QUICR-learning: A coordinated multiagent learning algorithm

Paper number 435

November 9, 2009

Abstract

Multiagent learning in Markov Decisions Processes is challenging because of the presence of two credit assignment problems. First, credit must be assigned for an action taken at time step t that results in a reward at time step $t' > t$. Second, credit must be assigned for the contribution of agent i to the overall system performance. The first credit assignment problem is typically addressed with temporal difference methods such as Q-learning or TD(λ). The second credit assignment problem is typically addressed by either decomposing the system or creating custom reward functions. In domains where the coupling among the agent is critical but not overly tight, both credit assignment problems can be addressed simultaneously. "Q Updates with Immediate Counterfactual Rewards-learning" (QUICR-learning) is designed to specifically address such problems and improve both the convergence properties and performance of Q-learning in large multiagent problems. QUICR-learning uses counterfactual actions to eliminate the component of an agent's reward that is independent of an agent's action, while allowing this reward to operate without explicit knowledge of the structure of the agents' coupling. Results on a multiagent grid-world problem shows that QUICR-learning provides significantly faster and better results than conventional and local Q-learning. Additional results on a traffic congestion problem show that the improvements due to QUICR-learning are not domain specific and can provide up to a ten fold increase in performance over existing methods.

1 Introduction

Coordinating a set of interacting agents that take sequences of actions to maximize a system level performance criteria is a difficult problem. Addressing this problem with a large single agent reinforcement learning algorithm is ineffective in general because the state-space becomes prohibitively large. A more promising approach is to give each agent in the multiagent system its own reinforcement learner. This approach, however, introduces a new problem: how to assign credit for the contribution of an agent to the system performance, which in general is a function of all agents. Allowing each agent to try to maximize the system level global reward is problematic in all but the smallest problems as an agent's reward is masked by the actions of all the other agents in the system. In Markov Decisions Processes (MDPs) presented in this paper, the global

reward may be influenced by as many as 3200 actions (actions of 100 agents over 32 time steps). Purely local rewards allow us to overcome this "signal-to-noise" problem. On the other hand, local rewards are problematic, since there are no guarantees that policies formed by agents that maximize their local reward will also maximize the global reward.

In this paper, we describe "Q Updates with Immediate Counterfactual Rewards learning" (QUICR-learning) which uses agent-specific rewards that ensure fast convergence in multiagent coordination domains [1]. Rewards in QUICR-learning are both heavily *agent-sensitive*, making the learning task easier, and *aligned* with the system level goal, ensuring that agents receiving high rewards are helping the system as a whole. QUICR-learning uses standard temporal difference methods but because of its unique reward structure, provides significantly faster convergence than standard Q-learning in large multiagent systems. The key contribution of this paper is to leverage existing multiagent rewards and ensure that they remain agent-sensitive and aligned in a multi-time step problem.

In the next Section, we discuss the Markov Decision Process, and variants needed to simplify large multiagent problems. We then discuss the temporal and structural credit assignment problems in multiagent systems, and describe the QUICR-learning algorithm. The following two sections present results on two different problems that require coordination, showing that QUICR-learning performs up to ten times better than standard Q-learning in multiagent coordination problems. Finally we present a brief summary of the related research, discuss the implications and limitations of QUICR-learning, then highlight future research directions.

2 Markov Decision Processes

The multiagent credit assignment problem consists of determining how to assign rewards (e.g., credit) for a sequence of actions taken by a set of agents. In this paper, we analyze this credit problem, using a Markov Decision Process (MDP). We can define a standard MDP as a tuple $\langle S, A, \Theta, \mathcal{R}, \Pi \rangle$, where S is a finite set of states, A is a finite set of actions, Θ is the transition function mapping a state and an action to the next state, $\mathcal{R}$ is the reward function mapping an action and state to a real valued reward, and Π is the policy mapping states to actions. The goal of the system is to find a policy $\pi \in \Pi$ that leads to high values of future sum of rewards. In complex, multiagent problems, finding an optimal policy π is difficult and requires centralization. In this paper, instead we focus on finding solutions to simplified versions of the Markov Decision Process that can be efficiently solved by a multiagent system. We summarize the centralized problem as well as three decentralized versions of decreasing difficulty as follows:

- Centralized $\langle S, A, \Theta, \mathcal{R}, \Pi \rangle$: Find single policy π mapping system state $s \in S$ to system action $a \in A$ that maximizes future system rewards $r \in \mathcal{R}$.
- Full State Multiagent $\langle S, A_i, \Theta, \mathcal{R}, \Pi_i \rangle$: Agent i finds policy $\pi_i \in \Pi_i$ mapping system state s to agent's action $a_i \in A_i$ that maximizes future system rewards r.

[1] QUICR-learning was originally presented in [2]

- Local State Multiagent $\langle S_i, A_i, \Theta, \mathcal{R}, \Pi_i \rangle$: Agent i finds policy π_i mapping agent's state $s_i \in S_i$ to agent's action a_i that maximizes future system rewards r.

- Independent Agents $\langle S_i, A_i, \Theta, \mathcal{R}_i, \Pi_i \rangle$: Agent i finds policy π_i mapping agent's state s_i to agent 's action a_i that maximizes future agent's local rewards $r_i \in R_i$.

In this work, we focus on problems where the system state $s \in S$ can be broken down into agent states, and denote the state of agent i by $s_i \in S_i$. In addition, the joint action $a \in A$ can be broken down into agent actions $a_i \in A_i$. Note, this makes no assumptions about the interactions of the agents' states or actions (particularly about how joint states and actions lead to rewards) and whether those actions are independent.

2.1 Full State Multiagent Problem

In the centralized MDP we need to find a policy that will generate a system action $a \in A$, given the state of the system $s \in S$. As the number of agents increases, the size of the state and action spaces grow exponentially. In all but the most simple multiagent problems, finding a good policy π through direct methods becomes intractable. As an alternative to finding a global policy π for all agents, we can have each agent i find its own policy π_i mapping a state s to an agents individual action a_i. In this approach, each agent is separately trying to create a policy that looks at the state of the entire system and will try to take an action that will lead to high values of future global rewards. The advantage of this approach is that the action space for each agent's policy is relatively small, and agents can create policies in a distributed manner. However, again the state space is grows exponentially with the number of agents, and finding good policies are difficult in all but the most trivial systems.

2.2 Local State Multiagent Problem

In domains where the state transitions for an individual agent are critically dependent on the states of all the other agents, an agent may be forced to take into account the state s of the entire system into its policy. These problems are extremely complex, as an agent has to take into account the effect of the states of all the other agents on its own state transitions, as well the the effect of all the states on its reward. Luckily in many domains, the states of all the other agents are more important with respect to an agent's reward than with respect to its own state transitions. Such domains include satellite and UAV surveillance. In these domains agents rarely collide or need to avoid each other so detailed information about other agents' states are not important. In these domains an agent needs to know about it's own state, and perhaps the value of an "aggregate" state summarizing the states of all the other agents. Domains such as traffic congestion control are similar in that we do not care about the states of individual vehicles, but instead care about the aggregate number of vehicles in important locations.

