Unifying Temporal and Structural Credit Assignment Problems *

Adrian K. Agogino
UCSC, NASA Ames Research Center
Mailstop 269-3
Moffett Field, CA 94035
adrian@email.arc.nasa.gov

Kagan Tumer
NASA Ames Research Center
Mailstop 269-3
Moffett Field, CA 94035
kagan@email.arc.nasa.gov

Abstract

Single-agent reinforcement learners in time-extended domains and multi-agent systems share a common difficulty known as the credit assignment problem. Multi-agent systems have the structural credit assignment problem of determining the contributions of a particular agent to a common task. Instead, time-extended single-agent systems have the temporal credit assignment problem of determining the contribution of a particular action to the quality of the full sequence of actions. Traditionally these two problems are considered different and are handled in separate ways. In this article we show how these two forms of the credit assignment problem are equivalent. In this unified framework, a single-agent Markov decision process can be broken down into a single-time-step multi-agent process. Furthermore we show that Monte Carlo estimation or Q-learning (depending on whether the values of resulting actions in the episode are known at the time of learning) are equivalent to different agent utility functions in a multi-agent system. This equivalence shows how an often neglected issue in multi-agent systems is equivalent to a well-known deficiency in multi-time-step learning and lays the basis for solving time-extended multi-agent problems, where both credit assignment problems are present.

1. Introduction

The structural credit assignment problem of determining how a single agent's actions contribute to a system that involves the actions of many agents is inherent in multi-agent domains. For a reinforcement learning agent to learn properly, this credit assignment problem needs to be resolved and the agent needs to receive the appropriate reinforcement. Robotic soccer is a well studied domain that clearly exhibits this form of credit assignment problem, where learning algorithms need to judge a particular player's role in achieving the overall goal of the mutli-agent system of winning the game [6]. This structural credit assignment problem has been studied in other domains including foraging robots [5], network routing [15] and bimatrix games [3]. In these systems the credit assignment problem was handled implicitly by creating a reward structure that credited an agent's role in the performance of a larger system.

In a single-agent domain, the temporal credit assignment problem is concerned with how an action taken at a particular time step affects the final outcome. For example, if a player wins a game of checkers, it may be difficult for that player to determine which of his many moves were the most important in helping him win, and which moves may have actually been detrimental. Many reinforcement learning algorithms have been derived to assign proper credit assignment including Q-learning, Sarsa and TD(λ)[13, 8, 10]. The goal of these algorithms is to make the learner converge to the correct policy, in a speedy manner, or to at least make a good tradeoff between correctness and speed.

This paper poses the single-agent time-extended problem as a multi-agent single-time-step problem, transforming the temporal credit assignment problem into a structural credit assignment problem. This credit assignment problem is then solved with multi-agent utilities, where credit is assigned through agent-specific utility functions. In our solution an agent evaluates its role in the outcome of a global utility function over all agents through a private utility function that is both "aligned" with the global utility, yet is sensitive to the agent's actions. In many cases the multi-agent solution is equivalent to popular reinforcement learners. Showing this equivalence is beneficial in many ways: i) It allows users to pose many problems either as a structural or a temporal credit assignment problem, and choose the one that is best suited for the domain; ii) it highlights potential pitfalls of some approaches to structural credit assignment by expressing their problems

* Appears in: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems.* pp. 978-985, New York, NY, 2004

as well known deficiencies of temporal credit assignment algorithms; iii) it lays the basis for deriving principled solutions to time-extended multi-agent systems, where both credit assignment problems are present.

In this work, Section 2 describes the structural credit assignment problem and presents a solution in terms of learnable private utilities that are aligned with a global utility. Section 3 summarizes relevant issues in the standard temporal credit assignment problem, where a single agent needs to determine how an action affects the entire sequence of rewards received after that action. Section 4 then shows how the two credit assignment problems can be unified by transforming a single-agent multi-time-step problem into a structural credit assignment problem, allowing temporal credit assignment problems to be posed as structural credit assignment problems. Sections 5, 6 and 7 then show how the new structural credit assignment problem can be solved using three utilities presented in section in 2. The application of two of the utilities illustrates the relationship between two popular temporal credit assignment problem methods and the multi-agent concepts of utility alignment, learnability and system observability. The application of the third utility shows how a subtle pitfall common in mutli-agent utilities relates to an obvious problem in multi-time step systems.

