

User Modeling through Multi-Agent Markov Decision Processes

Preliminary Report

Peter Gorniak

December 2, 1999

Abstract

Current approaches to user modeling largely are specific to an application, do not generalize easily and have no theoretical foundation. Viewing the user as an agent acting in an environment and assuming that the agent can be modeled in term of standard methods from Artificial Intelligence is one way to address these issues. Here, the user is modeled as solving a Markov Decision Problem. This assumption provides a general representation of the user that can help with most user modeling tasks. This MDP is only partially observable as the internal state of the user is unknown, resulting in a hidden state problem. These hidden states can be disambiguated using the agent's history with an algorithm derived from the same problem in reinforcement learning. The algorithm works well in an example gridworld agent modeling problem, and has potential applications to the general partially observable reinforcement learning problem as well as the user modeling context discussed here.

1 Introduction

The homepage for this year's International Conference on User Modeling states that an application employs user modeling if it explicitly represents properties of a particular user. Such a representation then facilitates reasoning about the user and lets the application adapt. Examples of user modeling abound, because in some sense any program that keeps preferences on a per-user basis, such as an email client, engages in user modeling. More interesting are applications that do not require the user to provide the model, but rather derive such a model automatically.

Research concerning user modeling in this sense has usually focussed on a specific application within which the user's actions have a known meaning. While these approaches use well-known representations such as the Bayesian Networks in the Lumiere project (Horvitz *et al.*, 98), the Dynamic Bayesian Networks to predict a character's quest in a Multi User Dungeon game (Albrecht *et al.*, 1997), or Neural Networks and other learning algorithms to predict the user's text categorization behaviour (Gorniak, 1998a) (Gorniak, 1998b), these representations serve only to store and analyse data pertaining to one particular aspect. Even schemes that make claims about general applicability, such as an algorithm used for UNIX command prediction (Davison & Hirsh, 1998), are not based on a unifying representation.

In this paper I argue that modeling the user as an agent that is solving a Markov Decision Process has many properties of a unifying representation.

Specifically, such a model is in its basic version, as presented here, independent of the interpretation of states or actions within the application used. Despite that, it allows for prediction of future actions of the user and can be analysed to obtain some preferences of the user in using the application. The state space necessary for phrasing this problem in terms of an MDP can be built up during the actual modeling process. That is, while it is necessary that different application states can at least in some case be distinguished, neither the states nor the actions have to be known beforehand as long as they can be identified when they occur. This allows the MDP representation to apply to, say, an unknown application in which only the user interface can be examined. Specific application interpretations can then be added to the states and actions within this MDP and provide the power to reason about the application use in more contentful ways.

The MDP that the user is assumed to be solving can not be observed directly. All that can be observed is a sequence of user actions and partial states of the application environment the user is acting in. Neither the internal state of the user nor the full state of the application are usually known. Thus, the goal is to derive the MDP the user solves from observations of actions and environment states. This turns out to be a general hidden state identification problem for the case in which agents share an environment and wish to model each other. Here a technique adapted from McCallum's instance based identification method for reinforcement learning yields good results.

Section 2 discusses the implications of viewing the user as an agent solving an MDP and presents the state identification problem. Section 3 details the solution to the state identification problem, whereas section 4 shows the performance of this method and presents an example of the multi-agent modeling problem outside of the user modeling context. Finally, section 5 ends with a discussion of MDPs for user modeling.

2 The User as an Agent

The user is an agent acting in the environment of an application. For that, he or she must make decisions on how to act. Within modern artificial intelligence there is a standard paradigm to deal with deciding how to act in an environment: that of Markov Decision Processes. It seems only reasonable to apply this paradigm to the user modeling case. However, the problem does not consist of letting an intelligent agent make reasonable decisions. It consists of making an intelligent agent model another agent, namely the user. This task becomes better defined under the assumption that the user is an agent following that standard paradigm, that he or she is solving an MDP to select actions.

