Multiagent Learning with a Noisy Global Reward Signal

Scott Proper and **Kagan Tumer**
Oregon State University
Corvallis, OR 97331, USA
proper@eecs.oregonstate.edu
kagan.tumer@oregonstate.edu

Abstract

Scaling multiagent reinforcement learning to domains with many agents is a complex problem. In particular, multiagent credit assignment becomes a key issue as the system size increases. Some multiagent systems suffer from a global reward signal that is very noisy or difficult to analyze. This makes deriving a learnable local reward signal very difficult. Difference rewards (a particular instance of reward shaping) have been used to alleviate this concern, but they remain difficult to compute in many domains. In this paper we present an approach to modeling the global reward using function approximation that allows the quick computation of local rewards. We demonstrate how this model can result in significant improvements in behavior for three congestion problems: a multiagent "bar problem", a complex simulation of the United States airspace, and a generic air traffic domain. We show how the model of the global reward may be either learned on- or off-line using either linear functions or neural networks. For the bar problem, we show an increase in reward of nearly 200% over learning using the global reward directly. For the air traffic problem, we show a decrease in costs of 25% over learning using the global reward directly.

1 Introduction

Reinforcement learning (RL) in large multiagent systems is a wide area of research with applications ranging from robocup soccer (Stone, Sutton, and Kuhlmann 2005), to rover coordination (Agogino and Tumer 2008), to trading agents (Sherstov and Stone 2005; Wellman et al. 2003), to air traffic management (Tumer and Agogino 2007). The challenge of multiagent learning in complex environments is for each agent to extract a useful reward signal from the noise of other agents acting within the same environment. Agents must somehow learn to coordinate among themselves and develop a joint set of policies to solve the problem. Agents are usually learning simultaneously, further complicating the learning process as the behavior of the other agents is changing in unpredictable ways.

In some domains, it can be very difficult to analytically derive a local agent-specific reward signal from the global signal provided to the agents – the global reward may be a "black box", or the system dynamics may just be too complex. Under such conditions, it may still be possible to learn an approximate model of the global objective, and then use that model to derive a local reward function for the agents.

Given a model of the global objective function, we can create an agent-specific reward signal using difference rewards, which are a specific type of shaped reward that encourages good agent behavior by rewarding actions that are closely aligned with the desired overall system behavior, while still allowing agents to learn from the reinforcement signal. Difference rewards have been shown to perform very well in multiagent domains (Agogino and Tumer 2008), however they have previously suffered from one great disadvantage: it has not always been possible to calculate the value of the difference reward, or even an approximation of it, generally due to complex system dynamics. A modeling approach mitigates this difficulty in approximating the difference reward in complex domains.

The proposed modeling approach takes advantage of function approximation techniques to approximate the global reward signal, which we may then use to calculate an approximate difference reward. We use tabular linear functions (Proper and Tadepalli 2006) – consisting of a number of tables or neural networks – to model the value of the global (system) reward. This model may then be used to calculate the difference reward. We apply this technique to three multiagent congestion problems of varying complexity. The results show that we can greatly improve performance over learning on the system reward directly, and in some cases even outperform the true model of the reward signal.

In Section 2 we summarize the basic agent learning architecture. In Section 3 we show how we model the difference reward using the system reward and discuss the function approximation techniques we used in experiments. In Section 4 we discuss the congestion problems that we use in the reported experiments. In Section 5 we report the results. Finally, in Section 6 we discuss the results and provide directions for future research.

2 Multiagent Learning with Reward Shaping

The algorithm we use for our experiments is a simple reinforcement learner, essentially an implementation of stateless Q-learning (Sutton and Barto 1998). Each agent keeps a vector providing its estimates of the reward it would receive for taking each possible action. In any episode, an agent esti-

mates its expected reward for a given action based on action values it has developed in previous episodes. We use ϵ-greedy exploration so agents will explore with a random action with probability ϵ. Each agent receives reward R and updates the action value vector using a value function V_k:

$$V_k = (1 - \alpha) \cdot V_k + \alpha \cdot R \quad (1)$$

When providing an agent's reward signal, a reasonable option is to give each agent the **global reward** $G(z)$, where z is a vector of features describing a set of actions or state-actions pair. However, this reward is not particularly sensitive to an agent's actions and for large systems, leads to very slow learning. Previous work has shown that we can instead provide a *difference reward*, which can significantly outperform agents either receiving a purely local reward or all agents receiving the same global reward (Agogino and Tumer 2008; Tumer and Agogino 2007). The difference reward, a.k.a. D, is given by:

$$D_i(z) = G(z) - G(z - z_i) \quad (2)$$

where $z - z_i$ specifies the state of the system without agent i. In this instance z is the actions of the agents, and $z - z_i$ represents the actions of all the agents without agent i. Difference rewards are *aligned* with the system reward, in that any action that improves the difference reward will also improve the system reward. This is because the second term on the right hand side of Equation 2 does not depend on agent i's actions, meaning any impact agent i has on the difference reward is through the first term (G) (Tumer and Agogino 2007). Furthermore, it is more sensitive to the actions of agent i, reflected in the second term of D, which removes the effects of other agents (i.e., noise) from agent i's reward function.

Intuitively, this causes the second term of the difference reward function to evaluate the performance of the system without i, and therefore D measures the agent's contribution to the system reward directly.

3 Reward Modeling

For some simple domains, we may be given an equation for $G(z)$ from which it is possible to directly calculate a derivation of the difference reward using Equation 2. Unfortunately, many domains are not so simple – the global reward may result from a complex process that cannot be described by an equation, although some structure may be known about the reward signal. In the worst case, one simply has the vector z and some reward signal $G(z)$.

Our approach to solving this problem is to approximate D via a function approximator based on modeling $v(z) \approx G(z)$. We can then approximate $D_i(z) \approx v(z) - v(z - z_i)$. This solution, while conceptually simple, addresses a major criticism of difference rewards in the past: that they are difficult to use if an equation for $G(z)$ is not available. Thus, this work expands the possible uses of difference rewards into a large number of new domains. The key property that distinguishes this approach from standard function approximation is that the structural form of D has a built-in bias reducer. The subtraction operation ensures systematic errors in function approximation are eliminated, particularly if the two terms ($v(z)$ and $v(z - z_i)$) are close to one another. In this paper, we explore using two function approximators to model $G(z)$: tabular linear functions (TLFs) and neural networks. We discuss TLFs below.

3.1 Tabular Linear Functions

In previous work (Proper and Tadepalli 2006), tabular linear functions (TLFs) were shown to provide a simple, flexible framework to consider and incorporate different assumptions about the functional form of an approximated function and the set of relevant features. Previously, TLFs have been used to approximate the value function of an RL agent. In this work, we use it to approximate the reward model.

A TLF is a sum over several terms. Each term is given by multiplying a weight and feature value, just as with any linear function. Unlike standard linear functions, the weight of each term is given by an arbitrary function – typically a table – of other discretized (or "nominal") features.

More formally, a tabular linear function is represented by Equation 3, which is a sum of n terms. Each term is a product of a linear feature ϕ_i and a weight θ_i. The features ϕ_i need not be distinct from each other. Each weight θ_i is a function of m_i nominal features $f_{i,1}, \ldots, f_{i,m_i}$.

$$v(z) = \sum_{i=1}^{n} \theta_i(f_{i,1}(z), \ldots, f_{i,m_i}(z))\phi_i(z) \quad (3)$$

A TLF using tables to store the value of θ reduces to a linear function when there are no nominal features, i.e. when $\theta_1, \ldots, \theta_n$ are scalar values.

When using TLFs in a reinforcement learning algorithm, each θ_i is updated using the following equation:

$$\theta_i(f_{i,1}(z), \ldots, f_{i,m_i}(z)) \leftarrow \theta_i(f_{i,1}(z), \ldots, f_{i,m_i}(z)) + \alpha(E(z))\nabla_{\theta_i} v(z) \quad (4)$$

where $\nabla_{\theta_i} v(z) = \phi_i(z)$ and α is the learning rate. $E(z)$ is the error of a particular model value for the given features.