In these types of problems, the state space for an agent can be significantly simpler than the full state of the entire system. In such domains the state space for an agent i can include its own state s_i, in addition to the aggregate state of all the other agents. In

problems where this aggregate state changes slowly, we can even eliminate the aggregate state, and the agent's state space will only consist of its own state s_i. The agents goal then is to create a policy that maps an agents current state to an action that will try to maximize the the system reward, which is still a function of all the states of the system. The goal the the QUICR-learning algorithm is to improve the performance of this process.

2.3 Independent Agent Problem

In some domains the states of the agents are completely independent: the action an agent should take to maximize the system reward is independent of the states of the other agents. In such domains the goal of an agent is to find a policy that maps its own state s_i to its action a_i that will maximize its own reward r_i. In these domains there is no true multiagent problem, since all the agents can act independently.

3 Solving MDPs and Credit Assignment

In solving an MDP we need to create a policy that maximizes the sum of future rewards. Starting from the current time step t, the undiscounted sum of rewards until a final time step T can be represented by:

$$R_t = \sum_{\tau=t}^{T} r_\tau \ . \tag{1}$$

To solve this problem we need to address the **temporal credit assignment problem** of how our current action a at time t affects the future rewards. Solving this problem can be complicated since an action taken now may increase our immediate reward, at the cost of reducing future rewards. In most multiagent problems, in addition to the temporal credit assignment problem, we also have the **structural credit assignment problem**: how an action a_i taken by an individual agent i affects the system reward R, which is a function of the actions of all of the agents. Solving both these credit assignment problems at once is difficult since each reward is a function of actions from different agents over different time steps. Every reward is a function of the states of all the agents, and every state is a function of all the actions that preceded it (even though it is Markovian, the previous states ultimately depend on previous actions). In a system with n agents, a reward received on the last time step can be affected by up to $n * T$ actions. Looking at rewards received at different time steps, on average $\frac{1}{2}n * T$ actions may affect a reward in tightly coupled systems. Agents need to use this reward to evaluate their single action; in the domains presented in the results sections with forty agents and twenty time steps there are up to 800 actions affecting the reward!

Note that both the full state and local state multiagent problems contain both credit assignment problems since they both maximize the full system reward, while the independent agent problem only contains the temporal credit assignment problem, since each agent is maximizing a reward that is only a function of its own state. In the following sections we describe how to address these credit assignment problems within the framework of reinforcement learning.

3.1 Standard Q-Learning

Reinforcement learners such as Q-learning address (though imperfectly) how to assign credit of future rewards to an agent's current action. The goal of Q-learning is to create a policy that maximizes the sum of future rewards, R_t, from the current state [23, 41, 49]. It does this by maintaining tables of Q-values, which estimate the expected sum of future rewards for a particular action in a particular state. In the TD(0) version of Q-learning, a Q-value, $Q(s_t, a_t)$, is updated with the following Q-learning rule [2]:

$$Q(s_t, a_t) = r_t + max_a Q(s_{t+1}, a) \ . \tag{2}$$

This update assumes that the action a_t is most responsible for the immediate reward r_t, and is less responsible for the sum of future rewards, $\sum_{\tau=t}^{T} r_\tau(s_\tau)$. This assumption is reasonable since rewards in the future are affected by uncertain future actions and noise in state transitions. Instead of using the sum of future rewards directly to update its table, Q-learning uses a Q-value from the next state entered as an estimate for those future rewards. Under certain assumptions, Q-values are shown to converge to the actual value of the future rewards [49].

Even though Q-learning addresses the temporal credit assignment problem (i.e., tries to apportion the effects of all actions taken at other time steps to the current reward), it does not address the structural credit assignment problem (i.e., how to apportion credit to the individual agents in the system). At every time step Q-learning uses the system reward r_t, which is a function of the actions of many agents. As a result when many agents need to coordinate their actions, standard Q-learning is generally slow since it needs all agents to tease out their time dependent contribution to the global system performance based on the system reward they receive. An additional issue with standard Q-learning is that in general an agent needs to fully observe the actions of other agents in order to compute its reward. This requirement is reduced in the "Local Q-Learning" and "QUICR-Learning" algorithms presented in this paper.

3.2 Local Q-Learning

One way to address the structural credit assignment problem and allow for fast learning is to assume that our multiagent problem is an independent agent problem. Without this assumption, the immediate reward function for a multiagent reward system may be a function of all the states:

$$r_t(s_{t,1}, s_{t,2}, \ldots, s_{t,n}) \ ,$$

where $s_{t,i}$ is the state for agent i and is a function of only agent i's previous actions. The number of states determining the reward grows linearly with the number of agents, while the number of actions that determine each state grows linearly with the number of time steps. To reduce the huge number of actions that affect this reward, often the

[2] To simplify notation, this paper uses Q-learning update notation for deterministic (where learning rate $\alpha = 1$ converges), undiscounted Q-learning. The extensions to non-deterministic and discounted cases through the addition of learning rate and discounting parameters are straight-forward.

reward is assumed to be linearly separable:

$$r_t(s_t) = \sum_i w_i r_{t,i}(s_{t,i}) \ .$$

Then each agent receives a reward $r_{t,i}$ which is only a function of its action. Q-learning is then used to resolve the remaining temporal credit assignment problem. If the agents are indeed independent and their pursuing their local rewards has no deleterious side effects on each other, this method leads to a significant speedup in learning rates as an agent receives direct credit for its actions. However, if the agents are coupled, then though local Q-learning will allow fast convergence, the agents will tend to converge to the wrong policies (i.e., policies that are not *globally* desirable). In the worst case of strong agent coupling, this can lead to worse than random performance [50]).

3.3 QUICR-Learning

In this section we present QUICR-learning, a learning algorithm for multiagent systems that does not assume that the system reward function is linearly separable. Instead it uses a mechanism for creating rewards that are a function of all of the agents, but still provide many of the benefits of hand-crafted rewards. In particular, QUICR-learning rewards have:

1. high "alignment" with the overall learning task.
2. high "sensitivity" to the actions of the agent.

The first property of alignment means that when an agent maximizes its own reward it tends to maximize the overall system reward. Without this property, a large multiagent system can lead to agents performing useless work, or worse, working at cross-purposes. Having aligned rewards is critical to multiagent coordination. Reward sensitivity means that an agent's reward is more sensitive to its own actions than to other agents' actions. This property is important for agents to learn quickly. Note that assigning the full system reward to all the agents (e.g., standard Q-learning) has low agent-sensitivity, since each agent's reward depends on the actions of all the other agents.

QUICR-learning is based on providing agents with rewards that are both aligned with the system goals and sensitive to the agent's states. It aims to provide the benefits of customizing rewards without requiring detailed domain knowledge. In a task where the reward can be expressed as in Equation 1, let us introduce the difference reward (adapted from [50]) given by:

$$d_t^i(s_t) \quad = \quad r_t(s_t) - r_t(s_t - s_{t,i} + c_{t,i}) \ ,$$

where $s_t - s_{t,i} + c_{t,i}$ denotes a counterfactual where the state of agent i, $s_{t,i}$, has been replaced by the counterfactual state $c_{t,i}$, which is independent of the state s_t [3]. We

[3]This notation uses zero padding and vector addition rather than concatenation to form full state vectors from partial state vectors. The vector "z_i" in our notation would be $z_i e_i$ in standard vector notation, where e_i is a vector with a value of 1 in the ith component and is zero everywhere else.

discuss the justification for this reward heuristic at the end of the section. Difference rewards of this form have been used successfully in a number of wide number of single-time-step domains, where rewards that were highly aligned the the overall task and highly sensitive to the agents actions were needed [4, 3, 46, 42, 45, 47, 50].

