

Modeling Use of Unmodified Applications from Observed Interactions

Peter Gorniak and David Poole
Department of Computer Science
University of British Columbia
Vancouver, British Columbia, Canada
pgorniak@cs.ubc.ca, poole@cs.ubc.ca

November 30, 1999

1 Problem Statement

Artificial Intelligence supplies a vast set of tools to be applied to the design of intelligent user interfaces. While our previous research (Gorniak, 1998) as well as countless other projects sample indulgently from this set and often produce quite impressive results in their own environments (for example, see (Horvitz *et al.*, 98) and (Albrecht *et al.*, 1997),) there emerge some new challenges when one attempts to apply these results to a new application.

1. The research results often do not transfer easily to a new application.
2. The actual implementation used in the research relies upon a modified application. This modification is usually non-trivial, time-consuming to repeat and increases application complexity.
3. Researchers work from various, often hand-crafted application models. In addition to the application building work in 2), an application designer needs to specify such a model. This process is often not straightforward and may rely on empirical data from user trials. An application designer's primary task does not include designing such a model, and thus the task seems an added difficulty to him or her. Also, the model needs to be updated and will tend to lag behind the application during maintenance.

We are currently addressing these problems by investigating how much knowledge can be extracted from a user's interaction with an application without any prior information about the application's purpose or structure and without any modifications to the application (somewhat in the spirit of (Lieberman, 1998).) We hypothesize that enough knowledge can be extracted to yield a detailed model of the application and its user. This model can then serve both as a knowledge source for other algorithms as well as provide a context under which to unite methods.

2 Approach

Let us view the user as an agent. Our assumption is that we can and have observed this agent acting in an environment, namely using an application. But 'agent acting in an environment' rings a bell. Much of Artificial Intelligence concerns itself with agents acting in environments and worries about what decisions such agents should make. A common approach to such a problem consists of phrasing it in terms of states and actions between states and coming up with a policy that, perhaps stochastically, dictates which actions to take in which states. We are faced with the opposite problem: we see an agent acting in an environment and want to model the agent's decision process. We assume that the agent acts according to a policy. Each action is the result of some (possibly stochastic) function of what the agent observes and the agent's belief state. Our goal is to determine this policy and the state space to which it applies.

A state for the user consists of a combination of the user's internal state and the application's interface state. At a given time, the user chooses an action from a probability distribution based upon the current state. We attempt to determine the policy the user is employing from the user's interaction history we observe. To do so, we hypothesize internal states of good predictive power in addition to our observations of interface states. By itself, the resulting state graph can be used to identify the user's current state, to predict future user actions and to gain a better understanding of application usage. In future we see the graph being augmented by grouping states to define application contexts, by assigning interpretations to the automatically derived states and by applying other approaches to better estimate the action probabilities in some states. Finally, other interface agents may find valuable information in this graph, such as future user actions which in an augmented graph lead to the identification of user context, goals and plans.

The approximation of state space and policy starts with a graph of observations of interface states as recorded from the application. Frequently, such observed interface states are very coarse in that many actions commonly occur in them, and they do not approximate the user's states nor the application's states well. Leaning on McCallum's algorithm for using history information for reinforcement learning (McCallum, 1996), we split states if the user's history yields sequences that strongly indicate preference for certain actions in those states. To do this, we choose a k nearest neighbours approach, where neighbours are matching segments of history and distance between them is measured by the length of the match. In this way states can be split implicitly and in real-time to estimate the state of the system and predict future states. Our algorithm collects k history sequences that match the user's immediate history together with a possible next action and averages their lengths to estimate history support for that action. Empirical evidence indicates better performance for on-line next action prediction than other algorithms for that purpose (see section 4.)

Lately, we have also investigated splitting states explicitly. In our initial design, a state splits if there are at least k sequences of minimum length l in history that match. The sequence of maximum length that matches at least k neighbours attaches to the new state to identify it. Performing such explicit splitting lets us draw graphs that model the application usage, such as the one shown in section 4.

Other application independent user models either build no application model at all, and thus do not provide any automatic analysis of the application and perform worse in cases where such application knowledge boosts performance, such as future action prediction (Davison & Hirsh, 1998); or, they stop early on in their analysis and subsequently rely on application specific knowledge (Encarnacao, 1997).

3 Implementation

Java's reflective capabilities make the language a prime candidate for an application independent approach. It allows to not only inspect a structure of known visual components, but it can also inspect unknown components for common state information (such as their visibility, their displayed values and whether they are enabled) and thus extract much valuable state information. The system used for the experiments presented below runs as a wrapper to a Java application. Before it starts the application, it hooks itself into the application's event queue and thus sees all event activity within the Java Abstract Window Toolkit and components derived from it. It intercepts each such event that it considers an action (such as a button being pressed or a window closed) and records the observed state of the application's interface before and after the event occurs. In this way, this system establishes a state space of interface observations as the application is used and records a history consisting of actions and visited states at the same time.