4 Congestion Problems

Congestion problems – where system performance depends on the number of agents taking a particular action – provide an interesting domain to study the behavior of cooperative multiagent systems. This type of problem is ubiquitous in routing domains (e.g., on a highway, a particular lane is not preferable to any other lane, what matters is how many others are using it) (Klügl, Bazzan, and Ossowski 2005; Tumer, Welch, and Agogino 2008). Agents in such a system often require sophisticated coordination in order to avoid a "tragedy of the commons" where selfish agents reduce the reward gained by everyone, including themselves (Hardin 1968). Two such congestion problems are discussed below.

4.1 Multi-night Bar Problem

The multi-night bar problem (shortened to "bar problem" in this paper) is an abstraction of congestion games (and a variant of the El Farol bar problem (Arthur 1994)) which has been extensively studied (Arthur 1994; Jefferies, Hart, and

Johnson 2002). In this version of the congestion problem, each agent has to determine which day of the week to attend a bar. The problem is set up so that if either too few agents attend (boring evening) or too many people attend (crowded evening), the total enjoyment of the attending agents drop.

The system performance is quantified by a system reward function G, which is a function of the joint action of all agents in the system z, and is given by:

$$G(z) = \sum_{day=1}^{n} x_{day} e^{\frac{-x_{day}}{C}} \quad (5)$$

where n is the number of actions (days), x_{day} is the total attendance on a particular day, and C is a real-valued parameter that represents the optimal capacity of bar.

Selfish behavior by the agents tends to lead the system to undesirable states. For example, if all agents predict an empty bar, they will all attend (poor reward) or if they all predict a crowded bar, none will attend (poor reward). This aspect of the bar problem is what makes this a "congestion game" and an abstract model of many real world problems, ranging from lane selection in traffic, job scheduling across servers, or data routing.

For this domain, we are fortunate that we can directly calculate the difference reward for the bar problem from Equation 5. In this domain, the only effect each agent has on the system is to increase the attendance, x_{day}, for night k by 1. This leads to the following difference reward:

$$\begin{aligned} D_i(z) &= G(z) - G(z - z_i) \\ &= x_{day_i} e^{\frac{-x_{day_i}}{C}} - (x_{day_i} - 1) e^{\frac{-(x_{day_i}-1)}{C}} \quad (6) \end{aligned}$$

where x_{day_i} is the total attendance on the day selected by agent i.

We can also apply TLFs to *learn* the model for the bar problem, rather than directly calculating it. The true model is given by Equation 5. The form of the TLF we define for this problem is:

$$F(z) = \sum_{day=1}^{n} \theta(x_{day}) \quad (7)$$

where $\theta(\cdot)$ is a table over possible nightly attendances, and is updated n times (once for each x_{day}) per episode as per Equation 4: $\theta(x_{day}) \leftarrow \theta(x_{day}) + \alpha(G(z) - F(z))$. Equation 7 thus allows an approximation of $G(z)$ to quickly be found given relatively few training examples.

4.2 Air Traffic Simulation

A second congestion problem explored in this paper is an air traffic simulation of the United States national airspace (NAS). Reinforcement learning and reward modeling has been applied to air traffic congestion previously (Tumer and Agogino 2009), however this past work only used a hand-coded reward model created by the experimenters, rather than a learned model of the reward as in this paper.

There has been significant research into agent-based methods for automating air traffic systems (G. Jonker 2007; Hill et al. 2005). These solutions typically involve a set of autonomous agents that try to optimize some overall goal either through learning or through negotiation. Agent interactions, inspired by economic principles, have been shown to achieve fairness in air traffic management through an artificial monetary system that allows for retaliation against greedy agents (G. Jonker 2007).

One key problem that needs to be addressed with learning agents is how to derive reward functions for each agent so that agents do not learn to hinder each other. In other contexts this has been addressed through a "satisficing" reward that specifically encourages cooperation and penalizes anti-cooperative behavior (Hill et al. 2005), and difference rewards where the actions of the agents aim to improve the system-wide performance criteria (Tumer and Agogino 2007). To date, the most complete path planning, control and collision avoidance based on agent technology is AgentFly which achieves reliable air traffic flow control without centralized planning (Sislak, Samek, and Pechoucek 2008).