Given the definition of the difference reward, an MDP that is maximizing the sum of future difference rewards will be maximizing the following:

$$D_t^i(s_t) = \sum_{\tau=t}^{T} r_\tau(s_\tau) - r_\tau(s_\tau - s_{\tau,i} + c_{\tau,i}) \ . \tag{3}$$

From this equation, there are clear restrictions for the counterfactual $c_{\tau,i}$. First of all to be Markovian, given the state s_τ, $c_{\tau,i}$ should be independent of all the previous states. In addition our definition of difference reward requires $c_{\tau,i}$ to be independent of current state. These two restrictions force the sequence of $c_{\tau,i}$ to have completely independent dynamics to the rest of the problem. Since adding dynamics to $c_{\tau,i}$ simply introduces unneeded complexity, we simplify with a time-independent c_i and or an agent-independent c. The effect of these counterfactuals is to move the agent i to an *absorbing state*, where it never moves. Any attempt to create a sequence of counterfactuals that follows the actual dynamics will either break the Markov assumption of the definition of d. Given this simplification, the difference reward can be defined as:

$$d_t^i(s_t) = r_t(s_t) - r_t(s_t - s_{t,i} + c) \ , \tag{4}$$

where the expression $s_t - s_{t,i} + c$ denotes replacing agent i's state with state c.

Now the Q-learning rule can be applied to the difference reward, resulting in the QUICR-learning rule:

$$\begin{aligned} Q_{UICR}(s_t, a_t) &= r_t(s_t) - r_t(s_t - s_{t,i} + c) \\ &\quad + max_a Q(s_{t+1}, a) \\ &= d_t^i(s_t) + max_a Q(s_{t+1}, a) \ . \end{aligned} \tag{5}$$

Note that since this learning rule *is* Q-learning, albeit applied to a different reward structure, it shares all the convergence properties of Q-learning. In order to show that Equation 5 leads to good system level behavior, we need to show that agent i maximizing $d_t^i(s_t)$ (e.g., following Equation 5) will maximize the system reward r_t.

3.3.1 Alignment of Difference Reward

Note that by definition $(s_t - s_{t,i} + c)$ is independent of the actions of agent i, since it is formed by moving agent i to the absorbing state c from which it cannot emerge. This effectively means the partial differential of $d_t^i(s_t)$ with respect to agent i is[4]:

$$\frac{\partial}{\partial_{s_i}} d_t^i(s_t) = \frac{\partial}{\partial_{s_i}} \left(r_t(s_t) - r_t(s_t - s_{t,i} + c) \right)$$

[4]Though in this work we show this result for differentiable states, the principle applies to more general states, including discrete states.

$$
\begin{aligned}
&= \frac{\partial}{\partial_{s_i}} r_t(s_t) - \frac{\partial}{\partial_{s_i}} r_t(s_t - s_{t,i} + c) \\
&= \frac{\partial}{\partial_{s_i}} r_t(s_t) - 0 \\
&= \frac{\partial}{\partial_{s_i}} r_t(s_t). \qquad (6)
\end{aligned}
$$

Therefore any agent i using a learning algorithm to optimize $d_t^i(s_t)$ will tend to optimize $r_t(s_t)$.

3.3.2 Sensitivity of Difference Reward

In a wide variety of domains the difference reward has been empirically shown to be more sensitive to the agents' actions than the system reward [4, 3, 46, 42, 45, 47, 50]. Although this benefit is highly dependent on the structure of the particular domain. A more theoretical justification can be given in domains where an agent's impact on the system reward is coupled with a small number of other agents, but independent of many others. Let us assume that the reward in such a domain can be decomposed into an additive set of terms:

$$r(s) = r_i(s_i) + r_{-i}(s_{-i}) + r_k(s) \ , \qquad (7)$$

where $r_i(s_i)$ is the impact of agent i's action independent of the actions of other agents, $r_{-i}(s_{-i})$ is the impact of other agents' actions independent of agent i, and $r_k(s)$ is the impact of agent i's action coupled with the actions of other agents. If the system reward is in this form, the difference reward becomes:

$$
\begin{aligned}
d_t^i(s_t) &= r_t(s_t) - r_t(s_t - s_{t,i} + c) \\
&= r_i(s_{t,i}) + r_{-i}(s_{t,-i}) + r_k(s_t) - (r_i(c) + r_{-i}(s_{t,-i}) + r_k(c, x_{t,-i})) \\
&= r_i(s_{t,i}) + r_k(s_t) - r_k(c, x_{t,-i}) \ .
\end{aligned}
$$

Here the terms independent of agent i's action cancel out and all that we are left with is the term solely dependent on agent i's action and the coupled terms. If the magnitude of these coupled terms are relatively small (agents are not tightly coupled together), then we would expect the difference reward to be more sensitive to an agent's actions.

3.4 QUICR-Learning and Difference Rewards

The difference reward in QUICR-Learning, $d_t^i(s_t) = r_t(s_t) - r_t(s_t - s_{t,i} + c)$, is closely related to the *Difference Reward* used in multiple non-time-extended problems [3, 50]:

$$DR^i(s) = r(s) - r(s - s_i + c) \ , \qquad (8)$$

where c is independent of state s_i. The strait-forward conversion of this into single-time-step rewards is:

$$DR_t^i(s_t) = r(s_t) - r(s_t - s_{t,i} + c_t) \ , \qquad (9)$$

where c_t is independent of state s_t. The reward $d_t^i(s_t)$ is a form of $DR_t^i(s_t)$ that places greater restriction on c_t: it must be independent of *all* previous states. This restriction is needed to keep the Markov assumption as mentioned previously.

One could consider loosening the restrictions on b_t so that it is only independent of the current state. With this less restrictive, c_t, an action taken by agent i can still only affect the first term of DR_t^i: $r(s_t)$. Therefore using the less restrictive, c_t, still results in a reward that is aligned with the system reward for a single time step, and allows us additional flexibility in creating rewards. In this paper we analyze a particular form of this reward, which we call the local difference reward, dl:

$$dl_t^i(s_t) = r(s_t) - r(s_t - s_{t,i} + cl_t) \ , \tag{10}$$

where cl_t is the state entered as a result of taking action zero instead of action a_t. By replacing $s_{t,i}$ with a more local state, we can often improve sensitivity reward. Since the counterfactual is more closely related to the original action, the value of $r(s_t - s_{t,i} + cl_t)$ will be more related to $r(s_t)$, reducing the side effects that a more extreme counterfactual would cause. However, while dl is aligned with the system reward at any single time step, it breaks the Markov assumption and ultimately causes the reward to be not "aligned through time." The subtle difficulty is that values of dl_t^i get propagated back to previous states through Q-learning. If cl_t is not independent of all previous states, values that are not aligned with the system reward may be propagated back to previous states. This problem is shown in Figure 1. In this example an agent can take two actions for two time steps, ending up in one of four possible states. In addition to the possible "factual" states is also a set of "fictional" counterfactual states that are used in the computation of d. The topmost figures show that the sum of difference rewards ranks four possible final states the same way that the sum of global rewards does. However the bottom right figure shows how this is not true with the local difference reward. While at any particular time step the local difference reward is aligned with the global reward, the sum of local difference rewards is not aligned with the sum of global rewards. Even though this lack of alignment through time sometimes does not matter, experimental results presented later in this paper show that it is often important.

