A Social Reinforcement Learning Agent

Charles Lee Isbell, Jr. Christian R. Shelton
Michael Kearns Satinder Singh Peter Stone
AT&T Labs
180 Park Avenue
Florham Park, NJ 07932-0971

In the Fifth International Conference on Autonomous Agents (Agents-2001)

Abstract

We report on our reinforcement learning work on Cobot, a software agent that resides in the well-known online chat community LambdaMOO. Our initial work on Cobot (Isbell et al., 2000) provided him with the ability to collect *social statistics* and report them to users in a reactive manner. Here we describe our application of reinforcement learning to allow Cobot to proactively take actions in this complex social environment, and adapt his behavior from multiple sources of human reward. After 5 months of training, Cobot has received 3171 reward and punishment events from 254 different LambdaMOO users, and has learned nontrivial preferences for a number of users. Cobot modifies his behavior based on his current state in an attempt to maximize reward. Here we describe LambdaMOO and the state and action spaces of Cobot, and report the statistical results of the learning experiment.

1 Introduction

While most applications of reinforcement learning (RL) to date have been to problems of control, game playing and optimization (Sutton and Barto, 1998), there has been a recent handful of applications to human-computer interaction. Such applications present a number of interesting challenges to RL methodology (such as data sparsity and inevitable violations of the Markov property). These previous studies focus on systems that encounter human users one at a time, such as spoken dialogue systems (Singh et al., 2000).

In this paper, we give a report on our on-going efforts to build an RL-based agent for a complex, open-ended, multi-user chat environment known as LambdaMOO. We describe LambdaMOO in requisite detail in the next section, but here it suffices to say that it is an online service-providing enhanced real-time chat, populated by a community of human users with rich and often enduring social relationships. Our long-term goal is to build a software agent that can learn to perform useful, interesting and entertaining actions in LambdaMOO on the basis of user feedback. While this is a deliberately ambitious and underspecified goal, we describe here our implementation, the empirical experiences of our agent so far, and some of the lessons we have learned about this challenging domain.

In previous work (Isbell et al., 2000), we developed a software agent named Cobot that resides in LambdaMOO, and interacts in various ways with human users. The primary functionality of Cobot was twofold. First, Cobot gathered "social statistics" (such as which users interacted with which others, how frequently, and in what ways), and provided summaries of these statistics as a service to users. Second, Cobot had rudimentary chatting abilities based on the application of information retrieval methods to large documents. The original Cobot was entirely *reactive*[1], in that he[2] never initiated interaction with human users, but would only respond to their queries or utterances. As we documented in our earlier paper, Cobot proved tremendously popular with LambdaMOO users, setting the stage for our current efforts.

In May of 2000, we modified Cobot to allow him to occasionally take certain verbal actions (such as proposing a topic for conversation, introducing two human users to each other, or engaging in certain word play routines that are common in LambdaMOO) under his own initiative. The hope is to build an agent that will eventually take unprompted actions that are meaningful, useful or amusing to users. Rather than hand-code complex rules specifying when each action is appropriate (rules that, in addition to being inaccurate, might quickly become stale due to shifting user tastes), we wanted Cobot to *learn the individual and communal preferences* of users. Thus, we provided a mechanism by which users can reward or punish Cobot for his behavior, and programmed Cobot to use RL algorithms to alter his behavioral policy on the basis of this feedback.

[1] We use the term *reactive* to mean "restricted to responding to human-invoked interaction", rather than the AI term meaning "non-deliberative".

[2] Characters in LambdaMOO all have a specified gender. Cobot's description, visible to all users, indicates that he is male.

The application of RL (or any machine learning methodology) to such an environment presents a number of interesting domain-specific challenges, including:

- **Choice of an appropriate state space.** To learn how to act in a social environment such as LambdaMOO, Cobot must represent the salient features. These should include social information such as which users are present, how experienced they are in LambdaMOO, how frequently they interact with one another, and so on.
- **Multiple reward sources.** Cobot lives in an environment with multiple, often conflicting sources of reward from different human users. How to integrate these sources in a reasonable and satisfying way is a nontrivial empirical question.
- **Inconsistency and drift of user rewards and desires.** Individual users may be inconsistent in the rewards they provide (even when they implicitly have a fixed set of preferences), and their preferences may change over time (for example, due to becoming bored or irritated with an action). Even when their rewards are consistent, there can be great temporal variation in their reward pattern.
- **Variability in user understanding.** There is great variation in users' understanding of Cobot's functionality, and the effects of their rewards and punishments.
- **Data sparsity.** Training data is scarce for many reasons, including user fickleness, and the need to prevent Cobot from generating too much spam in the environment.
- **Irreproducibility of experiments.** As LambdaMOO is a globally distributed community of human users, it is virtually impossible to replicate experiments taking place there.