One problem that all approaches to managing air traffic have is that it is impractical to experiment outside of simulation. Thus, we implement a simulator to test methods of air traffic control. Existing simulators are unsuitable for our purpose, being either slow, expensive, closed-source, or having other difficulties with obtaining or using it for machine learning experiments. We have chosen to implement a more suitable simulator, which we discuss below.

FEATS Simulator We developed FEATS (Fast Event-based Air Traffic Simulator) to quickly simulate thousands of aircraft of different characteristics taking off from airports, navigating via waypoints and airways to their destination airport, and landing. This simulator is optimized for speed, simulating 26,000 flights/second. Individual simulations require a fraction of a second, allowing efficient experimentation with machine learning techniques.

Aircraft are routed via A* search over sequences of fixes (2D locations in space) towards their destinations. Some of these fixes are "meter fixes", or locations at which aircraft may be slowed down as they approach, analogous to traffic lights in a traffic control simulation (Bazzan 2005). As in (Tumer and Agogino 2007) we choose to make meter fixes, rather than aircraft, into learning agents. We manage traffic by controlling aircraft separation distances – called "Miles in Trail" (MIT) separations – at meter fixes surrounding ten of the nations busiest airports (see Table 1). Each airport has a capacity of aircraft it can accept over any given 15-minute window. By increasing MIT separation at the meter fixes

Airport	Agents	Cap	Flights	Airport	Agents	Cap	Flights
ATL	6	20	300	JFK	5	11	165
CVG	6	18	270	LAX	13	21	315
DEN	8	30	450	MEM	5	20	300
DFW	10	30	450	MIA	7	17	255
IAH	8	18	27	ORD	9	20	300

Table 1: Airports, number of agents per airport, airport capacity, and number of flights assigned to each airport for each episode of the air traffic simulation.

Monitored	Total airports	Agents	Fixes	Edges	Flights	Km per side	Area (sq. km)	Time (seconds)
10	100	129	1000	11353	3585	3000	9,000,000	0.13
20	200	232	2000	23001	7125	4242	18,000,000	0.37
40	400	395	4000	35683	14295	6000	36,000,000	0.94

Table 2: Monitored airports (airports around which agents were placed), total airports, number of agents (metered fixes) in simulation, number of fixes, number of edges in the airspace graph, number of flights per simulation, kilometers per side of simulated airspace, area of simulated airspace, and time required per simulation for each simulated airspace domain.

surrounding an airport, air traffic coming into that airport can be delayed and spikes or "rushes" in air traffic can be moderated, allowing aircraft to land safely. However, doing this may cause costly delays so this is a tactic that should be used to the minimum amount required to allow all planes to land safely. This balancing of airport congestion and aircraft delays creates a complex multiagent control problem.

Each agent (meter fix) has four actions: to delay either 0, 10, 20, or 30 nautical miles in trail. A "delay" of 0 indicates that aircraft are permitted to fly right through the fix with no delay. Larger values require aircraft to slow down if they must do so to follow a proceeding aircraft by a sufficient margin. For this paper, each episode is a single simulated "rush" of aircraft from all over the national airspace to the various airports modeled in our simulation. We modeled 10 airports, 77 agents (meter fixes), and 3075 separate aircraft per simulation. The number of aircraft was scaled according to the capacity of each airport (see Table 1). In order to test air traffic control at even larger scales, we also created a "generic" air traffic domain artificially generated using experimenter-supplied parameters (Table 2).

We used a linear combination of terms for measured congestion and delay to calculate the global (system) reward of the air traffic domain as follows:

$$G(z) = -(B(z) + \alpha C(z)), \tag{8}$$

where $B(z)$ is the delay penalty for all aircraft in the system, and $C(z)$ is the total congestion penalty. The relative importance of these two penalties is determined by the value of α, a congestion cost, which we set to 5 (providing an effective tradeoff between congestion and delay). $B(z)$ is calculated by simply taking the sum of minutes of delay suffered by all aircraft at the meter fixes. $C(z)$ is given by:

$$C(z) = \sum_{p \in P} \int_T \Theta(k_{p,t} - c_p)(k_{p,t} - c_p)^2 dt, \tag{9}$$

where P is the set of airports monitored by the simulation, $k_{p,t}$ is the number of aircraft that have landed in the past 15 minutes (a rolling time window), c_p is the capacity of airport p as defined by the FAA, and $\Theta(\cdot)$ is an indicator function that equals 1 when its argument is greater or equal to zero, and has a value of zero otherwise. Thus $C(z)$ penalizes states where airports become over-capacity. The quadratic penalty provides strong feedback to return the airport to FAA mandated capacities. We use an integral over time due to the fact that our simulation occurs in real time.

4.3 Neural Network Approximation

The true difference reward for the air traffic problem is impossible to calculate, as we do not know how to analytically determine what the effects on $B(z)$ and $C(z)$ are of removing an agent from the system. However, we can approximate $D_i(z)$ using neural networks. All neural networks were randomly initialized and learned via backpropogation. Inputs were scaled between 0 and 1; outputs were scaled between 0 and 1 using the cumulative distribution function.

For the air traffic problem, we learned a model of $G(z)$ by decomposing it into a linear function of nonlinear terms, each of which we learned separately using a neural network, i.e. a TLF using several neural networks instead of tables:

$$v(z) = \sum_{p \in P} (\theta^p_B(z_p) + \theta^p_C(z_p)) \tag{10}$$

where z_p are the actions for the agents surrounding airport p, $\theta^p_B(\cdot)$ is a neural network approximating $B_p(z) = \sum_{a \in A_p} B_a(z)$, the sum of delays over all aircraft approaching p, and $\theta^p_C(\cdot)$ is a neural network approximating $C_p(z) = \int_T \Theta(k_{p,t} - c_p)(k_{p,t} - c_p)^2 dt$, the congestion penalty for a single airport. Each of the $2|P|$ networks has an input node for each action taken by the n_p agents (meter fixes) surrounding that airport, $n_p + 1$ hidden units, and 1 output, with a learning rate of 1.0. We train each network separately using $B_p(z)$ or $C_p(z)$ as appropriate for each airport p, allowing a more accurate approximation than training on only $G(z)$. Note that a meter fix may control incoming traffic to more than one airport. The action taken by such an agent is given as input to several of the neural networks.

Given this approximation of $G(z)$, we can now estimate $D_i(z) \approx v(z) - v(z - z_i)$, where we set $z - z_i$ to indicate that agent i takes a "default" action (in this case, setting its "Miles in Trail" value to zero, as imposing no delay on aircraft is a reasonable default action).

5 Experimental Results

We performed three sets of experiments with various combinations of reinforcement learning on local, global, and difference rewards for the bar problem and air traffic simulation. For each domain, we experimented with strategies for approximating $G(z)$ (and thus $D(z)$). 20 points were plotted for each result, each point the average of 200 (for the bar problem) or 50 (for the air traffic domain) episodes. We additionally averaged results over 30 runs for all experiments. Error bars are shown and were calculated using the sample standard error of the mean $\sigma/\sqrt{n}$ where n is the number of runs, however in most cases they are so small they disappear. Many experiments were conducted by learning the reward model "offline" (prior to training the agents) via randomly generated examples, as opposed to "online" (trained via examples observed by the agents during learning).

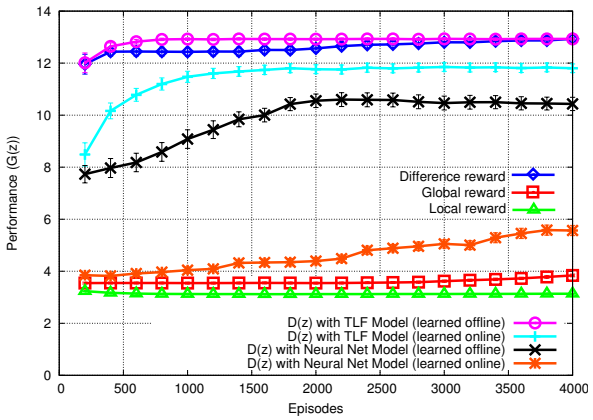

Figure 1: Comparison of rewards, function approximation techniques, and training method for the bar problem. Offline learning with TLFs worked very well, actually outperforming knowledge of the true model.