2. Structural Credit Assignment

Many structural credit assignment problems in multi-agent systems have been successfully addressed with multi-agent utilities [4, 15, 11, 1, 7, 12]. This section summarizes how to create an effective multi-agent utility that has two important properties: It is both aligned with a global utility over all agents, and it is easy for the agents to learn. These properties are called "factoredness" and "learnability" respectively. Results from this section are used later to show how a multi-time-step single agent system can be cast as a single-time-step multi-agent problem. To avoid the confusion by overloading the word "agent", we use the world "agent" exclusively in the single-agent problem. Instead we use the term "node" in place of "agent" in the mutli-agent extensions we discuss, and in particular we call such systems "multi-node systems."

The goal of the multi-node system is to maximize a world utility function, $G(z)$, which is a function of the joint move of all nodes in the system, z. Instead of maximizing $G(z)$ directly, each node, i, tries to maximize its **private utility function** $g_i(z)$. The goal of this section is to solve the structural credit assignment problem in this context, i.e., to create private utility functions that lead the multi-node system to high values of $G(z)$. Note that in many systems an individual node i will only influence some of the components of z. We use the notation z_i to refer to the parts of the state that are influenced by the actions of i. The vector z_i is the same size as z and is equal to z except that all the components that i does not influence are set to zero. In this notation $z - z_i$ expresses the states of all the nodes other than i. This notation is used instead of standard vector decomposition to make the addition and subtraction of vector components explicit.

2.1. Node Utilities

While each node is trying to maximize its private utility, as a whole we want the system to try to maximize the global utility. To do this, we want the nodes' private utilities to be aligned with the global utility. We call such an aligned utility a "factored utility." Formally, a private utility is **factored** when for each agent i:

$$g_i(z) \quad \geq \quad g_i(z') \;\Leftrightarrow\; G(z) \geq G(z') \quad \forall z, z' \;\; s.t. \;\; z - z_i = z - z'_i \quad .$$

Intuitively, for all pairs of vectors z and z' that differ only for agent i, a change in i's state that increases its private utility cannot decrease the world utility.

As a trivial example, any system in which all the private utility functions equal G is factored [2]. However such systems often suffer from low signal-to-noise, a problem that get progressively worse as the size of the system grows. This is because for large systems where G's sensitively depends on all components of the system, each agent may experience difficulty discerning the effects of its actions on G. As a consequence, each i may have difficulty achieving high g_i. We call this signal/noise effect learnability. Intuitively learnability is the ratio of the sensitivity of the utility to $i's$ actions, to the sensitivity of the utility to the actions of all other agents. So at a given state z, the higher the **learnability**, the more $g_i(z)$ depends on the move of agent i, i.e., the better the associated signal-to-noise ratio for i.

2.2. Fully Observable Difference Utility

A factored utility that has been shown to be easier to learn than the global utility in domains where z is fully observable is the **difference utility** [15], given by:

$$DU_i(z) = G(z) - G(z - z_i) \; . \tag{1}$$

This utility is factored because the second term does not depend on the state of i, and thus the only way

i can change the value of the difference utility is by changing the value of the first term, which is the global utility. Intuitively, the second term of the difference utility is the value of the global utility without node i. The difference utility then quantifies node i's contribution to the global utility. In addition to being factored it can be proven that in many circumstances, especially in large problems, that DU_i has higher learnability than does the global utility [15]. This is mainly due to the second term of the DU_i, which removes a lot of the effect of other agents (i.e., noise) from i's utility.

The fully observable difference utility has been successfully applied to various domains, including packet routing over a data network [15], congestion games [16], data downloads from a constellation of satellites [14], and multi-agent gridworlds [11].