4 Multiagent Grid World Experiments

To evaluate the performance of QUICR-learning, we perform experiments that test the ability of agents to maximize a reward in a grid world domain. These experiments will show the relative performance of QUICR-learning, Q-learning, local Q-learning, and Q-learning using the local difference reward. In addition these experiments will give us insight into the relative tradeoffs between reward sensitivities and reward alignment between the four different rewards used in these learning algorithms.

Our grid world domain used in these experiments is a multiagent version of a standard grid world problem [41]. In the single-agent problem, at each time step, the agent can move up, down, right or left one grid square, and receives a reward (possibly zero) after each move. The observable state space for the agent is its grid coordinate and the reward it receives depends on the grid square to which it moves. In the episodic

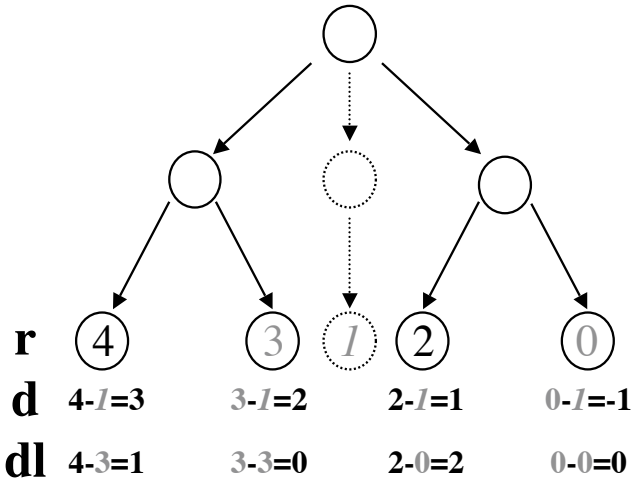

Figure 1: Alignment of Rewards. An agent starts at the top most node and takes one of two possible actions for two time steps. System reward is received at last time step shown is circles of leaf nodes. Dotted circles represent "counterfactual" states used in the computation of d. To receive the maximum system reward the agent should go left twice. Below each leaf node is the reward computed by the difference reward (d) and the local difference reward (dl). Here agents using d will order the rewards the same way as agents using r. To maximize d an agent will go left twice. However, agents using dl will order them differently. To maximize dl an agent will go right, then left.

version, which is the focus of this paper, the agent moves for a fixed number of time steps, and then is returned to its starting location.

In the multiagent version of the problem there are multiple agents navigating the grid simultaneously influencing each others' rewards. In this problem agents are rewarded for observing tokens located in the grid. Each token has a value between zero and one, and each grid square can have at most one token. When an agent moves into a grid square, it observes a token and receives a reward for the value of the token. Rewards are only received on the first observation of the token. Future observations from the agent or other agents do not receive rewards in the same episode. More precisely, r_t is computed by:

$$r_t(s_t) = \sum_i \sum_j V_j I^t_{s_{t,i}=L_j} \ , \tag{11}$$

where I^t is the indicator function which returns one when an agent in state $s_{t,i}$ is in the location of an unobserved token L_j. The global reward of the multiagent grid world problem is to observe the highest aggregated value of tokens in a fixed number of time steps T.

4.1 Learning Algorithms

In our experiments for both this grid world problem and the traffic congestion problem(presented in the next section) we tested the multiagent system using variants of the temporal difference method with $\lambda = 0$ (TD(0)). The actions of the agents were chosen using an epsilon-greedy exploration scheme and tables were initially set to zero with ties broken randomly (in the traffic congestion experiment ϵ was set to 0.05 and in the multiagent grid world experiment ϵ was set to 0.15). The learning rate was set to 0.5 (however to simplify notation we do not show the learning rates in the update equations). The four algorithms are as follows:

- Standard Q-learning is based on the full reward r_t:

$$Q(s_t, a_t) = r_t(s_t) + max_a Q(s_{t+1}, a) \ . \tag{12}$$

- Local Q-learning is only a function of the specific agent's own state:

$$Q_{loc}(s_t, a_t) = sum_j V_j I^{t,i}_{s_{t,i}=L_j} + max_a Q_{loc}(s_{t+1}, a).$$

- QUICR-learning instead updates with a reward that is a function of all of the states, but uses counterfactuals to suppress the effect of other agents' actions:

$$\begin{aligned} Q_{UICR}(s_t, a_t) = r_t(s_t) - r_t(s_t - s_{t,i} + c) \\ + max_a Q_{UICR}(s_{t+1}, a) \ , \end{aligned}$$

  where $s_t - s_{t,i} + s_b$ is the state resulting from removing agent $i's$ state and replacing it with the absorbing state c.

- DL Q-learning is similar to QUICR-learning, but uses a different counterfactual state. Instead of replacing the state $s_{t,i}$ by the absorbing state s_b, it is replaced by the state that the agent would have been in if he had taken action 0, which causes the agent to move to the right:

$$\begin{aligned} Q_{DL}(s_t, a_t) = r_t(s_t) - r_t(s_t - s_{t,i} + s^{right}_{t-1,i}) \\ + max_a Q_{DL}(s_{t+1}, a) \ , \end{aligned}$$

  where $s^{right}_{t-1,i}$ is the state to the right of $s_{t-1,i}$.

4.2 Simulation Results

In the multiagent grid world experiments we use two different token distributions to test different aspects of the learning method. In the first set of experiments we explore a domain with a structured token distribution. In this distribution the most valuable tokens are in the center. The value of the tokens then decreases from the center until the edge of the domain is reached, where the tokens again increase in value. Figure 2 shows an instance of the grid world using this distribution for the 20x20 world, used in the experiments with 40 agents.

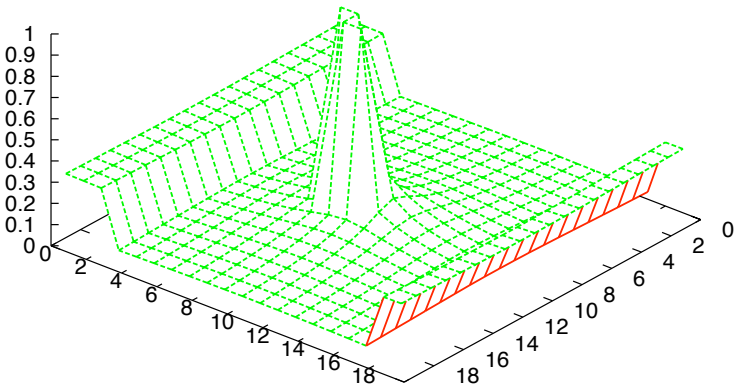

Figure 2: Edge Token Distribution.