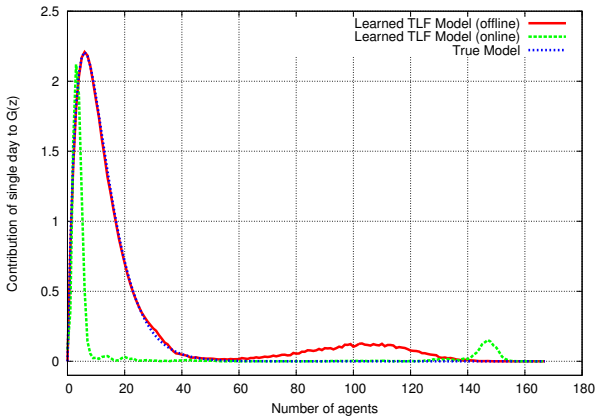

Figure 2: Comparison of the true and learned models for the bar problem. The model learned offline is very accurate for values up to about 50 agents, but has a long "hump" for larger values. The model learned online also shows two humps, one near the expected location of 6 agents and one farther along near the 144 agent position.

5.1 Bar Problem

The results of our experiments in the multi-night bar problem may be seen in Figure 1. We experimented with a week of 7 days, 168 agents, $C = 6$, $\alpha = .1$, and $\epsilon = .05$. Plots for local, global, and difference rewards are shown, along with several approximations of $D(z)$. Local reward was calculated by simply using $L_i(z) = x_{day_i} e^{\frac{-x_{day_i}}{C}}$. Global and difference rewards were calculated as seen in Equations 5 and 6, and approximations of D using Equation 7. For comparison purposes, we also use a simple neural network to approximate D, with 7 inputs (one for each day's attendance), 14 hidden units, and 1 output, with a learning rate of 1.5. In addition, we also experimented with learning the model both on- and off-line by training the TLF with 100,000 randomly generated examples of actions and their resulting global rewards. The TLF performed significantly better than neural network approximation, due to the fact that the form of the approximation (Equation 7) is very close to the true model (Equation 5). The neural network does not have the advantage of generalizing between days: each day is a separate input for which different weights must be learned. The additive decomposition of the TLF is also correctly biased: the true model also uses an additive decomposition (Equation 5). The neural network does not share this advantage.

As expected, training either the TLF or the neural network offline significantly improved performance. In fact, agents using a TLF converged faster than agents using the true model! The reason for this may be seen in Figure 2, which graphs the response of the model for attendance for a single day. The true model follows the expected curve defined by Equation 5. The TLF follows this curve very closely, until attendance grows over about 40 agents, at which point it forms a long second "hump". This hump makes the learned approximation a "shaped" approximation that encourages swift formation of a good policy by encouraging agents to group themselves into a single day.

5.2 Air Traffic Simulation

We performed experiments testing local, global, and difference rewards for FEATS as described in Section 4.2. Each episode simulated a single traffic "rush" from start to finish. The actions taken by the meter fixes controlled the delay each aircraft suffered as it was routed through that fix.

We used TLFs with neural networks approximating each term (Section 4.3) to estimate $G(z)$ and thus $D_i(z)$. We train each network offline using 10,000 randomly-generated examples. Samples were generated non-uniformly: a bias was introduced favoring lower-valued actions, which we expect to be more common in the true solution.

As may be seen in Figure 3, the estimated $D(z)$ significantly outperforms using either local or global rewards to learn. We also tested learning using the estimated $G(z)$ and found that performance was almost identical to learning with the true $G(z)$. Thus we believe that for purposes of learning,

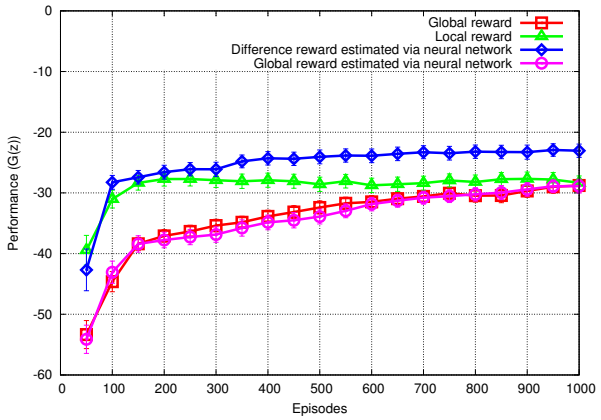

Figure 3: Comparison of rewards for the NAS simulation. The approximated difference reward outperforms other approaches, while the approximated global reward show that the neural network approximation used is very accurate.