This is only a sample of the many issues we have had to confront in our Cobot work. We do not have any simple answers to them (nor do we believe that simple answers exist), but here provide a case study of our choices and findings. Our primary findings are:

- **Inappropriateness of average reward.** We found that the average reward that Cobot has received over time, the standard measure of success for RL experiments, is an inadequate and perhaps even inappropriate metric of performance in the LambdaMOO domain. Reasons include that user preferences are not stationary, but drift as users become habituated or bored with Cobot's behavior; and the tendency for satisfied users to stop providing Cobot with any feedback, positive or negative. Despite the inadequacy of average reward, we are still able to establish several measures by which Cobot's RL succeeds, discussed below.
- **A small set of dedicated "parents".** While many users provided only a moderate or small amount of RL training (rewards and punishments) to Cobot, a handful of users did invest significant time in training him.
- **Some parents have strong opinions.** While many of the users that trained Cobot did not exhibit clear preferences for any of his actions over the others, some users clearly and consistently rewarded and punished particular actions over the others.
- **Cobot learns matching policies.** For those users who exhibited clear preferences through their rewards and punishments, Cobot successfully learned corresponding policies of behavior.
- **Cobot responds to his dedicated parents.** For those users who invested the most training time in Cobot, the observed distribution of his actions is significantly altered by their presence.
- **Some preferences depend on state.** Although some users for whom we have sufficient data seem to have preferences that do not depend upon the social state features we constructed for the RL, others do in fact appear to change their preferences depending upon prevailing social conditions.

The outline for the rest of the paper is as follows. In Section 2, we give brief background on LambdaMOO. In Section 3, we describe our earlier (non-RL) work on Cobot. Section 4 provides some brief background on RL. In Sections 5, 6 and 7 we describe our implementation of Cobot's RL action space, reward mechanisms and state features, respectively. Our primary findings are presented in Section 8, and Section 9 offers conclusions.

2 LambdaMOO

LambdaMOO, founded in 1990 by Pavel Curtis at Xerox PARC, is one of the oldest continuously operating MUDs, a class of online worlds with roots in text-based multiplayer role-playing games. MUDs (multi-user dungeons) differ from most chat and gaming systems in their use of a persistent representation of a virtual world, often created by the participants, who are represented as characters of their own choosing. The mechanisms of social interaction in MUDs are designed to reinforce the illusion that the user is present in the virtual space. LambdaMOO appears as a series of interconnected rooms (modeled as a mansion), populated by users and objects who may move from room to room. Each room provides a chat channel shared by just those users in the room (users can also communicate privately), and typically has an elaborate text description that imbues it with its own "look and feel." In addition to speech, users express themselves via a large collection of *verbs*, allowing a rich set of simulated actions, and the expression of emotional states, as in the following transcript:

(1) Buster is overwhelmed by all these paper deadlines.
(2) Buster begins to slowly tear his hair out, one strand at a time.
(3) HFh comforts Buster.
(4) HFh [to Buster]: Remember, the mighty oak was once a nut like you.
(5) Buster [to HFh]: Right, but his personal growth was assured. Thanks anyway, though.
(6) Buster feels better now.

Lines (1) and (2) are initiated by verb commands by user Buster, expressing his emotional state, while lines (3) and (4) are examples of verbs and speech acts, respectively, by HFh. Lines (5) and (6) are speech and verb acts by Buster. (In our transcripts the name of the user initiating an action always begins the description of that action or utterance.) Though there are many standard verbs, such as the use of the verb **comfort** in line (3) above, the variety is essentially unlimited, as players have the ability to create their own verbs.

The rooms and objects in LambdaMOO are created by users themselves, who devise descriptions, and control access by other users. Users can also create objects with methods (or *verbs*) that can be invoked by other players.[3] As of this writing, the database contains 118,154 objects, including 4836 active user accounts. LambdaMOO's long existence and the user-created nature of the environment combine to give it one of the strongest senses of virtual community in the on-line world. Many users have interacted extensively with each other over a period of years, and many are widely acknowledged for their contribution of interesting objects. LambdaMOO is an attractive environment for experiments in AI (Foner, 1997; Mauldin, 1994), including learning. The population is generally curious and technically savvy, and users are interested in automated objects meant to display some form of intelligence (often called "puppets").