Examining the user modeling task under this assumption yields three state spaces, as depicted in figure 1. Firstly, the user has a state. That is, there is a goal the user is trying to accomplish and there may be any other number of things that are not part of the application state and influence the user's actions. Secondly, the environment or application has states. Such a state is a complete description of the program the user is interacting with in the sense that from it one can predict future states given actions, i.e. they must fulfill the Markov assumption. Finally, there are states made up of observations of states of the environment. Application states may only be partially observable (for example, the visible user interface controls that are observed may not completely encode the state of the application). I will call

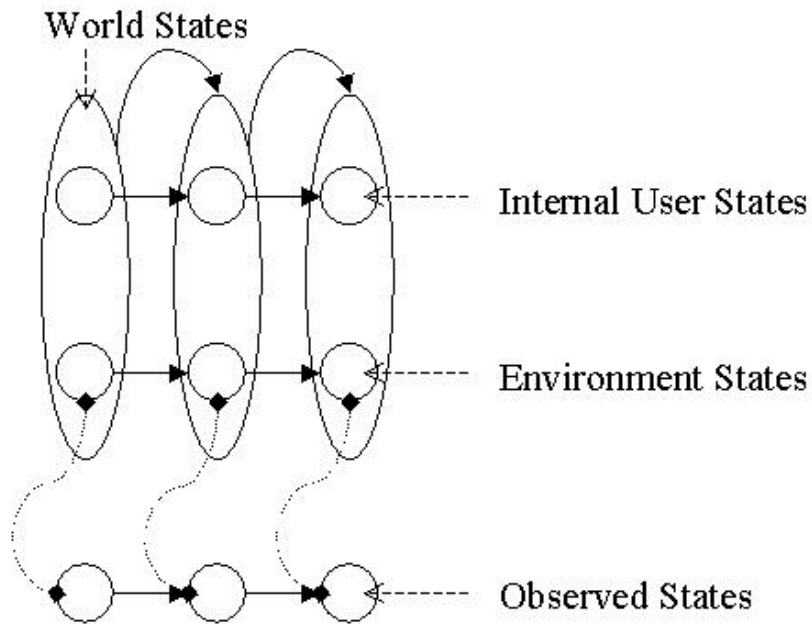


Figure 1: State Spaces for the User Modeling Problem

these state spaces in that order user states, states of the environment and observed states. There is a fourth kind of state, a world state, that is made up of a combination of a user state and an environment state. All of these kinds of states are depicted in figure 1.

The assumption made is that to act, the user solves an MDP with a state space made up of the combined internal user states and the environment state. At any given time the application is in a certain state and the user is in a certain state and from that information the user will select the next action. In fact, as an optimal policy in an MDP prescribes, the user will select the action with maximum expected reward, or maximum Q value. Let

us assume that this is true at least on average. This is not as strong an assumption as one may think. The optimality claim is made with respect to the user's specific MDP, not with respect to some 'optimal use of the application' criterion. Such a criterion can only be examined with more detailed application knowledge and is one of the possible uses of the model presented here.

The central problem of this method consists of the fact that we cannot directly gain access to the user's MDP. Even if the application state is completely observable, the world state consisting of the user's state and the application state certainly is not. So, we are faced with the task of approximating this state space. Once known, we can simply observe the frequencies of the user's actions from all states and must see that one action is strongly preferred - the optimal action. The prediction is then simply that the user will choose that action.

The approximation problem runs as follows: given a partially observed state of the environment and a history of user actions, approximate the world MDP state space from it. When do we know that the observed state does not correspond to a world state? Whenever the frequencies of user actions from that world state do not clearly indicate a strongly preferred action. In that case we know from our assumptions that there must truly be several world states for the one observed state. In fact, there must be at least one distinct world state for each high-frequency action from an observed state. In such a case, all we have access to to identify these hidden states is the history

of user actions and observed states. Section 3 introduces an algorithm that achieves this identification.

Note that history can disambiguate between a fair number of hidden states, but there are cases in which it fails. For example, at application startup time there exists no immediate history to perform hidden state identification with. Also, unobservable events can change the state without the model being updated. For example, the user may get a phone call and as a result suddenly select a different next action than his or her history might suggest. In a standard application such external influences will often exist and the resulting prediction errors are unavoidable. However, if a method is developed to monitor such events (for example, through a speakerphone on the computer), they can easily be incorporated as actions into the MDP model.