2.3. Partially Observable Difference Utility

In many cases, it may be impossible to compute $DU_i(z)$ (or $G(z)$) because some of the component values of z are unknown to node i. We will denote the components of z that are known by i using the vector z^{o_i} and the part of z that is unknown to i using the vector z^{h_i}. The vector z^{o_i} is the same as z except that all the elements that are unknown to i are set to zero. We call the known components the **observable components** of the worldline. The vector z^{h_i} conversely contains all of the values that are not observable. The vector z is the sum of these two vectors: $z = z^{o_i} + z^{h_i}$. If z does not equal z^{o_i}, then node i may not be able compute $DU_i(z)$ directly. Instead we must approximate it using the information in z^{o_i}. One way to do this is to simply use z^{o_i} as the parameter to the difference utility. We will call this utility the **truncated difference utility** (TDU) since the non-observable components are essentially truncated out [1]. The TDU is given by:

$$TDU_i(z) = DU_i(z^{o_i}) = G(z^{o_i}) - G(z^{o_i} - z_i^{o_i}) \; . \quad (2)$$

This utility has been shown to be highly learnable, since the second term removes much of the noise caused by the actions of other agents [1]. In fact TDU_i in general is even more learnable than DU_i since z^{o_i} is likely to contain information pertinent to node i whereas z may contain a lot of irrelevant information. The main problem with this utility however is that it is not factored with respect to $G(z)$. This utility is only factored insofar as $G(z^{o_i})$ approximates $G(z)$. This means that a node could take actions that improve the value of its TDU_i, yet reduce the value of the global utility.

Another alternative to truncating out the non-observable components is to estimate the value of the difference utility given observable components. We will call this utility the **estimated difference utility** (EDU) which is given by:

$$\begin{aligned} EDU_i(z) &= E[DU_i(z)|z^{o_i}] \\ &= E[G(z)|z^{o_i}] - E[G(z - z_i)|z^{o_i}] \; , \end{aligned} \quad (3)$$

where $E[\cdot|z^{o_i}]$ is the expected value over non-observed states. While this utility is also not factored with respect to $G(z)$, in general $E[G(z)|z^{o_i}]$ is closer to $G(z)$ than $G(z^{o_i})$ is, and hence is more likely to be factored than TDU [1]. Any action that an agent takes to increase the value of the EDU must increase the value of $E[G(z)|z^{o_i}]$, since its action is removed from the second term of the EDU. When a good estimate is used, an action that increases the value of $E[G(z)|z^{o_i}]$, is very likely to increase the value of the global utility, $G(z)$. Similarly an action that increases the value of TDU must also increase the first term of the TDU, $G(z^{o_i})$. However when many of the components of z are not observable, there are many possible actions that will increase $G(z^{o_i})$, but will not increase $G(z)$, due to interactions with non-observable components.

3. Temporal Credit Assignment Problem

In a typical temporal credit assignment problem, an agent takes a sequence of actions, transitions through a sequence of states, and receives a sequence of rewards. The global utility for such a system in the episodic case is the undiscounted sum of rewards:

$$G(z) = \sum_t R_t(z) \; . \quad (4)$$

We use the undiscounted version for simplicity, since this paper uses an episodic "finite-horizon" model of learning where discounting is not needed. When learning is not episodic, discounting must be used to avoid infinite sums. In such systems the utility at a time step is the discounted sum of future rewards, $G(z) = \sum_t \gamma^t R_t(z)$, where γ is the discount factor in the range [0 1]. We will focus on problems where an agent chooses its actions based on the estimates of future rewards stored in a "Q-table." This Q-table is indexed by all the possible states and actions, where the value $Q(s, a)$ is the estimation of the sum of future rewards when action a is taken in state s. The credit assignment problem in this case consists of determining how an action

1 This utility is called "TTU" in [1].

at time step t, a_t, affects all of the rewards after time step t.