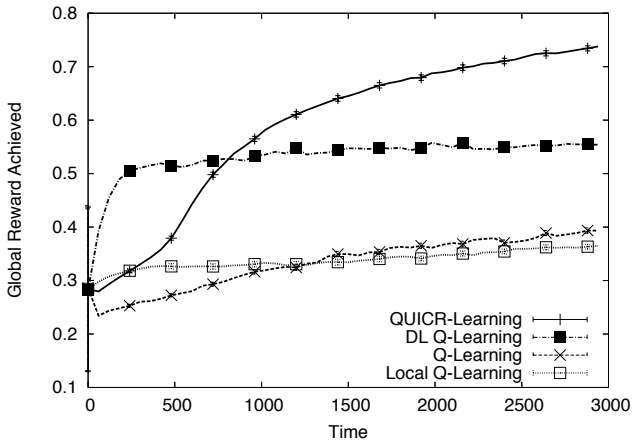

Figure 3: Multiagent Grid World Problem (40 Agents) Edge Token Distribution.

This domain is designed to create both credit assignment problems at once. Having a small set of valuable tokens in the center creates a structural credit assignment problem, since all of the agents want to observe these valuable tokens, if they are not taking into account what other agents are doing. This token distribution also creates a temporal credit assignment problem as many of the valuable tokens are located at the edge of the domain and can only be observed close to the end of an episode. To reach these tokens, agents have to take actions that have little return for many time steps.

Figure 3 shows the performance for 40 agents on a 400 unit-square world for episodes of 20 time steps (error bars of $\pm$ one σ are included). The performance measure in these figures is the sum of full rewards ($r_t(s_t)$) received in an episode, normalized so that the maximum reward achievable is 1.0. Note all learning methods are evaluated on the same reward function, independent of the reward function that they are internally using to assign credit to the agents. The results show that agents using lo-

cal Q-learning perform poorly and only achieve a highly suboptimal solution. This low performance is caused by all the agents focusing on the valuable tokens in the center of the domain. Agents using Q-learning also perform poorly since each agent as little influence on the value of its reward. Note however, that agents using Q-learning eventually do outperform agents using local Q-learning, since Q-learning is at least aligned with the system goal. Agents using QUICR-learning learn quickly in this domain and achieve the highest end result. This happens since QUICR-learning using a reward that is both aligned to the system goal and also highly sensitive to the agent's actions. Interestingly DL Q-learning performs even better than QUICR-learning at first, but then its performance stagnates as agents reach a lower performing solution. This result is caused by the dl_t reward used in DL Q-learing being even more sensitive to the agent's actions than the d reward, but not being aligned with the system level goals. The high sensitivity causes the agents to learn quickly, but ultimately they learn to maximize a reward that is slightly different than the system reward, causing a suboptimal convergence.

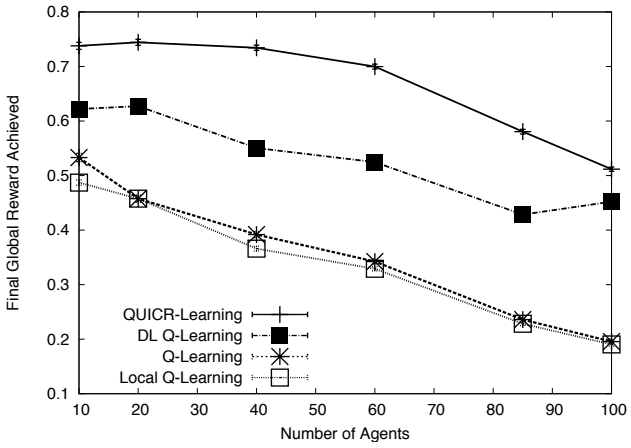

Figure 4: Scaling Multiagent Grid World Problem Edge Token Distribution.

Figure 4 explores the scaling properties for each algorithm using this token distribution. As the number of agents is increased, the difficulty of the problem is kept constant by increasing the size of the grid world, and allocating more time for an episode. Specifically the ratio of the number of agents to total number of grid squares and the ratio of the number of agents to total value of tokens is held constant. In addition the ratio of the furthest grid square from the agents' starting point to the total amount of time in an episode is also held constant (e.g., 40 agents, 20x20 grid, 20 steps, 400 agents, 63x63 grid, 63 time steps).

As predicted agents using Q-learning scale poorly, because of the low sensitivity of the system reward. Local Q-learning also scales poorly even though the local reward is highly sensitive. However, this can be explained by both algorithms simply performing barely above random in all the experiments. In contrast QUICR-learning achieves high levels of performance even when there are many agents.

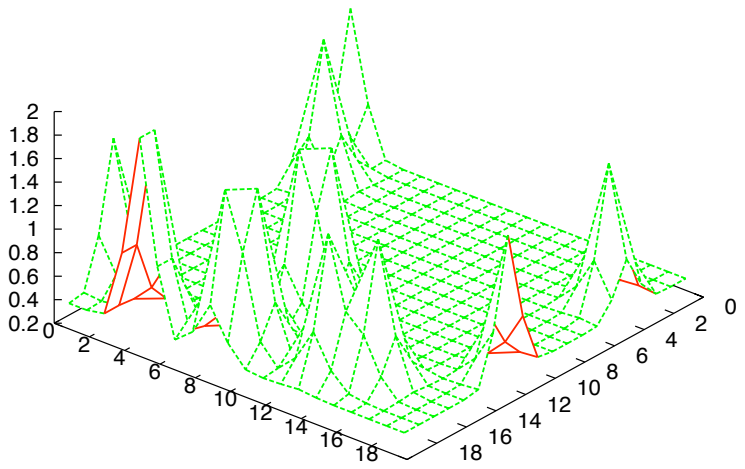

Figure 5: Random Token Distribution.

In the second set of experiments we investigate the behavior of agents in a grid world where the token values are randomly distributed. In this world, for n agents, there are $n/3$ Gaussian "attractors" whose centers are randomly distributed. Figure 5 shows an instance of the grid world using this distribution for the 20x20 world, used in the experiments with 40 agents.

Figure 6 shows the performance for 40 agents on a 400 unit-square world for episodes of 20 time steps. The results show again that local Q-learning generally produces poor results. This problem is caused by all agents aiming to acquire the most valuable tokens, and congregating towards the locations of high value tokens. In essence, in this case agents using local Q-learning compete, rather than cooperate. The agents using standard Q-learning perform even worse as the agents are plagued by the credit assignment problem associated with each agent receiving the full world reward for each individual action they take. Agents using QUICR-learning on the other hand learn rapidly, outperforming both local and standard Q-learning by a large margin. Agents using DL Q-learning initially learn even faster than agents using QUICR-learning, but converge to an inferior final solution.

Figure 7 explores the scaling properties for each algorithm. The scaling results show that agents using standard Q-learning deteriorate rapidly as the number of agents increases. Agents using QUICR-learning on the other hand are not strongly affected by the increase in the size of the problem, and outperformed local and standard Q-learners. This is because QUICR-learning agents received rewards that were both aligned with the sytem goal had high agent sensitivity (i.e.,less affected by the size of the system). This result underscores the need for using rewards that suppress the affect of other agents actions in large systems.