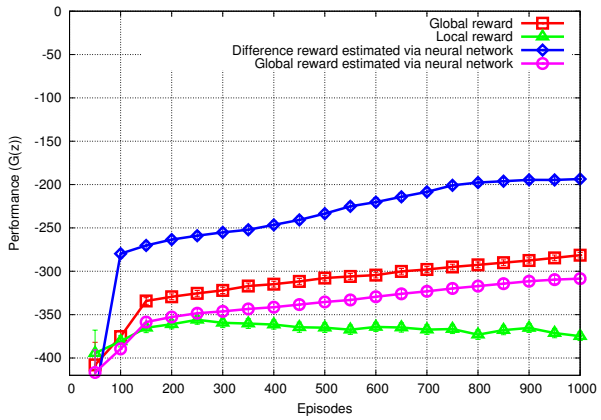

Figure 4: Results for 400 total airports and 395 agents in the generic air traffic domain. At the largest scales, global reward converges increasingly slowly, while difference reward continues to do well. This simulation is roughly four times the size of the true NAS.

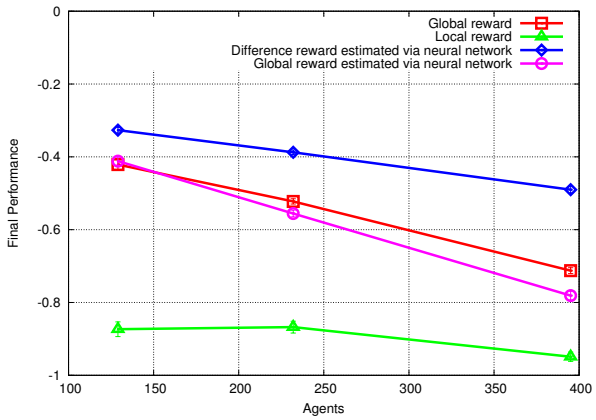

Figure 5: Comparison of scaling results for different numbers of agents (Table 2). The y-axis shows the average global performance obtained during the last 50 of 1000 time steps, divided by the number of agents. The estimated difference reward outperforms other methods at all scales, doing proportionately better as the number of agents increase.

the estimation of $G(z)$ (and thus $D(z)$) is very accurate.

Note that the graphs do not include the costs for offline training, as this training was not computationally significant compared to the time required to learn the policy, and was re-used for all 30 runs. Thus the costs for training are greatly amortized in this domain.

5.3 Generic Air Traffic Experiments

To test how well our methods scale to very large numbers of agents, we scaled up our experiments by using a set of generic air traffic domains created using the parameters described in Table 2. Each test was averaged over 30 runs, with a learning rate $\alpha = .1$, exploration rate $\epsilon = .1$, 1000 episodes, and the settings for various experiments as shown in Table 2. These experiments were otherwise similar to those performed in the previous section.

As may be see in Figures 4 and 5, the performance of the estimated difference reward outperforms any other method at all scales. As the scale of each test increases, the difference reward performs increasingly better in comparison to other methods due to the growing difficulty agents have with extracting a learnable reinforcement signal from an ever more noisy global reward. Local reward performs consistently poorly: it does not allow for coordination between agents, which is critical in this domain. The estimated global reward does well in comparison to the true global reward at all scales, but performance does degrade slightly at higher scales as the difficulty of the modeling problem increases.

The performances graphed in Figure 5 are divided by the number of agents at each scale, so this graph compares the final performance of individual agents. We can see that increasing the scale harms performance no matter the reward given, as would be expected due to the increasingly noisy and complex environment. However, difference rewards handle increases in scale far better than any other method, despite the fact that it is using a learned model rather than the "true" difference reward.