Now let us summarize versions of two reinforcement learning methods that address this problem for simple deterministic domains: First-visit Monte Carlo estimation and Q-learning [9]. With Monte Carlo estimation, an action is given credit for all the subsequent rewards. Therefore the Q-table estimate of the future rewards resulting from action a_t in state s_t is based on the rewards obtained after the action was taken. In deterministic Monte Carlo estimation, the Q-table value, estimating the undiscounted sum of future rewards after action a_t is taken in state s_t is:

$$Q_{MC}(s_t, a_t) = \sum_{t' \geq t} R_{t'} \ , \tag{5}$$

where Q_{MC} is the Q-table for the Monte Carlo estimator. Monte Carlo estimation works best when the value of the future rewards obtained after an action are very dependent on that action. However, in some domains many of the values of rewards received after time t may not be dependent on the action a_t. In such cases the Q-table estimate for action a_t in state s_t may contain a lot of noise since it includes reward values that are primarily a function of future actions. If the future actions change, the value of action a_t in state s_t may be very different. In essence, the temporal credit assignment problem is only partially solved by this method.

In contrast to Monte Carlo estimation, a Q-learner only gives full credit to the immediate reward, and instead uses other Q-values to estimate the values of future rewards. In Q-learning the Q-table value for action a_t taken in state s_t is:

$$Q_{QL}(s_t, a_t) = R_t + \max_a Q_{QL}(s_{t+1}, a) \ , \tag{6}$$

where Q_{QL} is the Q-table used by the Q-learner. Agents using Q-learning can often learn more quickly than agents using Monte Carlo learning, when an agent's action has much more influence on its immediate reward than on future rewards. In addition the Q-tables can be updated after every action with Q-learning, without waiting for the end of the episode. Note that for simplicity we use the deterministic form for both learning methods, because their update rules are cleaner in deterministic problems. However the conclusions of this paper do not depend on this determinism. In the next section we will show how these two reinforcement learning methods can be seen as forms of difference utilities.

4. Unified View of Credit Assignment

In this section we show how any Markov Decision Process (MDP) for a single-agent system can be posed as a single-time-step multi-node problem. In a single-agent MDP system there is a set of states and transitions between the states. The agent starts in a start-state and then transitions through the state-transition diagram, receiving rewards depending on the transitions taken. In some cases the agent will continue making transition until it reaches an absorbing state. Instead, this paper will use a "finite horizon" model, where an agent moves for a fixed number of time steps after starting in its start-state. One of the most important properties of an MDP is that they are "memoryless" in that the expected value of future rewards that will be received after an agent enters a state will be independent of the set of states that the agent was in before. The best transition that an agent can take therefore can be determined solely on its current state. Figure 1 (top) shows a simple MDP with only four states with two transitions per state.

This section turns the MDP into a multi-node problem by first assigning a node to every state of the MDP. The notation $s(i)$ is used to indicate the state for a node i. An action for a node i corresponds to a transition out of state $s(i)$. The nodes' actions determine the transitions that the agent will take from its state. Note that the agent now simply follows the "actions" of the nodes and performs no learning of its own. All the learning takes place in the nodes. The actions of all of the nodes, z, therefore define the path an agent will take through the state-transition diagram of the MDP, given the agent's start-state. The sum of rewards received on this path define the global utility, $G(z)$. These nodes have a simple single-time-step learning task of mapping their immediate private utility values to their actions. This mapping can be stored in a simple single-state Q-table over actions for each node. In a deterministic domain, the update rule for this Q-table after taking action a is simply:

$$Q_i(a) = g_i \ , \tag{7}$$

where g_i is the utility the node is trying to maximize. Note that this is a much smaller Q-table than the ones used in Q-learning and Monte Carlo estimation, since it is only for one state.

An example multi-node system is shown in figure 1 (middle) where each of the four states in the MDP corresponds to one of four nodes. The action vector $z = [1\ 1\ 1\ 2]^T$ encodes a set of four actions, one per node. For this action vector, the agent in the MDP transitions right twice receiving a reward of R_3 and R_4. The global utility for z, $G(z)$, is therefore $R_3 + R_4$. Computing global utility for the action vector $z - z_i$ will require more definitions. Consider the case where we want to compute the utility $G(z - z_i)$ for node 3 in the