3 Cobot

Cobot is a software agent that resides in LambdaMOO. Almost all of Cobot's computation and storage occurs as a client of the LambdaMOO server (that is, computation is done off-server). As an off-server agent, Cobot appears to be just another user. Like a human user, he connects using the telnet protocol, and from the point of view of the LambdaMOO server, he is a user with all the rights and responsibilities that this implies. Once actually connected to LambdaMOO, Cobot wanders into the Living Room, where he spends most of his time. The Living Room is a central public place, frequented by many regulars. It is also located next to the Linen Closet, where guests tend to appear, so it is also frequented by users new to LambdaMOO. There are several permanent objects in the Living Room, including a couch with various features and a cuckoo clock. The Living Room usually has between five and twenty users, and is perpetually busy. Over the period of a year there, Cobot has logged well over 2.5 million separate events (on average roughly one event every twelve or thirteen seconds).

In previous work, we implemented a variety of (non-learning) functionality on Cobot. This included gathering and reporting *social statistics*. Thus, Cobot logs actions taken by users in his presence, building statistics on who performs what actions, and on whom they use them. For example, Cobot logs which users converse with each other most frequently. Cobot can answer queries about these usage statistics, and describe the statistical similarities and differences between users. Cobot also provides other reactive services. In particular, Cobot has a rudimentary chatting ability based on the application of information retrieval methods to large documents, and so can engage in crude "conversation" with users. He can also search the web to answer specific questions posed to him. A more complete description of Cobot's general abilities, and a detailed discussion of his early experiences as a social agent in LambdaMOO, can be found in (Isbell et al., 2000; Eisenberg, 2000).

The focus of our current work is in making Cobot *proactive* — that is, to allow him to take actions under his own initiative — in a way that is useful, interesting, or pleasing to LambdaMOO users. Since it is impossible to program rules anticipating when any given action is appropriate in such a complex and dynamic environment, we have applied reinforcement learning to allow Cobot to learn how to act directly from user feedback. We emphasize that *all of the original reactive functionality of Cobot remained operative throughout the RL experiment* — Cobot's popularity and acceptance in LambdaMOO is largely due to this original functionality, and we felt it was most interesting, and perhaps necessary, to conduct the RL work in the context of his established purpose.

As in any application of reinforcement learning, we must choose the actions, reward function, and states with care, and in a way that is meaningful in the domain. We describe our implementation of each of these following a brief RL background section.

4 RL Background

In RL, problems of decision-making by agents interacting with uncertain environments are usually modeled as Markov decision processes (MDPs). In the MDP framework, at each time step the agent senses the state of the environment, and chooses and executes an action from the set of actions available to it in that state. The agent's action (and perhaps other uncontrolled external events) cause a stochastic change in the state of the environment. The agent receives a (possibly zero) scalar reward from the environment. The agent's goal is to choose actions so as to maximize the expected sum of rewards over some time horizon. An optimal policy is a mapping from states to actions that achieves the agent's goal.

Many RL algorithms have been developed for learning good approximations to an optimal policy from the agent's experience in its environment. At a high level, most algorithms use this experience to learn *value functions* (or *Q-values*) that map state-action pairs to the maximal expected

[3] Everything in LambdaMOO is an object, and every event is the invocation of a verb on some object, including speech. The LambdaMOO server maintains the database of objects, and executes verbs.

sum of reward that can be achieved starting from that state-action pair. The learned value function is used to choose actions stochastically, so that in each state, actions with higher value are chosen with higher probability. In addition, many RL algorithms use some form of *function approximation* (parametric representations of complex value functions) both to map state-action features to their values and to map states to distributions over actions (i.e., the policy). See (Sutton and Barto, 1998) for an extensive introduction to RL.

In the next sections, we describe the actions available to Cobot, our choice of state features, and how we dealt with multiple sources of reward. The particular RL algorithm we use is a variant of (Sutton et al., 1999)'s policy gradient algorithm and its details are beyond the scope of this paper (however, see (Shelton, 2000) for details). One aspect of our RL algorithm that is relevant to understanding our results is that we use a *linear function approximator* to store our policy. For the purposes of this paper, this means that for each state feature, we maintain a vector of real-valued *weights* indexed by the possible actions. A positive weight for some action means that the feature increases the probability of taking that action, while a negative weight decreases the probability. The weight's magnitude determines the strength of this contribution.

5 Cobot's RL Actions

To have any hope of learning to behave in a way interesting to LambdaMOO users, Cobot's actions must "make sense" to users, fit in with the social chat-based environment, and minimize the risk of irritating them. Conversation, word play, and emoting routines are among the most common activity in LambdaMOO, so we designed a set of 9 actions for reinforcement learning along these lines, as detailed in Table 1. Many of these actions extract an utterance from the recent conversation in the Living Room, or from a continually changing external source, such as the online version of the Boston Globe. Thus a single action may cause an infinite variety of behavior by Cobot. Cobot also has a special "null" action, where he does nothing, so he may learn to be quiet.

