be predicted or avoided would be desirable. Another avenue for future research involves the stopping criterion for leaf node expansion. A comparison of various stopping criteria and their relative advantages and disadvantages would be of significant value. The possibility of pruning to eliminate unnecessary state space distinctions and reduce tree size should be investigated, as should methods of seeking higher-order correlations between unused percepts and observed rewards. Finally, the DJAP algorithm should be applied to a complex, real-world problem to determine whether the DJAP advantages observed in the matching pennies game extend to less controlled environments.

References

[1] David Chapman. Penguins can make cake. *AI Magazine*, 10(4):45–50, 1989.

[2] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, pages 746–752, 1998.

[3] J. Hu and M. Wellman. Multiagent refinforcement learning: theoretical framework and an algorithm. In *Proceedings of the 15th International Conference on Machine Learning*, pages 242–250, San Francisco, 1998. Morgan Kaufman.

[4] J. Hu and M. Wellman. Experimental results on q-learning for general-sum stochastic games. 2000.

[5] Martin Lauer and Martin Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *Proceedings of the 17th International Conference on Machine Learning*, pages 535–542, San Francisco, CA, 2000. Morgan Kaufman.

[6] Michael Littman. Markov games as a framework for multiagent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, 1994.

[7] M. J. Mataric. Learning social behavior. *Robotics and Autonomous Systems*, 20:191–204, 1997.

[8] Andrew McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 387–395, 1995.

[9] Andrew McCallum. Learning to use selective attention and short-term memory in sequential tasks. In *From Animals to Animats, Fourth International Conference on Simulation of Adaptive Behavior*, Cape Cod, Massachusetts, 1996.

[10] A.W. Moore and C. G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21(4):199–234, December 1995.

[11] M. Mundhe and S. Sen. Evaluating concurrent reinforcement learners. In *Learning about, from and with other agents workshop, ijcai99*, 1999.

[12] Ann Nowe, Katja Verbeeck, and Tom Lenaerts. Learning agents in a homo egualis society. In *Proceedings of Learning Agents Workshop, Agents 2001 Conference*, Montreal, Canada, 2001.

[13] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie P. Kaelbling. Learning to cooperate via policy search. In *Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 307–314, San Francisco, CA, 2000. Morgan Kaufmann.

[14] R. Sun and D. Qi. Rationality assumptions and optimality of co-learning. In *Design and Applications of Intelligent Agents, Lecure Notes in Artificial Intelligence*, volume 1881, pages 61–75, 2000.

[15] C. Watkins and P. Dayan. Technical note: Q-learning. *Machine Learning*, 8:279–292, 1992.

[16] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.