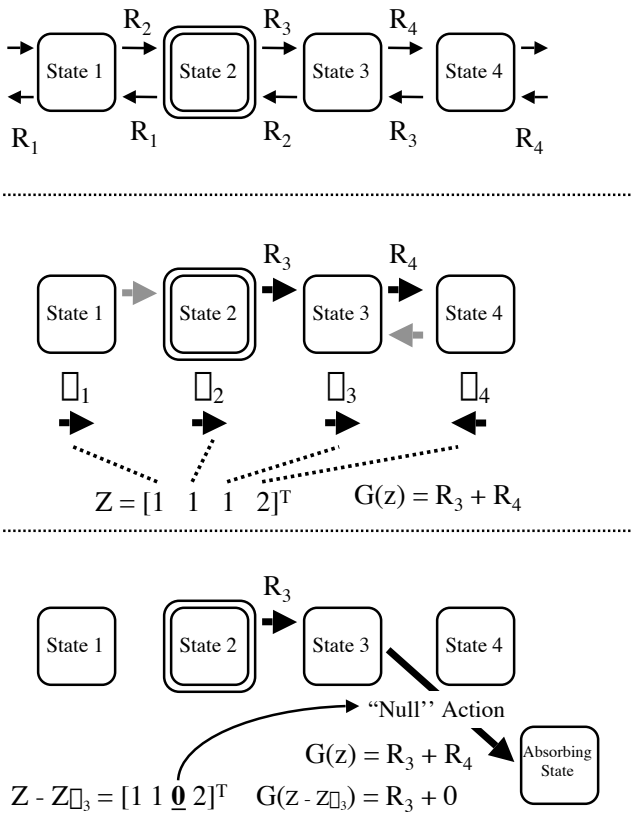

Figure 1. Four State MDP. (Top) Agent starts in State 2 and makes transitions for two time steps, receiving two rewards. (Middle) In multi-node version, a node is assigned to each state. A node's action is a choice of a transition. The choice is encoded as either a 1 (right-transition) or a 2 (left-transition) in the action vector, z. (Bottom) The action vector $z - z_{i_3}$ has a zero in its third element. This corresponds to a "zero action" transitioning the agent to an absorbing state.

example, where $z - z_i = [1\ 1\ 0\ 2]^T$. The third element of this action vector is zero, which does not correspond to any transition. The transition the agent would take out of the third state is undefined. We will therefore define this "zero" action to correspond to a "null" transition, which will always return a reward of zero and will transition into an absorbing state as shown in Figure 1 bottom. All rewards after this transition is taken will have a value of zero. The value of $G(z-z_i)$ is therefore equal to R_3, since the agent takes the right-action from state 2, receiving a reward of R_3 and then takes the null transition from state 3. Note that other definitions can be made for the zero-action, and depending on the encoding, the zero-action may refer to an actual transition. However the results shown in this paper are based on a transition to an absorbing state, and assume that actions are encoded so that an action encoded as a zero never refers to an actual transition in the MDP.

Let us now describe the learning that takes place in this MDP. This paper uses an episodic model of learning where the agent starts at a start-state at the beginning of an episode and moves according to the transitions available to the MDP. If the agent is using Monte Carlo estimation or Q-learning, it makes the decisions about which transition to take from a state, based on the Q-table values associated with that state. An agent using Monte Carlo estimation will update the Q-table values at the end of the episode, based on the reward values received during the episode. An agent using Q-learning will update the Q-table values during the course of an episode. In the multi-node version of the MDP, each node will perform a single non-null action at the beginning of the episode. This action will be determined from the small Q-table used by each node, which contains estimate values only for the node's state. Each node will then update its Q-table using its private utility, either at the end of the episode or during the episode, depending on the private utility used.

As an example take the gridworld problem (Figure 2). In this classic problem, the agent can move from grid square to grid square, until it reaches a terminal state. The agent can move in four directions, and the state is determined by which grid square the agent is in. This problem can be broken down into nodes, where each grid square is assigned a node. At the beginning of the episode each node independently chooses an action from one of four possible moves. The MDP agent then follows the actions chosen by the nodes, until it reaches the terminal square. All the nodes associated with grid squares that the gridworld agent actually went through are updated at the end of the episode.