The applications under consideration here are educational AI applications. That is, they were written to help undergraduate university students learn concepts in Artificial Intelligence. The applications ran unmodified with our recording system in the background. The following discussion and results stem from an application that demonstrates common search algorithms.

4 Results

Figure 1 shows the explicit state graph derived for the search application. States starting with "Split" are states not given by the user interface. This graph captures the application structure well (for example, the horizontal split down the middle is a result of the application having two main modes.) Also, the

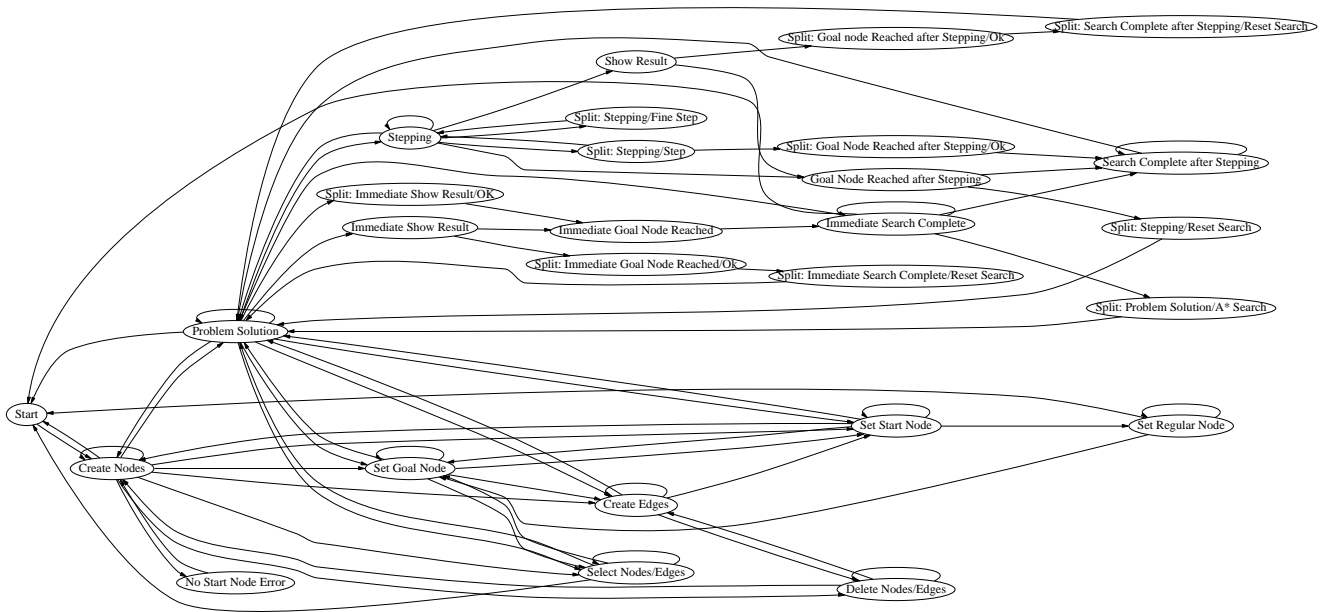


Figure 1: State Graph for Search Application

split states yield valuable information about the user, such as when he or she is likely to switch searching methods, something that the application interface does not dictate.

We compared the implicit version of our state space approximation to IPAM (Davison & Hirsh, 1998), a next action prediction algorithm. On the ~2200 actions recorded from 9 students the algorithm presented here correctly predicts ca. 80%, whereas IPAM predicts 70% percent considering only actions and 75% using state-action pairs. This gives a significant improvement in the macroaverage with better than 95% confidence. All predictions were based upon the complete data set and IPAM was run with $\alpha = 0.8$.

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.
- Encarnacao, L.M. 1997. *Concept and Realization of intelligent user support in interactive graphics applications.* Ph.D. thesis, Eberhard-Karls-Universität Tübingen, Fakultät für Informatik.
- Gorniak, Peter J. 1998. *Sorting Email Messages by Topic.* 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.*
- Lieberman, Henry. 1998. Integrating User Interface Agents with Conventional Applications. *In: Proceedings of the International Conference on Intelligent User Interfaces, San Francisco.*
- McCallum, Andrew R. 1996. *Instance-Based State Identification for Reinforcement Learning.* Tech. rept. University of Rochester.

More details and papers available at <http://www.cs.ubc.ca/~pgorniak/um/index.html>