Imitating Agent Game Strategies Using a Scalable Markov Model

Priyath Sandanayake and Diane J. Cook
Department of Computer Science and Engineering
Box 19015
University of Texas at Arlington
Arlington, TX 76019-0015
{sandanay, cook}@cse.uta.edu

Abstract

Humans exhibit regularities in almost everything they do. We describe a Markov model derived from the behavior patterns of an agent, which is used to determine strategies by predicting which action a user is likely to execute next. We evaluate the predictive accuracy of this approach on a large dataset collected from sample Wumpus World games. We demonstrate from this approach that, the model can correctly predict the user's next action with minimal computation and memory resources. Such predictions can then be used to imitate player strategies in a variety of games and other strategic domains.

Introduction

Are we predictable enough to be imitated? We are all unique in our own ways and most of the time this is a good thing. But what about the times when we want to train some other individual to do some task just like we do it? Then we have to go through the process of training that person step by step through the task. We use this same approach when we want a computer to perform a task the way we do it. What if tomorrow, to do the same task the strategy is changed? We have to then program it all over again to do the same task differently. What if we also want to train an agent to perform some other task? Training with all these changes and new tasks constitutes a fair amount of work. Some of us, when training a person, take the approach of training by imitation, like the phrase we always hear "just watch and learn". This approach sounds like a more practical method in many ways. Can we use the same approach to train a software agent?

There are many applications that are developed which use this methodology. For example, when using a word processor such as Microsoft Word, if the user starts creating a list with bullets, Word will take the user into its *list* environment and help the user with the next bullet. This methodology is even used in Operating Systems. Windows Millennium uses this type of methodology by watching the applications a user employs, then displaying the recently used applications in the start up program bar hiding others. Most of these applications keep only the recent information it learned about the user, and do not take into consideration their past history and patterns when making such decisions. Users' history becomes very important in determining a good pattern of the users' strategy.

This paper investigates this problem. We design and implement a game known as *Wumpus World* to learn users' strategies and imitate them. We investigate how much of the history we need to incorporate to make the best model of a user in order to make good decisions. The following section describes our approach in more detail.

Related Work

Today there is a wide variety of computer applications ranging from web browsers to database systems that attempt to determine user patterns. Research has been performed by the community on user modeling to accomplish this task. Some researchers have used Bayesian Networks to infer user future actions from past behaviour (Horvitz et al., 1998) and (Albrecht et al., 1998). Bayesian networks and influence diagrams, in embedded applications allow them to make inferences about the goals of users, and to take ideal actions based on probability distributions over these goals. Horvitz et al. (1998) determined from the users' actions that the user is likely trying to define a custom chart in Excel, and Albrecht et al. (1997) determined from the users' actions that the user is likely trying to rescue a teddy bear. There are also other approaches taken, such as applying backpropagation neural networks to the real world problem of sorting e-mail messages based upon the sender's address (Gorniak 1998). This approach becomes a little problematic when trying to design the optimal network structure because the outputs frequently vary over time in real world settings. Both of these methods use informed modeling strategies, which means there is some prior knowledge of the model that is known. In most cases this prior information includes the task goal. Our approach requires no prior information about the application's purpose.

Other researchers have focused on independent user modeling strategies that do not build a model at all (Davison and Hirsh 1998) and (Korvemaker and Greiner 2000). They choose pairs of actions occurring in sequences as a pattern and build a simple probability table. The table is built by, increasing the probability of those action pairs that occurred recently and decreasing the probability of all others. They make predictions by selecting the action with the highest probability for the current state. These kinds of selections make an implicit Markov assumption, that the last action together with the current values provided by the probability table contains enough information to predict the next action. To determine patterns we need to consider more than the last action.

The motivation to our approach was based on the following two approaches, which explicitly builds a concept model. One approach predicts future actions by matching patterns from historical data (Gorniak and Poole 2000). The other predicts future actions using a Markov Model built from frequencies (Zukerman et al., 1999). In Gorniak's approach, the next action is predicted by choosing the longest sequences in history that match the current action sequence. This leads to problems of space when we take a large dataset into consideration. Zukerman's approach is the opposite of Gorniak's approach. Instead of keeping all the actions and states in history, they build a simple Markov Model that predicts the next action based on the previous action.

The foundation of our work is similar to the action prediction performed by Zukerman et al., who use simple Markov Models for web pre-caching. They describe several Markov models derived from the behavior patterns of many users, and present a hybrid model, which combines the individual models. In this paper, we start with a first-order Markov Model and work towards a sequence matching approach, trying to find a middle ground that will yield an optimal solution.