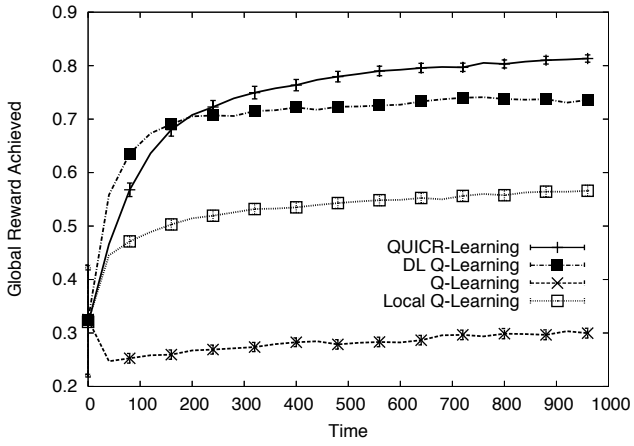

Figure 6: Multiagent Grid World Problem (40 Agents) Random Token Distribution.

5 Traffic Congestion Experiment

To evaluate the performance of QUICR-learning in another domain, we perform experiments that test the ability of agents to maximize a reward based on an abstract traffic simulation. In this experiment n drivers can take a combination of m roads to make it to their destination. Each road j has an ideal capacity c_j representing the size of the road. In addition each road has a weighting value w_j representing a driver's benefit from taking the road. This weighting value can be used to represent such properties such as a road's difficulty to drive on and convenience to destination. In this experiment a driver starts on a road chosen randomly. At every time step, the driver can choose to stay on the same road or to transfer to one of two adjacent roads. In order to test the ability of learners to perform long term planning, the global reward is zero for all time steps, except for the last time step when it is computed as follows:

$$r_t = \sum_j k_{j,t} e^{(-\frac{k_{j,t}}{c_j})} , \tag{13}$$

where $k_{j,t}$ is the number of drivers on road j at time t.

5.1 Learning Algorithms

As in the multiagent grid world problem, we test the performance of the following four learning methods:

- Standard Q-learning is based on the full reward r_t:

$$Q(s_t, a_t) = r_t(s_t) + max_a Q(s_{t+1}, a) . \tag{14}$$

- Local Q-learning is only a function of the specific driver's own road, j:

$$Q_{loc}(s_t, a_t) = k_{j,t} e^{(-\frac{k_{j,t}}{c_j})} + max_a Q_{loc}(s_{t+1}, a) .$$

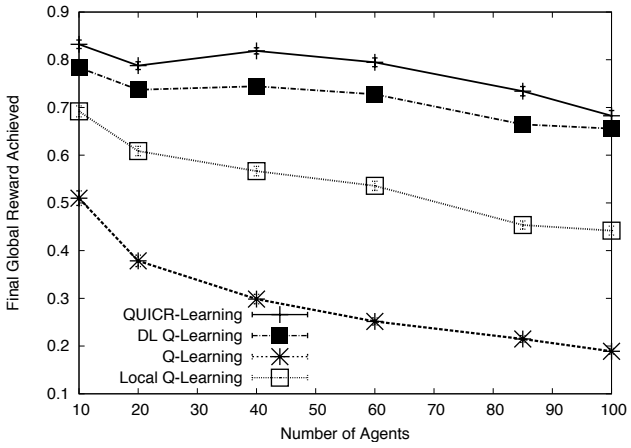

Figure 7: Scaling Multiagent Grid World Problem Random Token Distribution.

- QUICR-learning instead updates with a reward that is a function of all of the states, but uses counterfactuals to suppress the effect of other driver's actions:

$$Q_{UICR}(s_t, a_t) = r_t(s_t) - r_t(s_t - s_{t,i} + s_b) + max_a Q_{UICR}(s_{t+1}, a) \ ,$$

  where $s_t - s_{t,i} + s_b$ is the state resulting from removing agent $i's$ state and replacing it with the absorbing state s_b.

- DL_t Q-learning is similar to QUICR-learning, but uses a simpler form of counterfactual state. Instead of replacing the state $s_{t,i}$ by the absorbing state s_b, it is replaced by the state that the driver would have been in if he had taken action 0, which in this case is the same road he was on the previous time step : $s_{t-1,i}$. The resulting update equation is:

$$Q_{DL}(s_t, a_t) = r_t(s_t) - r_t(s_t - s_{t,i} + s_{t-1,i}) + max_a Q_{DL}(s_{t+1}, a) \ .$$

5.2 Simulation Results

Experimental results on the traffic congestion problem show that QUICR-learning learns more quickly and achieves a higher level of performance than the other learning methods (Figure 8). While standard Q-learning is able to improve performance with time, it learns very slowly. This slow learning rate is caused by Q-learning's use of the full system reward $r_t(s_t)$, which is a function of the actions of all the other drivers. When a driver takes an action that is beneficial, the driver may still receive a poor reward if some of the fifty nine other drivers took poor actions at the same time. In contrast, local Q-learning learns quickly, but since it uses a reward that is not aligned

with the system reward, drivers using local Q-learning eventually learn to take bad actions. Early in learning drivers using local Q-learning perform well as they learn to use the roads with high capacity and higher weighting. However, as learning progresses the drivers start overusing the roads with high weighting, since their reward does not take into account that using other roads would benefit the system as a whole. This system creates a classic Tragedy of the Commons scenario. By over utilizing the "beneficial" roads, the drivers end up being worse off than if they had acted in a cooperative manner. Drivers using DR_t Q-learning have similar problems because although they are aligned at each time step, they are not aligned across time steps.

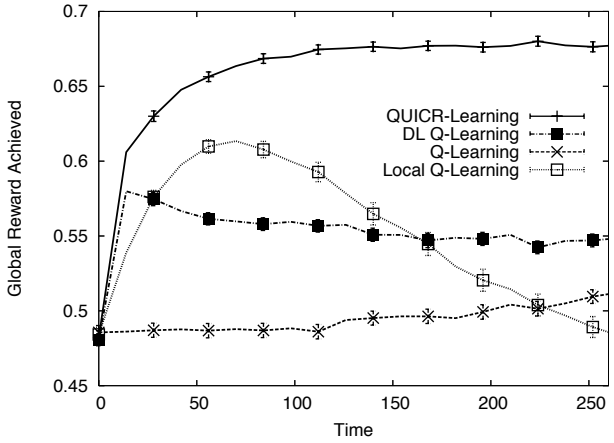

Figure 8: Traffic Congestion Problem (50 Agents).

To test the ability of the learning methods to scale to large numbers of drivers, we have a set of scaling experiments ranging from 40 to 500 drivers. Scaling is performed by adjusting the capacities of the roads in proportion to the number of drivers, and normalizing the system reward. The results of the scaling experiments are shown in Figure 9. These results show that drivers using Q-learning do not scale well, as their performance becomes even lower than their initially low level when the number of drivers increases. Drivers using local Q-learning are mostly unaffected by scaling, maintaining a constant low level of performance. In contrast, drivers using QUICR-learing scale very well when there are a large number of drivers, with performance increasing as the number of drivers increases. This increase in performance can be attributed the discrete nature of the reward function. When there are a large number of drivers it is easier to position them in such a way that the distribution is close to optimal.