If one of the node's actions were replaced with the null action, then the gridworld agent would not progress beyond the node's state as shown in Figure 3. In this example, the node associated with the third state entered has its actions changed from the move-right action to the null action, corresponding to its component in the action vector $z - z_i$. Whereas the reward summation for $G(z)$ sums eight rewards, the reward summation for $G(z-z_i)$ will include only the first two rewards, since the rewards received after the second time step have a value of zero. In a sense $G(z-z_i)$ gives the quality of the path up to node i. Therefore difference of these two utilities $G(z) - G(z-z_i)$ gives the quality of the path past node i. Since anything that

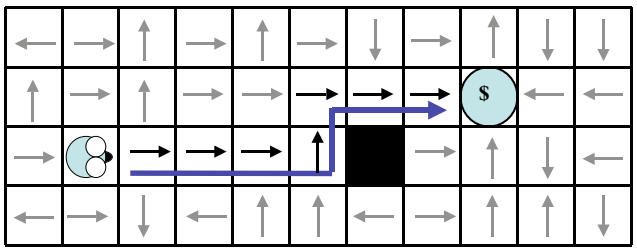

Figure 2. Classic Gridworld Problem broken down into multiple nodes. Each grid square is assigned a node, which chooses an action at the beginning of each episode. The agent then follows these actions. Nodes that were visited by the agent are updated (black arrows).

happened before the agent entered node i does not affect the value of $G(z)-G(z-z_i)$, that utility can be interpreted as the contribution of node i to the full path.

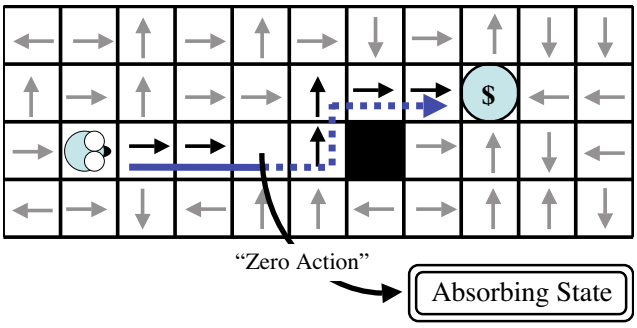

Figure 3. When z is changed to $z - z_i$ the action of node i (third square right of agent's starting place) is changed from the move-right action to the "null" action. With this change, the gridworld agent no longer moves along the dotted line and instead moves into an absorbing state, receiving rewards of zero.

5. DU and Monte Carlo Estimation

When all of the rewards of an episode of N time steps are known when the private utility for a node is compute, the DU (Equation 1) can be used as the node's private utility:

$$DU_i(z) = G(z) - G(z - z_i) \quad (8)$$

$$= \sum_{t=1}^{N} R_t(z) - \sum_{t=1}^{N} R_t(z - z_i) \quad (9)$$

$$= \sum_{t=1}^{N} R_t(z) - \sum_{t=1}^{T(i)-1} R_t(z - z_i) - \sum_{t=T(i)}^{N} R_t(z - z_i)\ , \quad (10)$$

where $T(i)$ is the first time the agent entered state $S(i)$. For times before $T(i)$, the action taken by node i is irrelevant so $R_t(z - z_{T(i)})$ equals $R_t(z)$ for all $t < T(i)$. Therefore we can rewrite DU as:

$$DU_i(z) = \sum_{t=1}^{N} R_t(z) - \sum_{t=1}^{T(i)-1} R_t(z) \quad (11)$$

$$- \sum_{t=T(i)}^{N} R_t(z - z_{T(i)}) \quad (12)$$

$$= \sum_{t=T(i)}^{N} \left(R_t(z) - R_t(z - z_i)\right)\ . \quad (13)$$

In addition all rewards $R_t(z - z_{T(i)})$ past time $T(i)$ are zero because the action at time $T(i)$ is now the null-action. Therefore we can simply write DU as:

$$DU_i(z) = \sum_{t=T(i)}^{N} R_t(z)\ . \quad (14)$$

The difference utility for node i's action is therefore the same as the undiscounted Monte Carlo estimation that would be received in a single-agent system for taking an action in state $S(i)$.