At set time intervals (only every few minutes on average, to minimize the risk of irritating users), Cobot selects an action to perform from this set according to a distribution determined by the Q-values in his current state (described below). Any rewards or punishments received from users up until the next RL action are attributed to the current action, and used to update Cobot's value functions. It is worth noting that from an internal perspective, Cobot has two different categories of action: those actions taken in a proactive manner as a result of the RL, and those actions taken reactively in response to a user's action towards Cobot (his original functionality). However, this distinction is not explicitly announced by Cobot. Some users are certainly aware of the distinction and can easily determine which actions fall into which category, but other users may occasionally reward or punish Cobot in response to a reactive action. Such "erroneous" rewards and punishments act as a source of noise in the training process.

6 The RL Reward Function

Cobot learns to behave directly from the feedback of LambdaMOO users, any of whom can reward or punish him. There are both *explicit* and *implicit* feedback mechanisms.

For the explicit mechanism, we implemented **reward** and **punish** verbs on Cobot that LambdaMOO users can invoke at any time. These verbs give a numerical (positive and negative, respectively) training signal to Cobot that is the basis of the RL. Again, any such reward or punishment is attributed as immediate feedback for the current state and the RL action most recently taken. (This feedback will, of course, be "backed up" to previous states and actions in accordance with the standard RL algorithms.)

There are several standard LambdaMOO verbs that are commonly used among human users to express, sometimes playfully, approval or disapproval. Examples of the former include the verb **hug**, and of the latter the verb **spank**. In the interest of allowing the RL process to integrate naturally with the LambdaMOO environment, we chose to accept a number of such verbs as implicit reward and punishment signals for Cobot. (Indeed, many LambdaMOO users had been invoking such verbs on Cobot in response to his reactive behavior, long before we implemented his RL component.) However, such implicit feedback is numerically weaker than the feedback generated by the explicit mechanisms.

One fundamental design choice is whether to learn a single value function from the feedback of the entire community, or to learn separate value functions for each user based on their feedback alone, and combine the value functions of those users present to determine how to act at each moment. We opted for the latter route for three primary reasons.

First, it was clear that for learning to have any hope of succeeding, there must be a representation of which users are present at any given moment — different users simply have different personalities and preferences. Because of the importance of user identity, we felt that representing which users are present as additional state features would throw away valuable domain information, since the RL would have to discover on its own the primacy of identity. Having separate reward functions for each user is thus a way of asserting the importance of user identity to the learning process.

Second, despite the extremely limited number of training examples available in this domain (empirically < 700 per *month*), learning must be quick and significant. Without a clear sense that their training has some impact on Cobot's behavior, users will quickly lose interest in providing rewards and punishments. A known challenge for RL is the "curse of dimensionality," which refers to the fact that the size of the state space increases exponentially with the number of state features. By avoiding the need to represent the presence or absence of roughly 250 users, we are able to maintain a relatively small state space and therefore speed

Null Action	Choose to remain silent for this time period.
Topic Starters (4)	Introduce a conversational topic. Cobot declares that he wants to discuss sports, that he wants to discuss politics, or he utters a sentence from either the sports section or political section of the Boston Globe, for a total of four distinct actions.
Roll Call (2)	Initiate a "roll call". Roll calls are a common word play routine in LambdaMOO. For example, a discussion may be taking place where someone declares that she is tired of Monica Lewinsky. Another user might then announce, "TIRED OF LEWINSKY ROLL CALL", and each user who feels the same will agree with the roll call. Cobot initiates a roll call by taking a recent utterance, and extracting either a single noun, or a verb phrase. These are treated as two separate RL actions.
Social Commentary	Make a comment describing the current social state of the Living Room, such as "It sure is quiet" or "Everyone here is friendly." These statements are based on Cobot's logged statistics from the recent Living Room activity. While there are several different such utterances possible, they are treated as a single action for RL purposes.
Social Introductions	Introduce two users who have not yet interacted with one another in front of Cobot. Again, this is determined by Cobot's social statistical database.

Table 1: **The 9 RL actions available to Cobot.**

up learning. A similar approach to speeding up learning by maintaining a focused state representation in complex, dynamic environments with limited training data was explored previously in (Stone and Veloso, 1999).

Third, we (correctly) anticipated the fact that certain users would provide an inordinate amount of training to Cobot, and we did not want the overall policy followed by Cobot to be dominated by the preferences of these individuals. By learning separate policies for each user, and then combining these policies among those users present, we can limit the impact any single user can have on Cobot's actions.

7 Cobot's RL State Features

The decision to maintain and learn separate value functions for each user means that we can maintain separate state spaces as well, in the hopes of simplifying states and thus speeding learning. Cobot can be viewed as running a large number of separate RL processes (one for each user) in parallel, with each process having a different state space (with some common elements). The state space for a user contains a number of features containing statistics about that particular user.