Wumpus World

We will test our approach using the Wumpus World game. Wumpus World is a square grid of locations where each location has the possibility of containing a pit, a pot of gold, an obstacle, a Wumpus, an Agent, or a combination of these elements. This grid world will contain many pits, pots of gold, obstacles, and Wumpii. Each location in the world is differed by a Cartesian system, with the Agent's initial location being the lower left grid cell (1,1).

The object of the game is for the agent to traverse the world collecting as many pots of gold as possible, return to the initial start location, and exit without getting killed by a pit or a Wumpus. The Agent gets killed if it moves to a location that contains a Wumpus or a pit. The Agent receives a reward only if it exits the world with one or more pots of gold. The Agent can select an action to execute from the following eleven possibilities: move up, move down, move right, move left, shoot up, shoot down, shoot right, shoot left, grab the gold, climb out of the world, or sit and miss a turn. The Agent selects an action based on its knowledge of the world, represented by the following five percepts: stench, breeze, glitter, bump, and scream. The Agent perceives a stench if there is a Wumpus in one of the four adjacent (horizontal and vertical) locations, a breeze if there is a pit in one of the four adjacent locations, a glitter if there is pot of gold at the current location, a bump if the Agent hits a wall or an obstacle while executing an action, and a scream if the Agent kills a Wumpus.

Data Preparation

In order to determine a sample software Agent's behaviors and actions in the Wumpus World application, we collected a dataset that contains 2000 games. The dataset was collected with one Agent playing the Wumpus World using the same strategy for all of the games. Our model has no information about the strategy the Agent is employing. It also does not have a global view of the environment, and does not store the history of the Agent's moves explicitly. It only tries to create a Markov Decision Process (MDP) model based on the Agent's local information and actions. This local information contains the Agent's percepts (stench, breeze, glitter, bump, scream), whether the agent is carrying gold or not, and what knowledge the agent has about the adjacent four locations (whether there is an obstacle and whether the Agent has visited the location). The collected dataset contains 19628 action state tuples.

The Models

We built a Markov Decision Process from the data that was collected. Each state was represented by a node and each action from a particular state was represented by an arc or transition from the current state to the state resulting from executing the action. The frequencies of actions that the user makes were used to generate transition probabilities. These probabilities were then used to make predictions to determine the users' next action. Each node that represents a state contained the following features. (1) Percepts – the percepts the agent perceives, (2) HaveGold – whether the agent has gold or not, and (3) AdjLocations – whether it was visited, and whether it has an obstacle into which the agent bumped. Each arc that represents an action contained a probability frequency. To illustrate how the Markov Decision Process was created from the data set, consider the following action state sequence example:

{ (s1, a1), (s2, a1), (s3, a2), (s1, a1), (s2, a1), (s3, a2),
(s1, a2), (s4, a2), (s3, a2), (s1, a2) }

Each state and the previous action leading to that state are represented by a tuple in this sequence. Figure 1 describes the model that was created for the above data set. The states are connected with the actions. Each action from state S_i has an associated weight, $w(S_i, a_j)$, which is the normalized frequency of action a_j. Thus, after observing a state, the probability that the next action is action a_i can be computed as follows.

$$\Pr(a_i \mid S_i) = \frac{w(S_i, a_i)}{\sum_{a \in succ(S_i)} w(S_i, a)}$$

To illustrate the calculation of the probability of a particular action from a particular state, let us reconsider the dataset, and assume that we are in the last state S_1 at the current moment. The probability of each action will be calculated as shown below.

$\Pr(a_1 \mid S_1) = 2 / (2 + 1) = 0.66$,
$\Pr(a_2 \mid S_1) = 1 / (2 + 1) = 0.33$

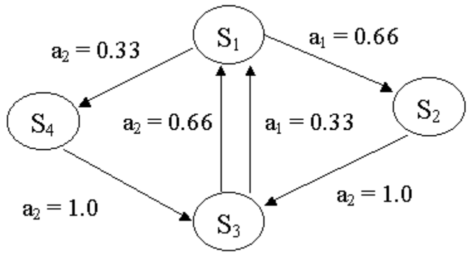


Figure 1. Initial Model

By the calculations, the probability of the next action from the current state S_1 would be a_1. The MDP that we created is a first-order Markov Model with no history included in the state representation. In practice this kind of model frequently does not perform well due to the fact that most of the actions one makes are influenced by past actions and results (not just the current state). Therefore we needed a method to improve the performance of predicting the next action. To improve this model, we included the previous action into the state representation. By including the previous action into a state description, predictive accuracies improved. Table 1 shows the improvement in prediction.

		Total Prediction	*Predicted Correctly*	*Total Number of States*
Next Action Predicted	*Initial Model*	19628	17462	373
	Augment. Model	19628	18208	604

Table 1: Performance values of the two models before merging

Augmenting our model by including the previous action into the state made the model much bigger. The number of states the model is able to contain became much larger. Our initial decision was to determine a point where we could include information into a state to make the prediction more accurate, but also not to insert too much information into each state that will create a huge model. Inserting the previous action into the state makes the model look different from the previous model, Figure 1. To illustrate how this new model that contains the previous action into the state appears, Figure 2 was created out of the example dataset. As seen, the number of states grew from four states into five states in our example dataset.