6. TDU and Immediate Rewards

Using the difference utility in this problem requires a node to know all of the future actions of an episode. This is an issue with Monte Carlo estimation, where an episode has to be completed before learning is performed. However in many reinforcement learning domains we want learning to be performed immediately, even before the future actions are taken. When the future actions of an episode are unknown, we can use the TDU instead of the DU by including only the current and previous actions in the observable components of z. The TDU for node i can be computed as follows:

$$TDU_i(z) = DU_i(z^{o_i}) \quad (15)$$

$$= \sum_{t=T(i)}^{N} R_t(z^{o_i}) \quad (16)$$

$$= R_{T(i)}(z^{o_i}) + \sum_{t=T(i)+1}^{N} R_t(z^{o_i}) \quad (17)$$

Since the future actions are unknown, all actions past time $T(i)$ are not observable. Therefore the actions in the z^{o_i} corresponding to actions that occur past time $T(i)$ are null-actions from the definition of z^{o_i}. Rewards that are a function of z^{o_i} therefore have a value of zero past time $T(i)$. The TDU therefore simply reduces to the immediate reward:

$$TDU_i(z) = R_{T(i)}(z^{o_i}) \tag{18}$$

This utility is unsatisfactory since to properly evaluate the quality of an action, a node needs to see the consequences of that action on future rewards. It is equivalent to Monte Carlo estimation with infinite discounting ($\gamma = 0$), which is clearly unacceptable in most domains. Note that the unacceptability of this utility is not obvious from its definition in Section 2. In fact even though this utility is not factored, it seemed promising due to its high learnability. The use of non-factored utilities is a potential problem in many multi-agent system applications. Unfortunately the downside of a non-factored utility is not as obvious as the downside of infinite discounting. The results in this section shows that they are equivalent, and that non-factored utilities have to be carefully evaluated in multi-agent domains, since the complexity of the multi-agent system may hide their danger.

7. EDU and Q-learning

Instead of the TDU, the EDU can be used as the utility for node i when the future actions in an episode are unknown. In this system the EDU is computed as follows:

$$\begin{aligned} EDUi(z) &= E[DU_i(z)|z^{o_i}] \\ &= \sum_{t=T(i)}^{N} E[R_t(z)|z^{o_i}] \\ &= E[R_{T(i)}(z)|z^{o_i}] + E[\sum_{t=T(i)+1}^{N} R_t(z)|z^{o_i}] \, . \end{aligned}$$

Since the reward at time $T(i)$ is completely determined from the observable components, $E[R_{T(i)}(z)|z^{o_i}]$ equals $R_{T(i)}(z)$, leaving us with the problem of estimating the sum of future rewards for an action. This estimate can be made by keeping a record of all of the rewards received for all the episodes. Using the rewards from previous episodes, a node, i, can estimate $\sum_{t=T(i)+1}^{N} R_t(z)$ by looking at the sequence of rewards received the last time an agent went through state S_{T_i+1}. However recording all of these rewards is unnecessary since the relevant rewards are summarized in the Q-tables of other nodes. Assuming the state entered after taking an action at time $T(i)$ is known, this estimate of future rewards can be obtained from Q-table values of the node used in the next state, and we can write EDU as follows:

$$EDUi(z) = R_{T(i)}(z^o) + E[Q_{i'}(z_{T(i)+1})|z^{o_i}] \tag{19}$$

where $z_{T(i)+1}$ is the next action after time $T(i)$ and i' is the node corresponding to the state entered at time $T(i)+1$. Since $z_{T(i)+1}$ is not observable, the estimation of $Q_{i'}(z_{T(i)+1})$ will depend on the exploration method used for each node. However for most exploration methods, as the rate of exploration approaches zero, the action correspond to the highest Q-value will always be used, resulting in the following computation of EDU:

$$EDUi(z) = R_{T(i)}(z^o) + \max_a Q_{i'}(a) \, , \tag{20}$$

where a is a possible action from state $S(i')$. In this situation the EDU for node i therefore provides the Q-learning estimate for action z_i in state $s(i)$. Note that we started out trying to make the "on-policy" estimate of what the sum of future rewards actually will be given the actions taken. However, since none of the future actions were known we ended up with the "off-policy" Q-learning estimate instead. The on-policy Sarsa type estimate cannot be used in this situation because the action taken after time $T(i)$ is not observable.