Since LambdaMOO is a social environment, and Cobot is learning to take social actions (roll calls, suggestions of conversational topic, user introductions) we felt that his state features should contain information allowing him to gauge social activity and relationships. Table 2 provides a description of the state features used for RL by Cobot for each user. Even though we have simplified the state space by partitioning by user, the state space for a single user remains sufficiently complex to preclude standard table-based representation of value functions. (In particular, each user's state space is effectively infinite, since there are real-valued state features.) Thus, linear function approximation is used for each user's policy. Cobot's RL actions are then chosen according to a mixture of the policies of the users present. We refer the reader to (Shelton, 2000) for more details on the method by which policies are learned and combined.

8 Experimental Procedure and Findings

The older, reactive version of Cobot has been present in LambdaMOO more or less continuously since September 1999. The RL version of Cobot that is the subject of this study was installed on May 10, 2000. Again, all of Cobot's original reactive functionality, as well as some further reactive features added later, was left intact for the duration of the RL experiment reported here. Cobot is a working system with real human users, and we wanted to perform the RL experiment in this context.

After implementing the RL action and state spaces, the explicit and implicit reward mechanisms, and the multiple-user learning algorithms described in preceding sections, but prior to publicly fielding RL Cobot in the Living Room, we tested the new software on a separate agent we maintain for such purposes. These tests were performed by the authors and a few friendly users of LambdaMOO in a private room. In addition to tuning the learning algorithm's parameters, we calibrated the frequency with which the agent said something of its own volition. Such calibration is extremely important in a social environment like LambdaMOO due to users' understandably low tolerance for spam (especially spam generated by software agents).

Upon launching the RL functionality publicly in the Living Room, Cobot logged all RL-related data (states visited, actions taken, rewards received from each user, parameters of the value functions, etc.) from May 10 until October 10, 2000. During this time, 63123 RL actions were taken (in addition, of course, to many more reactive non-RL actions), and 3171 reward and punishment events[4] were received from 254 different users. The findings we now summarize are based on these extensive logs.

Inappropriateness of average reward. The most standard and obvious sign of successful RL would be an increase in the average reward received by Cobot over time. Instead, as shown in Figure 1, the average cumulative reward received by Cobot actually goes *down*. However, rather than indicating that users are becoming more dissatisfied as Cobot learns, the decay in reward reveals some peculiarities of human feedback in such an open-ended environment, as we now discuss.

There are at least two difficulties with average cumulative reward in an environment of human users. The first is

[4] That is, actions which received at least one act of reward or punishment from some user.

Social Summary Vector	A vector of four numbers: the rate at which the user is producing events; the rate at which events are being produced that are directed at the user; the percentage of the other users present who are among this user's ten most frequently interacted-with users ("playmates"); and the percentage of the other users present for whom this user is among their top ten playmates.
Mood Vector	A vector measuring the recent use of eight groups of common verbs (for instance, one group includes the verbs *grin* and *smile*). Verbs were grouped according to how well their usage was correlated.
Rates Vector	A vector measuring the rate at which events are being produced by the users present, and also by Cobot.
Current Room	The room where Cobot currently resides.
Roll Call Vector	Indicates whether the currently saved roll call text has been used before by Cobot, whether someone has done a roll call since the last time Cobot did a roll call, and whether there has been a roll call since the last time Cobot grabbed new text.
Bias	Each user has one feature that is always "on"; that is, this bias is always set to a value of 1. Intuitively, it is the feature indicating the user's "presence."

Table 2: **State space of Cobot.** Each user has their own state space and value function; the table thus describes the state space maintained for a generic user.

that humans are fickle, and their tastes and preferences may drift over time. Indeed, our experiences as users, and with the original reactive functionality of Cobot, suggest that novelty is highly valued in LambdaMOO. Thus a feature that is popular and exciting to users when it is introduced may eventually become an irritant. (There are many examples of this phenomenon with various objects that have been introduced into LambdaMOO.) In RL terminology, we do not have a fixed, consistent reward function, and thus we are always learning a moving target. While difficult to quantify in such a complex environment, this phenomenon is sufficiently prevalent in LambdaMOO to cast serious doubts on the use of average cumulative reward as the primary measure of performance.