3 Hidden State Identification

The algorithm to identify hidden states in the agent modeling problem presented in this section is based on McCallum’s algorithm to do the same in a reinforcement learning context (McCallum, 1996). The main difference between that algorithm and the one presented here is that the former does not use an existing state space, but rather uses the recorded history as a state space. There may only be a few cases of hidden states in the problem presented here, and in general it seems like a lot of useful information is thrown away by discarding the existing state space completely. Instead, the algorithm presented here modifies the recorded frequencies, which loosely correspond to Q values for the observed states, according to how strongly recorded history supports them. In this way the actions from a state are ranked based upon their frequencies, which discards all infrequent ones, and those that are frequent in turn are ranked based upon their support from the user’s history.

The algorithm uses a simple k nearest neighbours scheme using matching sequence length as a metric to identify history support for an action. History is simply recorded as pairs of observed states and actions taken from those states. Instead of the complete history a moving window of historic information can be recorded so that state visits older than a certain threshold time are discarded. This speeds up the algorithm, but enough history must be kept such that states that can be disambiguated are disambiguated. For

each possible action a from the current state s , the algorithm performs the following calculations with some small integer k and some α with $0 \leq \alpha \leq 1$:

1. Compare the immediate history starting (s, a) and running backwards with all the recorded history. Find the k longest sequences in the complete recorded history that match the immediate history.
2. Average the length of these sequences and call it $l(s, a)$. This number encodes the support history presents for the considered action in the current state.
3. Count the number of times a was taken from s and call this $f(s, a)$.
4. Compute

$$B(s, a) = \alpha \frac{f(s, a)}{\sum_i f(s, a_i)} + (1 - \alpha) \frac{l(s, a)}{\sum_i l(s, a_i)}$$

where the sums run over all possible actions from s .

The claim is that $B(s, a)$ provides a ranking of actions that takes into account the hidden states contained in the current observed state s as revealed by the action choices the optimally acting agent, the user, made previously. The parameter α can be used to trade off between ranking actions according to their frequencies and ranking actions according to their support from history. Of course, frequencies are a type of support from history, but bascially no sequential information is encoded in them.

Note that $B(s, a)$ only provides a local and immediate ranking of actions. It uses proportions of frequencies and proportions of the total historic support

to rank actions and does not provide an encoding of expected future reward like a Q value. Section 4 evaluates this algorithm for a gridworld problem.

4 Results

The algorithm introduced in section 3 was developed for use in user modeling. However, it in fact solves a general multi-agent modeling problem where agents share a common environment and are trying to model each other's behaviour. This view makes a whole sleugh of standard MDP instances relevant that are easier to simulate, evaluate and understand than the user modeling case and still provide a good evaluation of the proposed algorithm. Here, a gridworld problem shall serve as a testing ground.

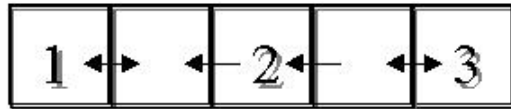


Figure 2: Gridworld Agent Modeling Problem

In this problem, an agent executes a fixed policy in a small gridworld consisting of only five states. Figure 2 shows this world and the three goals $\{1, 2, 3\}$ in it. At any given time the agent is trying to reach one of these goals, and once it reaches the goal will then pick a new goal. It picks the goals in the order 1, 2, 1, 3 and then cycles back to 1. The arrows show the maximum frequency counts for actions from states. For every state that has two leaving arrows it is impossible to predict the action the agent will pick from the frequency count alone. There are 12 actions in a cycle. Of those 12, we can only predict the actions from the leftmost and rightmost states, and the 'go-left' actions that occur in goalstate 2. As goalstates 1 and 2 are

visited twice in a cycle we can predict $5/12 = 42\%$ of actions.

Looking at the last state in addition to the previous state does not solve this problem completely, either. There is still now way to predict the 'go-right' action which occurs the second time the agent has entered goalstate 2 from the left.