8. Discussion

This paper unifies the structural credit assignment problem present in single-time-step multi-agent systems and the temporal credit assignment present in time-extended single-agent systems. It does this by showing the relationship between the three utilities, DU, EDU and TDU to the three different reinforcement learning methods, Monte Carlo estimation, Q-learning and immediate reward learning, respectively. In each case the relation between the utility and the reinforcement learner is made through a specific construction of a multi-node system. The structural credit assignment view of temporal credit assignment highlights the salient properties of these methods along with their deficiencies. This view shows how the use of non-factored utilities commonly used in multi-agent systems is equivalent to a multi-time-step utility generally not considered in that field. In its place this paper shows how factored and close to factored utilities relate to successful reinforcement learning methods.

This paper has discussed only single-time-step multi-agent problems and multi-time-step single-agent

problems. The multi-time-step multi-agent problem is more difficult, since both temporal and structural credit assignment problems have to be dealt with at once. However using the method described in Section 4 it is possible to view a multi-time-step multi-agent problem as only a structural credit assignment problem.

For example take the gridworld problem shown in Figure 2. A multi-agent version of this problem can be defined by allowing many agents to move on the same grid. The reward for a time-step in the multi-agent gridworld would then be a function of all the grid-squares the agents occupy at that time step. This can be turned into a multi-node single-time-step problem by assigning a node to each agent/grid-square pair. So if there are n grid-squares and m agents there would be nm nodes in the multi-node formulation. The difference utility can then be derived for each node in the same way it is done in Section 5, up to equation 13:

$$DU_i(z) = \sum_{t=T(i)}^{N} \left(R_t(z) - R_t(z - z_{T(i)}) \right) .$$

The key difference in the multi-agent version is how we define the effects of the null-action for an agent. In Section 4 it was defined so that the rewards $R_t(z - z_{T(i)})$ had a value of zero past time $T(i)$. This made sense in a single-agent system where we are summing rewards for a single agent. In the multi-agent version we have a double summation over time and agents $(\sum_i \sum_t R_{i,t})$ and this sum will not be zero if a single agent is taken out. Instead any reduction in the term $R_t(z - z_{T(i)})$ will be system dependent. Our current research focuses on exploiting this coupling to allow faster convergence and better performance in multi-time-step multi-agent systems.

References

[1] A. Agogino and K. Tumer. Team formation and communication restrictions in collectives. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Melbourne, Australia, July 2003.

[2] R. H. Crites and A. G. Barto. Improving elevator performance using reinforcement learning. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems - 8*, pages 1017–1023. MIT Press, 1996.

[3] 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.

[4] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 157–163, 1994.

[5] Maja J Mataric. Reward functions for accelerated learning. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 181–189, San Francisco, CA, 1994.

[6] P. Stone. *Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer*. MIT Press, Cambridge, MA, 2000.

[7] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3), 2000.

[8] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

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

[10] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:33–53, 1992.

[11] 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.

[12] K. Tumer and D. Wolpert, editors. *Collectives and the Design of Complex Systems*. Springer, 2004. To appear.

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

[14] D. H. Wolpert, J. Sill, and K. Tumer. Reinforcement learning in distributed domains: Beyond team games. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 819–824, Seattle, WA, 2001.

[15] D. H. Wolpert, K. Tumer, and J. Frank. Using collective intelligence to route internet traffic. In *Advances in Neural Information Processing Systems - 11*, pages 952–958. MIT Press, 1999.

[16] D. H. Wolpert, K. Wheeler, and K. Tumer. Collective intelligence for control of distributed dynamical systems. *Europhysics Letters*, 49(6), March 2000.