The second and related difficulty is that even when users do maintain relatively fixed preferences over time, they tend to give Cobot less feedback of either type (reward or punishment) as he manages to learn their preferences accurately. Simply put, once Cobot seems to be behaving as they wish, users feel no need to continually provide reward for his "correct" actions or to punish him for the occasional "mistake." This reward pattern is in contrast to typical RL applications, where there is an automated and indefatigable reward source. Strong empirical evidence for this second phenomenon is provided by two users we shall call User M and User S. As we shall establish shortly, these two users were among Cobot's most dedicated trainers, each had strong preferences for certain actions, and Cobot learned to strongly modify his behavior in their presence to match their preferences. Nevertheless, both users tended to provide less frequent feedback to Cobot as the experiment progressed, as shown in Figure 1.

We conclude that there are serious conceptual difficulties with the use of average cumulative reward in such a human-centric application of RL, and that alternative measures must be investigated, which we do below.

A small set of dedicated "parents." Among the 254 users who gave at least one reward or punishment event to Cobot, 218 gave 20 or fewer, while 15 gave 50 or more. Thus, we found that while many users exhibited a passing interest in training Cobot, there was a small group that was willing to invest nontrivial time and effort in teaching Cobot their preferences. Two of the most active users in the RL training were those we shall call User M and User S, with 594 and 69 rewards and punishments events given to Cobot, respectively.[5]

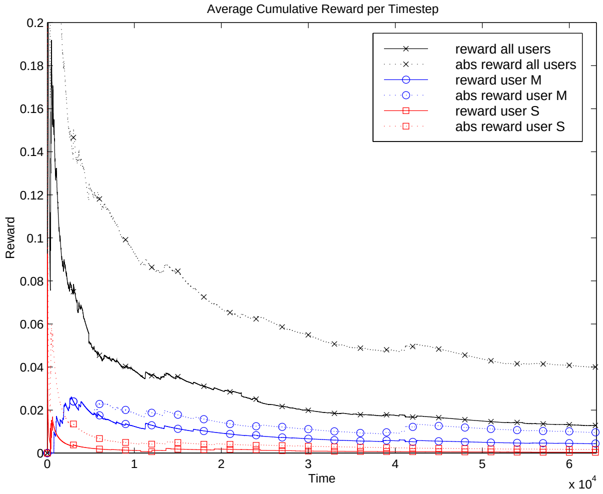

Figure 1: **The average reward received by Cobot over time.** The x-axis represents sequential Cobot-initiated RL actions for which it has received at least one reward or punishment event. The y-axis indicates the average cumulative reward received up to each such action. We examine both total reward (in which we sum the positive and negative values received), as well as absolute total reward (in which we ignore sign and simply count the total amount of feedback). We plot these quantities both over all users, and for two particulars users we call Users M and S. In all cases, as time progresses, Cobot is receiving less feedback. This occurs despite the fact that Cobot is learning policies that coincide with the preferences of Users M and S, suggesting that this may not be a useful measure of performance.

Some parents have strong opinions. For the vast majority of users who participated in the RL training of Cobot, the policy learned was quite close to the uniform distribution. Quantification of this statement is somewhat complex, since policies are dependent on state. However, we observed that for most users the learned policy's dependence on state was weak, and the resulting distribution near uniform (though there are interesting and notable exceptions, as we shall see below). This result is perhaps to be expected: most users

[5] By "reward event", we simply mean an RL action that received some feedback from the user. Note that the actual absolute numerical reward received may be larger or smaller than 1 at such time steps, since implicit rewards provide fractional amounts, and the user may also repeatedly reward or punish the action, with the feedback being summed. For example, the total absolute value of rewards and punishments provided by User M was 607.63 over 594 feedback events, while for User S it was 105.93 over 69 feedback events.

provided too little feedback for Cobot to detect strong preferences, and may not have been exhibiting strong and consistent preferences in the feedback they did provide.

However, there was again a small group of users for whom a highly non-uniform policy was learned. In particular, for Users M and S mentioned above, the resulting policies were relatively independent of state[6], and their entropies were 0.03 and 1.93, respectively. (The entropy of the uniform distribution over the actions is 2.2.) Several other users also exhibited less dramatic but still non-uniform distributions. User M seemed to have a strong preference for roll call actions, with the learned policy selecting these with probability 0.99, while User S preferred social commentary actions, with the learned policy giving them probability 0.38. (Each action in the uniform distribution is given weight 1/9 = 0.11.)

Cobot learns matching policies. In Figures 2 and 3, we demonstrate that the policies learned by Cobot for Users M and S[7] do in fact reflect the empirical pattern of rewards received over time. Thus, repeated feedback given to Cobot for a non-uniform set of preferences clearly pays off in a corresponding policy.

Cobot responds to his dedicated parents. The policies learned by Cobot for users can have strong impact on the empirical distribution of actions he actually ends up taking in LambdaMOO. In Figures 2 and 3, we examine how the distribution of actions taken by Cobot in the presence of these two users differs from the distribution of actions taken in their absence. In both cases, we find that the presence of the user causes a significant shift towards their preferences in the actions taken. In other words, Cobot does his best to "please" these dedicated trainers whenever they arrive in the Living Room, and returns to a more uniform policy upon their departure.