However, the algorithm introduces in section 3 predicts all actions reliably. In fact, after a few initial wrong predictions the algorithm has captured the problem completely and can predict its complete future correctly.

5 Discussion

This paper focusses on viewing the user as an agent solving an MDP to act and approximating this MDP. Section 4 applies the resulting algorithm to a next-action prediction problem and very successfully so. The advantages of using the MDP framework are that

- it provides a well-known formalism to guide thinking and a sufficiently general representation to base one's reasoning on,
- it makes very few assumptions about the nature of the application modelled,
- more application knowledge can easily be phrased in term of the existing state spaces and the model presented here thus used as a basis for more detailed and informed modelling processes,
- the model captured in this way should provide a useful basis for a wide range of user modeling tasks,
- as the evaluation in section 4 shows, the user modeling problem is recast as a multi-agent modeling problem,
- this paper presents an algorithm that estimates the MDP the modelled agent is solving from the user's history and observations of the applications state.

There are two main routes that research can progress along using this model. On the user modeling side another evaluation of the algorithm in section 3 for an actual user modeling problem should be performed. Furthermore, the claims above should be substantiated by using this formalism for deriving other aspects of user models like the user's goals or preferences and applying the results to actual modifications of the application's behaviour. Prime candidates for such modifications are automation of tasks and interface modifications.

The other route consists of generalizing the algorithm used here to reinforcement learning in POMDPs. In the form presented here it is impossible to perform dynamic programming or similar approaches. While actions in a state are ranked according to how much history supports them, the state does not actually ever get split into the corresponding hidden states, and thus no Q values can be recorded for those states. However, there is no reason this could not be done, for example by attaching the shortest sequence of the k nearest neighbours found as an identifier to the created hidden state. This still preserves some structure of the original problem, yet can lead to an algorithm more along the lines of McCallum's (McCallum, 1996) that can be used in the reinforcement learning case instead of the agent modeling case, while still using the existing state model as a basis.

6 Conclusion

Viewing the user as an agent making decisions and acting in an environment suggests modeling this agent in terms of standard decision making paradigms. One such paradigm is that of solving Markov Decision Problems. When the user is viewed as solving an MDP when using an application, the user modeling problem is transformed into a multi-agent modeling problem where the agents share an environment. This paper presents an algorithm to approximate the state space the observed agent must be acting in from observations of that state space and a history of the modeled agent's actions. The algorithm succeeds in predicting the actions of an agent in a gridworld setting completely, even though the agent's goals are invisible to the observer.

This MDP modeling strategy neither makes assumptions about the nature of the application, nor does the algorithm presented here need to be able to interpret application states or user actions to capture the MDP and successfully predict the user's actions. This model can then provide a basis for goal prediction, preference elicitation and other common user modeling tasks.

This paper solves a multi-agent problem. However, the algorithm presented in section 3 actually performs rather general hidden state identification based upon history in an existing state space. With some modifications it should be possible to adapt this algorithm to the reinforcement learning case.

References

- Albrecht, David W., Zukerman, Ingrid, Nicholson, Ann E., & Bud, Ariel. 1997. Towards a Bayesian Model for Keyhole Plan Recognition in Large Domains. *In: User Modeling: Proceedings of the Sixth International Conference, UM97.*
- Davison, Brian D., & Hirsh, Haym. 1998. *Predicting Sequences of User Actions*. Tech. rept. Rutgers, The State University of New York.
- Gorniak, Peter J. 1998a. *MailMind - A Connectionist E-Mail Sorting Agent*. Honours Thesis.
- Gorniak, Peter J. 1998b. *Sorting Email Messages by Topic*. Course Project Report.
- Horvitz, Eric, Breese, Jack, Heckerman, David, Hovel, David, & Rommelse, Koos. 98. The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users. *In: Uncertainty in Artificial Intelligence, Proceeding of the Fourteenth Conference.*
- McCallum, Andrew R. 1996. *Instance-Based State Identification for Reinforcement Learning*. Tech. rept. University of Rochester.