6 Single-Time-Step Problems

While reinforcement learning is usually used in problems where agents need to take a sequence of actions that maximize a sum of rewards, it is illustrative to show how

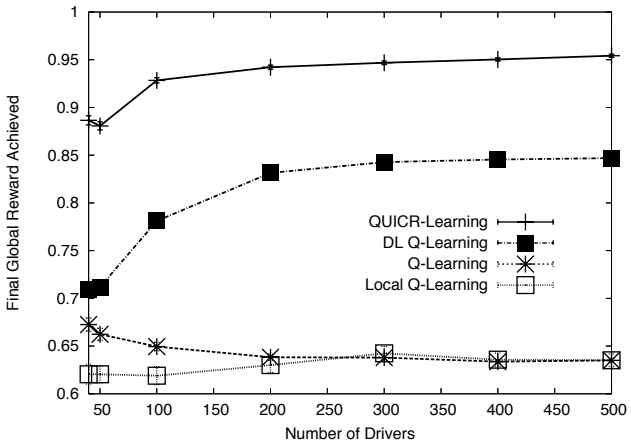

Figure 9: Scaling for Traffic Congestion Problem.

the learning algorithms perform on a single-time-step version of the traffic congestion problem. In particular, we claim that while the sum of dl_t rewards is not aligned with a sum of global rewards r_t, it is aligned at every particular time step. Therefore in a single-time-step problem the dl reward should rank actions the same as the global reward.

In this experiment again the drivers are trying to find the best lanes to be in to minimize congestion. However, in this version they can choose to be in any lane, instead of only being able to chose the lane to right or left of their previous lane. In addition in this experiment drivers only choose their lanes once, and the rewards reflect the drivers choice of lane for this single time step. The results of this experiment are shown in Figure 10. Again drivers using local Q-learing initially reaches a reasonable performance level, but quickly decline in performance, as the local reward is not aligned with the global one. Also as before, drivers using the global reward learn very slowly. As before drivers using QUICR-learning learn quickly and achieve a high value of the reward. However, drivers using DL Q-learning also perform nearly identical as drivers using QUICR-learning as the local difference reward becomes identical to the difference reward in the single-time-step case.

7 Related Work

Using machine learning techniques to coordinate multiagent systems has been approached from several different ways in the literature originating from a diverse set of field. While many of the methods are interrelated we will summarize them by dividing them up into four main categories:

- Subgoal Methods: These solutions use either hand made or automatically generated subgoals to significantly reduce the complexity of multiagent learning.

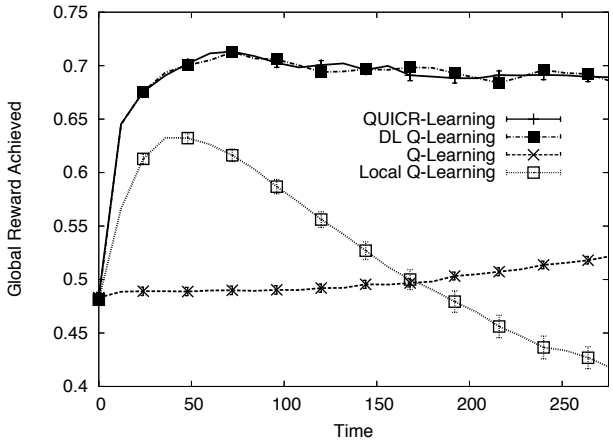

Figure 10: Single Action Traffic Congestion Problem.

- Factoring and Local Methods: These solutions setup the multiagent problem in such a way that a good global solution can be formed while agents optimize locally.
- Hierarchical MDPs: These solutions look at ways of breaking the multiagent problem into hierarchies, either through time or space.
- Game Theory: These techniques apply game theoretic analysis to the multiagent problem.

The applicability of these techniques range in how much information is known about the multiagent system, how tightly coupled the agents are, how many agents there are and the strictness of the convergence requirements.

Subgoal Methods

Currently subgoal methods that leverage domain knowledge to exploit a problem's substructure, are some of the best performing multiagent learning methods. In robotic soccer for example, player specific subtasks are used, followed by tiling to provide good convergence properties [40]. With proper implementation it was shown that in this domain effective learning could be accomplished using actual robots without any simulation. In a robot coordination problem for the foraging domain it was also shown that specific rules induce good division of labor [22]. In addition with multi-robot path finding spacial decomposition has been used to simplify path finding in both single agent and multiagent problems [39].

Factoring and Local Methods

As the number of agents becomes large it is difficult for multiagent reinforcement systems to perform well [43]. One way to address this issue is to decompose the multiagent

systems into independent groups, where learning within groups of agents (or groups of variables) becomes an easier problem [16, 24, 26, 8, 38, 7, 34, 30]. These techniques typically take advantage of a certain amount of linearity of the reward function [11, 37]. In domains where groups of agents can be assumed to be independent, the task can be decomposed by learning a set basis functions used to represent the value function, where each basis only processes a small number of the state variables [17]. In domains that can be decomposed into "sub-agents" with additive rewards, Q-decomposition can be used to apply Q-learning to multiagent problems [35]. This work has been extended by using bounding methods to improve performance through state pruning [33]. On a multiagent resource allocation problem, influence diagrams have been used with decomposition to simplify the multiagent problem [27]. Also multiagent Partially Observable Markov Decision Processes (POMDPs) can be simplified through piecewise linear rewards [29]. In addition, in some domains neighborhood information can be used to create local rewards that greatly speed up multiagent learning [5].

Hierarchical MDPs

Hierarchical methods can be used to simplify both single-agent and multiagent reinforcement learning problems [32, 28, 6]. Task decomposition in single agent RL can be achieved using hierarchical reinforcement learning methods such as MAXQ value function decomposition [13]. This work was extended to multiagent systems, taking advantage of hierarchical structure by limiting agent coordination to the highest levels of the hierarchy [28]. This technique was shown to be effective in an scheduling task for a set of automated vehicles [28]. In addition hierarchies have been addressed by extending the notion of a set of basis function used in factored MDPs to a system tree used to represent the hierarchy [18]. These hierarchies have been used to improve efficiency in multiagent problems [18]. Hierarchies have also been used to improve communication between agents in a multiagent gridworld problem [14].

Game Theory

Game theoretic techniques have also been successfully applied to multiagent reinforcement learning problems [36, 25, 21, 19, 12, 31]. For a small number of agents, game theoretic techniques were shown to lead to a multiagent Q-learning algorithm proven to converge to Nash equilibria [20]. This process was generalized to correlated equilibria where different methods were obtained by choosing which equilibria was obtained [19]. For use in stochastic games, convergence properties in Q-learning were improved with the "Win or Learn Fast" (WoLF) algorithm that varied the learning rate of the agents depending on their context [9]. Also applied to stochastic games methods based on Bayesian methods were shown to improve multiagent learning by providing better exploration capabilities [12]. In economics Groves' mechanism [15] presents a way of providing incentives so that a players payoff is aligned with the global payoff. In addition Vickrey tolls [48] present a mechanism for isolating a particular players cost to a system.

8 Discussion

Using Q-learning to learn a control policy for a single agent in a coordination problem with many agents is difficult, because an agent will often have little influence over the reward it is trying to maximize. In our examples, an agent's reward received after an action could be influenced by as many as 3200 other actions from other time-steps and other agents. Even temporal difference methods that perform well in single agent systems will be overwhelmed by the number of actions influencing a reward in the multiagent setting. To address this problem, this paper introduces QUICR-learning, which aims at reducing the impact of other agent's actions without assuming linearly separable reward functions. Within the Q-learning framework, QUICR-learning uses the difference reward computed with immediate counterfactuals. While eliminating much of the influence of other agents, this reward is shown mathematically to be aligned with the global reward: agents maximizing the difference reward will also be maximizing the global reward. Experimental results in a traffic congestion problem and a grid world problem confirm the analysis, showing that QUICR-learning learns in less time than standard Q-learning, and achieves better results than Q-learning variants that use local rewards and assume linear separability. While this method was used with TD(0) Q-learning updates, it also extends to TD(λ), Sarsa-learning and Monte Carlo estimation.