Notably, the time required to compute a single episode of a simulation using the difference rewards scales nearly linearly in the number of agents (Table 2). Any extra time required is due to the size of the simulation (extra edges and fixes requiring more calculation).

6 Discussion

We have shown that although calculating $D_i(z)$ for some multiagent domains may be impractical or impossible, it may still be possible to *estimate* $D_i(z)$ by learning a reward model of $G(z)$ using a variety of function approximators.

We found that a sufficiently accurate model of $G(z)$ does in fact allow us to estimate $D_i(z)$ well enough to obtain improved behavior over learning on either the local or global rewards. Further, as the number of agents increase, we show that using this model gives an increasing improvement over alternatives. We believe that for most applications, it will be necessary to learn the model offline in order to obtain sufficient data, however it remains a possibility of future work to show that learning a model online can eventually do nearly as well as having the true model. In the case of air traffic control, a vast database of states and actions already exists, or may be generated via sufficiently sophisticated simulations. This makes learning a model of the reward function offline a practical approach for many domains. Overhead of either approach was negligible compared to the cost of simulation.

We have also shown that in some cases, a learned model can converge faster than the true model. This is because an approximate model can shape the reward in such a way as to encourage faster convergence. In the case of the bar problem, the TLF learned to create a small "hump" in the model that encourages agents to densely populate a single day at the bar. Once this has happened, the normal difference reward signal takes over and convergence continues.

Future work includes continued experiments with model learning and the addition of states to our air traffic simulation, allowing agents to learn how to manage and route traffic by dynamically adapting to changing conditions.

Acknowledgements: This work was partially supported by the National Science Foundation under Grant No. CNS-0931591

References

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

Arthur, W. B. 1994. Complexity in economic theory: Inductive reasoning and bounded rationality. *The American Economic Review* 84(2):406–411.

Bazzan, A. L. 2005. A distributed approach for coordination of traffic signal agents. *Autonomous Agents and Multi-Agent Systems* 10:131–164.

G. Jonker, J.-J. Ch. Meyer, F. D. 2007. Achieving cooperation among selfish agents in the air traffic management domain using signed money. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems*.

Hardin, G. 1968. The tragedy of the commons. *Science* 162:1243–1248.

Hill, J. C.; Johnson, F. R.; Archibald, J. K.; Frost, R. L.; and Stirling, W. C. 2005. A cooperative multi-agent approach to free flight. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, 1083–1090. New York, NY, USA: ACM Press.

Jefferies, P.; Hart, M. L.; and Johnson, N. F. 2002. Deterministic dynamics in the minority game. *Physical Review E* 65 (016105).

Klügl, F.; Bazzan, A.; and Ossowski, S., eds. 2005. *Applications of Agent Technology in Traffic and Transportation*. Springer.

Proper, S., and Tadepalli, P. 2006. Scaling model-based average-reward reinforcement learning for product delivery. In *ECML '06: Proceedings of the 17th European Conference on Machine Learning*, 735–742.

Sherstov, A., and Stone, P. 2005. Three automated stock-trading agents: A comparative study. In *Agent Mediated Electronic Commerce VI: Theories for and Engineering of Distributed Mechanisms and Systems (AMEC 2004), Lecture Notes in Artificial Intelligence*. Berlin: Springer Verlag. 173–187.

Sislak, D.; Samek, J.; and Pechoucek, M. 2008. Decentralized algorithms for collision avoidance in airspace. In *Proceedings of Seventh International Conference on Autonomous Agents and Multi-Agent Systems*, 543–550.

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

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

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

Tumer, K., and Agogino, A. K. 2009. Multiagent learning for black box system reward functions. *Advances in Complex Systems* 12:475–492.

Tumer, K.; Welch, Z. T.; and Agogino, A. 2008. 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*.

Wellman, M. P.; Cheng, S.-F.; Reeves, D. M.; and Lochne, K. M. 2003. Trading agents competing: Performance, progress, and market effectiveness. *IEEE Intelligent Systems* 18(6):48–53.