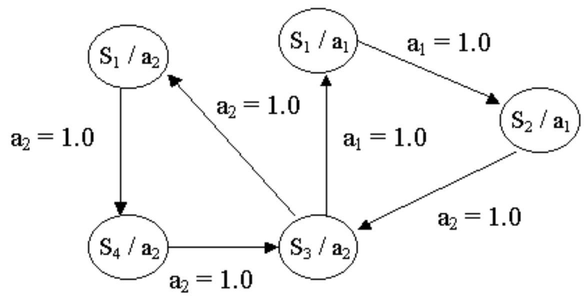


Figure 2: Augmented Model

We will illustrate the same example to see if we get the same performance as the previous method. Let us assume that we are in the same state as the previous example and consider calculating the probability of each next action.

$\Pr(a_2 \mid (S_1 / a_2)) = 1 / (1) = 1.0$,

From current state S_1 we predict action a_2. As can be seen from the example dataset, there is a pattern of action a_2 than action a_1, occurring from state S_1 after action a_2 occurring from state S_3. Therefore the Augmented Model that includes the previous action in the state gives a more accurate prediction than the Initial Model (in which the previous action is not included in the state representation). Table 1 shows the improvement that was obtained in prediction by inserting the previous action into the state representation.

Merging Process

Building a second-order Markov Model greatly increased the number of states in the model. To compensate for this cost, we selectively merged states. Merging states reduced the resource costs of the approach and reduced the amount of training data that was necessary to yield accurate predictions. In our implementation, we merged states if they were sufficiently similar and were rarely visited. With the merging approach, we can give more detailed information to states that are visited often, and generalize states that are not visited very often.

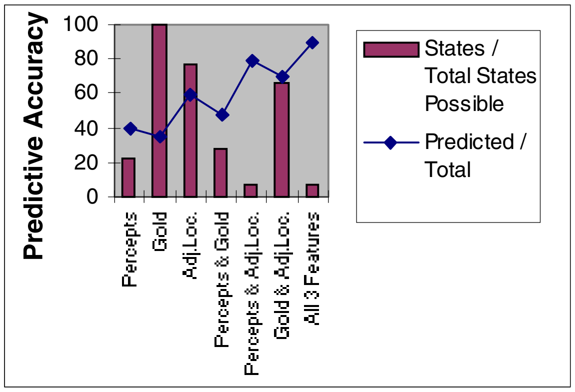
Merging Algorithm

We used two criteria in selecting states to be merged. (1) Merge states which are similar (states that have the same features except for the previous action), and (2) Merge states that are not visited that often (the minimum visit threshold, *T*, can be specified by the user). We executed the merge process every α number of games, where α is the number of games it will play before starting the merge process, and after the merging then continue another α number of games before executing the merge process again.

The states were merged as follows: first, the states that did not have a next action (the leaf nodes) were extracted from the model; next, the states that are visited below the threshold were selected. By comparing the selected states, all the states that contained a similar state representation (considering the attributes percepts, have gold, and adjacent location knowledge, and not the previous action) were combined together into a new state that generalizes the differing feature values by replacing them with a disjunction of the represented values. All the arcs that were linked to and from the merged states linked to and from this merged state. The probabilities of the next actions from this merged state were re-calculated after merging the states.

Experimental Results

We collected data on 2000 games played by known-strategy agents. The whole process was done on-line, which means while data is read and the model is updated, a prediction is made for the next action.



Graph 1: Predictive Accuracy and Percentage of States Visited for Different State Descriptions

In our first experiment, we evaluated the predictive accuracy of the model as the state descriptions become more detailed. The three initial features that were considered for the state description included the percepts the agent receives, whether the agent has gold or not, and the deduced contents of adjacent locations. Graph 1 compares the results of various combinations of these features. From these results, we saw that as we added more features, the number of actions that were predicted accurately compared to the total number of actions increased. We also saw that as we added more features, the percentage of the number of states visited to the number of states possible was very low. This was due to the fact that the states becoming more specific made the model capable of obtaining more states.

After picking the features that described a good state, we compared the initial three-feature model with an augmented model where we included the previous action in the state description. The results in Table 1 indicate that the augmented model yields a better performance than the initial model we implemented (without the previous action). The predictive accuracy of our augmented model using the 2000-game database was 91.8%.

Although the temptation is to include additional features, an increasing state representation complexity will ultimately result in a model that exceeds memory resources. In addition, a larger model requires a larger amount of training data to accurately represent the agent's strategy. State merging can be applied in this situation to reduce the space and training requirements of the model.