In our experiments an agent's state is never directly influenced by the actions of other agents. Despite this, the agents are still tightly coupled by virtue of their reward. Agents can, and did affect each other's ability to achieve high rewards, adding complexity that does not exist in systems where agents are independent. In addition even though agents do not directly influence each other's states, they indirectly affect each other through learning: an agent's actions can impact another agent's reward, and the agents select actions based on previous rewards received. Hence an agent's action at time step t does affect other agents at $t' > t$ through their learning algorithms. The mathematics in this paper does not address these indirect influences and is a subject of further research. However, experimental evidence shows that agents can still cooperate despite these indirect effects. In fact even when agents directly influence each other's states, in practice they may still cooperate effectively as long as they use agent-sensitive rewards that are aligned with the system reward as has been shown in experiments presented in [1] .

References

[1] A. Agogino and K. Tumer. Unifying temporal and structural credit assignment problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems*, New York, NY, July 2004.

[2] A. Agogino and K. Tumer. QUICR-learning for multi-agent coordination. In *Proceedings of the 21st National Conference on Artificial Intelligence*, Boston, MA, July 2006.

[3] A. K. Agogino and K. Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic environments. *Journal of Autonomous Agents and Multi Agent Systems*, 17(2):320–338, 2008.

[4] A. K. Agogino and K. Tumer. Efficient evaluation functions for evolving coordination. *Evolutionary Computation*, 16(2):257–288, 2008.

[5] J. A. Bagnell and A. Y. Ng. On local rewards and the scalability of distributed reinforcement learning. In *NIPS 18*, 2006.

[6] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[7] M. Bhaskara, D. Latham, S. Russel, and C. Guestrin. Concurrent hierarchical reinforcement learning. In *IJCAI*, Edinburgh, Scotland, 2005.

[8] Craig Boutilier, Richard Dearden, and Moisés Goldszmidt. Stochastic dynamic programming with factored representations. *Artif. Intell.*, 121(1-2):49–107, 2000.

[9] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250, 2002.

[10] J. A. Boyan and A. Moore. Learning evaluation functions for global optimization and boolean satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI Press, 1998.

[11] Darius Braziunas and Craig Boutilier. Minimax regret-based elicitation of generalized additive utilities. In *Proceedings of the Twenty-third Conference on Uncertainty in Artificial Intelligence*, pages 25–32, 2007.

[12] G. Chalkiadakis and C. Boutilier. Coordination in multiagent reinforcement learning: A bayesian approach. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, Melbourne, Australia, July 2003.

[13] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence*, 13:227–303, 2000.

[14] Mohammad Ghavamzadeh and Sridhar Mahadevan. Learning to communicate and act using hierarchical reinforcement learning. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1114–1121, Washington, DC, USA, 2004. IEEE Computer Society.

[15] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[16] C. Guestrin, D. Koller, R. Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, March 2003.

[17] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, 2002.

[18] Carlos Guestrin and Geoffrey Gordon. Distributed planning in hierarchical factored mdps. In *In the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 197 – 206, Edmonton, Canada, 2002.

[19] Keith Hall. Correlated q-learning. In *Proceedings of the Twentieth International Conference on Machine Learning*, pages 242 – 249, 2003.

[20] J. Hu and M. P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250, June 1998.

[21] Junling Hu and Michael P. Wellman. Nash q-learning for general-sum stochastic games. *J. Mach. Learn. Res.*, 4:1039–1069, 2003.

[22] C. Jones and M. J. Mataric. Adaptive division of labor in large-scale multi-robot systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-03)*, pages 1969–1974, Las Vegas, NV, July 2003.

[23] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[24] M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 740–747, 1999.

[25] M. Kearns and Y. Mansour. Efficient nash computation in large population games with bounded influence. In *Proceedings of UAI*, 2002.

[26] D. Koller and R. Parr. Policy iteration for factored MDPs. In *Unvertainty in Artificial Intelligence*, pages 236–334, 2000.

[27] Daphne Koller and Brian Milch. Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, 45:181–221, 2003.

[28] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 246–253, Montreal, Canada, 2001. ACM Press.

[29] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, and S. Marsella. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003.

[30] Sébastien Paquet, Ludovic Tobin, and Brahim Chaib-draa. An online pomdp algorithm for complex multiagent environments. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 970–977, New York, NY, USA, 2005. ACM.

[31] D. Parkes and S. Singh. An MDP-based approach to online mechanism design. In *NIPS 16*, pages 791–798, 2004.

[32] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 10*, pages 1043–1049. MIT Press, 1998.

[33] Pierrick Plamondon, Brahim Chaib-draa, and Abder Rezak Benaskeur. A q-decomposition and bounded rtdp approach to resource allocation. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[34] Pascal Poupart, Craig Boutilier, Relu Patrascu, and Dale Schuurmans. Piecewise linear value function approximation for factored mdps. In *Eighteenth national conference on Artificial intelligence*, pages 292–299, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[35] Stuart J. Russell and Andrew Zimdars. Q-decomposition for reinforcement learning agents. In *International Conference on Machine Learning*, 2003.

[36] Tuomas Sandholm and Robert H. Crites. On multiagent q-learning in a semi-competitive domain. In *IJCAI '95: Proceedings of the Workshop on Adaption and Learning in Multi-Agent Systems*, pages 191–205, London, UK, 1996. Springer-Verlag.

[37] Scott Sanner and Craig Boutilier. Approximate solution techniques for factored first-order mdps. In *Proceedings of the Seventeenth Conference on Automated Planning and Scheduling*, pages 288–295, 2007.

[38] D. Schuurmans and R. Patrascu. Direct value-approximation for factored mdps. In *Advances in Neural Information Processing*, volume 14, 2001.

[39] A. Singh, A. Krause, C. Guestrin, W. Kaiser, and M. Batalin. Efficient planning of informative paths for multiple robots. In *21st International Joint Conference on Artificial Intelligence (IJCAI 2007)*, Hyderabad, January 2007.

[40] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 2005.

[41] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[42] K. Tumer and A. Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 330–337, Honolulu,HI, May 2007.

[43] K. Tumer, A. Agogino, and D. Wolpert. Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 378–385, Bologna, Italy, July 2002.

[44] K. Tumer and N. Khani. Learning from actions not taken in multiagent systems. *Advances in Complex Systems*, 12:455–473, 2009.

[45] K. Tumer, Z. T. Welch, and A. Agogino. Aligning social welfare and agent preferences to alleviate traffic congestion. In *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Estoril,Portugal, May 2008.

[46] K. Tumer and D. Wolpert, editors. *Collectives and the Design of Complex Systems*. Springer, New York, 2004.

[47] K. Tumer and D. H. Wolpert. Collective intelligence and Braess' paradox. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 104–109, Austin, TX, 2000.

[48] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[49] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3/4):279–292, 1992.

[50] D. H. Wolpert and K. Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2/3):265–279, 2001.