Some preferences depend on state. Finally, we establish that the policies learned by Cobot for users do sometimes depend upon the features Cobot maintains in his state. We use two facts about the RL weights (described in Section 4) maintained by Cobot to determine which features are relevant for a given user. First, we note that by construction, the RL weights learned for the bias feature described in Table 2 represent the user's preferences *independent* of state (since this feature is always on whenever the user is present). Second, we note that because we initialized all weights to 0, only features with non-zero weights will contribute to the policy that Cobot uses. Thus, we can determine that a feature is relevant for a user if that feature's weight vector is far from that user's bias feature weight vector, *and* from the all-zero vector.[8] For our purposes, we have used (1) the normalized inner product (the cosine of the angle between two vectors) as a measure of a feature's distance from the bias feature, and (2) a feature's weight vector length to determine if it is away from zero.

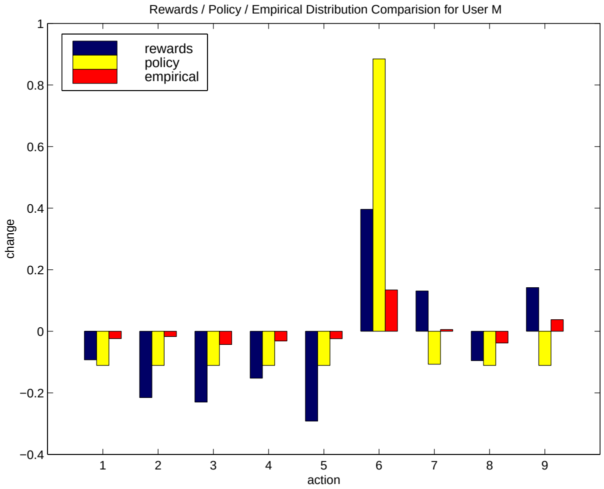

Figure 2: **Rewards received, policy learned, and effect on actions for User M.** For each of the nine RL actions, three quantities are represented. The blue bars (left) show the average reward given by User M for the corresponding action. The average reward given by User M across all actions has been subtracted off, so that "positive" bars indicate a relative preference for that action by User M, and "negative" bars indicate a relative objection to that action. The yellow bars (middle) show the policy (probability distribution) learned by Cobot for User M. The probability assigned to each action in the uniform distribution (1/9) has been subtracted off, so again the bars indicate relative preferences and objections of the learned policy. The red bars (right) show, for each action, the empirical frequency with which that action was taken by Cobot when User M was present, minus the empirical frequency with which that action was taken by Cobot over all time steps. Thus, these bars indicate the extent to which the presence of User M biases Cobot's behavior towards M's preferences. We see that (a) the policy learned by Cobot for User M aligns nicely with the preferences expressed by M through their training (comparison of left and middle bars), and (b) Cobot's behavior shifts strongly towards the learned policy for User M whenever they are present (comparison of middle and right bars). The actions are presented here in the same order as in Table 1. To go beyond a qualitative visual analysis, we have defined a metric that measures the extent to which two rankings of actions agree, while taking into account the fact that some actions are numerically extremely close in the each ranking. While we do not have the space to provide details here, the agreement between the action rankings given by the rewards of User M and the policy learned for User M are in near-perfect agreement by this measure, as are the rankings given by the policy and the empirical distribution of rewards.

As noted above, these measures show that for most users, Cobot learned a policy that is independent of state. For example, User M has a clear preference for roll calls, independent of state. As we consider the other users with non-uniform policies, however, a somewhat different story emerges. As we can see in Table 3, Cobot has learned a policy for other users that clearly depends upon state. Furthermore, personal observation of the users' behaviors by the authors provide anecdotal evidence that Cobot has learned relevant features that "make sense."

9 Conclusions and Future Work

We have reported on our efforts to apply reinforcement learning in a complex human online social environment. Lamb-

[6] We will discuss the methodology by which we determine the relative strength of dependence on state shortly.

[7] Again, we emphasize that since the policies for these users had only a very weak dependence on state, we can simply discuss a fixed policy.

[8] Technically, there is another possibility. All weights for a feature could be nearly the same, in which case the feature has no actual effect on which action is chosen, no matter how far away it is from zero. For the users we examined with non-uniform policies this has not turned out to be the case, so we do not discuss it further.