		α				
		100	**200**	**250**	**500**	**1000**
T	**1**	*15.56*	*14.90*	*14.07*	*12.25*	*9.93*
	2	*21.02*	*19.04*	*18.38*	*17.05*	*15.89*
	3	*24.01*	*22.18*	*21.19*	*19.20*	*17.38*
	4	*25.99*	*24.34*	*24.34*	*21.52*	*20.03*
	5	*27.32*	*26.16*	*25.33*	*22.51*	*21.19*

Table 2: Decrease in Percentage of States Visited

To determine the effect that state merging had on resource requirements and predictive accuracy, we calculated the performance when we apply the merging process to our augmented model. To generate the values shown in Tables 2 and 3, we collected results for five values of α and five values of *T*.

		α				
		100	**200**	**250**	**500**	**1000**
T	**1**	*1.120*	*0.730*	*0.632*	*0.132*	*0.066*
	2	*2.285*	*0.884*	*0.780*	*0.280*	*0.170*
	3	*3.207*	*1.988*	*1.285*	*0.341*	*0.192*
	4	*3.279*	*2.060*	*1.812*	*0.368*	*0.231*
	5	*3.438*	*2.416*	*1.944*	*0.395*	*0.247*

Table 3: Decrease in Predictive Accuracy

As seen in Table 2, the more often we merged the more the number of states decreased, and the bigger the value of *T*, the more the number of states decreased. Table 3 gives us the values of the predictive accuracy when performed the merging process. As we can see, there is always a drop in the predictive accuracy when we merge states. This is due to the fact that the merged states are more generalized than the rest. The data in Table 3 shows that the less often we merged and the smaller the value of *T*, we got the least prediction performance drop. Note that, overall when we applied the merge process every 100 games merging all the states that were visited one time or less, we got a huge drop of 15.56% in the number of states, costing only a 1.120% in predictive accuracy.

Conclusions

In this paper we have described an approach to imitating agent strategies by modeling the agent using a scalable Markov model. We first built a simple Markov Decision Process to create a user model for prediction that would not have the memory requirements of a sequence matching approach. Then we systematically determined how much information to include in the state description that would balance predictive accuracy with size limitations. We then considered how to incorporate the history benefits found in the sequence matching approach by adding the previous action to the state description. We have shown that adding history into the state description yields greater prediction accuracy than just using the current state knowledge. Further, to preserve scalability, we applied a state merging technique and evaluated the results. By doing so, we showed that although prediction accuracy decreased, it decreased by a small percentage compared to the gain in decreasing the model size. The decrement of the accuracy could have been due to the features selected for the merging. Selecting different features of the state that will meet the best merge, could have improved the accuracy. After building a MDP and applying a state merging technique, we conclude that the overall success of this approach has been positive, which is building a simple Markov Model on-line that will also include enough history to get a $(90.1 \pm 1.7)\%$ accuracy.

Future Work

We are refining our approach to imitating agent strategies in the Wumpus World game by trying to include more information in the state description, without losing too much of the prediction accuracy. Since the enhancement of adding history (like the previous action) improved our predictive power, we plan to build a higher-order Markov Model such as a second-order Markov Model. Another consideration for future work is, applying a different merging technique such as '*Best-first model merging*', or Decision Trees, which will give an increase in the predictive accuracy.

We are also considering a state splitting method to improve the prediction, such that we could split states that are frequently visited in order to generate more detailed predictions that incorporate a longer action history.

References

Albrecht, D. W.; Zukerman, I.; and Nicholson, A. E. 1998, Bayesian Models for Keyhole Plan Recognition in an Adventure Game, *User Modeling and User-Adapted Interaction.*

Davison, B.D.; and Hirsh, H. 1998, Predicting sequences of user actions, *Technical report*, Rutgers, The State University of New York.

Gorniak, P.J. 1998, Sorting email messages by topic, Project Report, University of British Columbia.

Gorniak, P.J. 2000, *Keyhole State Space Construction with Applications to User Modeling*, Masters thesis, University of British Columbia.

Horvitz, E.; Breese, J.; Heckerman, D.; Hovel, D.; and Rommelse, K. 1998, The lumiere project: Bayesian user modeling for inferring the goals and needs of software users, *Uncertainty in Artificial Intelligence, Proceedings of the Fourteenth Conference*.

Korvemaker, B.; and Greiner, R. 2000, Predicting UNIX Command Lines: Adjusting to User Patterns, *Proceedings of the 17th National Conference on Artificial Intelligence*.

Zukerman, I.; Albrecht, D.W.; and Nicholson, A.E. 1999, *Predicting Users' Requests on the WWW*, User Modeling: Proceedings of the 7th International Conference, UM99.