User O	$40°$	**Roll Call**. Two of the features from the Roll Call Vector have proven relevant for User O. User O appears to especially dislike roll call actions when there have been repeated roll calls and/or Cobot is repeating the same roll call again and again.
	$27 - 33°$	**Rates**. Two of the features from the Rates vector also seem to have a similar effect on the rewards given by User O, with the overall rate of events being generating having slightly more relevance than that of the rate of events being generated just by User O.
User B	$10°$	**Social Summary**. The presence of other users who generally direct events toward User B appears to have some relevance for his reward patterns. If we lower our threshold to consider features below 10 degrees we also notice that some of the other Social Summary features deviate from the bias by about 6 degrees. One might speculate that User B is more likely to ignore Cobot when he is with many friends.
User C	$34°$	**Roll Call**. User C appears to have strong preferences about Cobot's behavior when a "roll call party" is in progress (that is, everyone is generating funny roll calls one after the other), as suggested by the strong relevance of the Roll Call feature, indicating that other roll calls by other users have happened recently.
User P	$22°$	**Room**. Unlike most users, User P has followed Cobot to his home, where he is generally alone (but will continue to execute actions at an even higher rate than normal), and has trained him there. User P appears to have different preferences for Cobot under those circumstance than when Cobot is in the Living Room.

Table 3: **Relevant features for users with non-uniform policies.** Several of our top users had some features that deviated from their bias feature. The second column indicates the number of degrees between the weight vectors for those features and the weight vectors for the bias feature. We have only included features that deviated by more than 10 degrees. For the users above the double line, we have included only features whose weights had a length greater than 0.2. Each of these users had bias weights of length greater than 1. For those below the line, we have included only features with a length greater than 0.1 (these all had bias weights of length much less than 1).

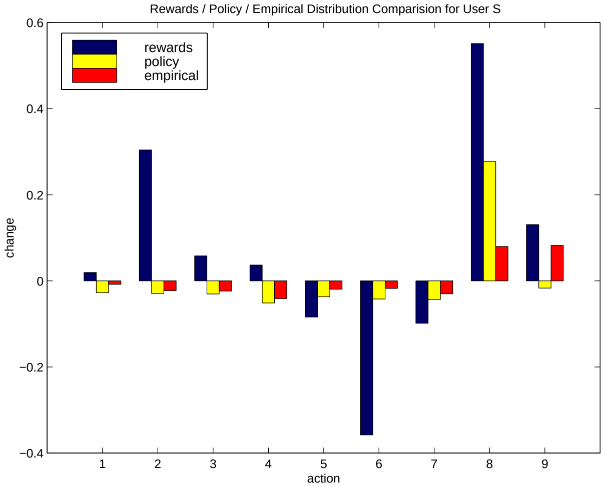

Figure 3: **Rewards received, policy learned, and effect on actions for User S.** Same as Figure 2, but for User S. While not as visually compelling as the data for User M, the quantitative agreement between the rankings is again very strong.

daMOO is a challenging domain for RL, as many of the standard assumptions (stationary rewards, Markovian behavior, appropriateness of average reward) are clearly violated. We feel that the results obtained with Cobot so far are preliminary but compelling, and offer promise for the application of RL in rather rich and open-ended social settings.

Cobot continues to take RL actions and receive rewards and punishments from LambdaMOO users, and we plan to continue and embellish this work as part of our overall efforts on Cobot. In addition to further analysis of the RL data as it comes in, we plan to expand both the state and action spaces of Cobot. Adding new state features is important to allow Cobot to learn more subtle lessons about which social contexts are appropriate for which actions, while adding new actions is important to keep Cobot fresh and interesting for the denizens of LambdaMOO.

References

Eisenberg, A. (2000). Find Me a File, Cache Me a Catch. New York Times, February 10, 2000. http://www.nytimes.com/library/tech/00/02/circuits/articles/10matc.html.

Foner, L. (1997). Entertaining Agents: a Sociological Case Study. In *Proceedings of the First International Conference on Autonomous Agents*.

Isbell, C. L., Kearns, M., Kormann, D., Singh, S., and Stone, P. (2000). Cobot in LambdaMOO: A Social Statistics Agent. *To appear in Proceedings of AAAI-2000*.

Mauldin, M. (1994). Chatterbots, TinyMUDs, and the Turing Test: Entering the Loebner Prize Competition. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*.

Shelton, C. R. (2000). Balancing Multiple Sources of Reward in Reinforcement Learning. *Submitted for publication in Neural Information Processing Systems-2000*.

Singh, S., Kearns, M., Littman, D., and Walker, M. (2000). Empirical Evaluation of a Reinforcement Learning Dialogue System. *To appear in Proceedings of AAAI-2000.*

Stone, P. and Veloso, M. (1999). Team partitioned, opaque transition reinforcement learning. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 206–212. ACM Press.

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

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *In Neural Information Processing Systems-1999*.