ONASI: ON-LINE AGENT MODELING USING A SCALABLE MARKOV MODEL

PRIYATH T. SANDANAYAKE

Department of Computer Science and Engineering, University of Texas at Arlington, Box 19015 Arlington, Texas 76019-0015, U.S.A.
priyath@ uta.edu

DIANE J. COOK

Department of Computer Science and Engineering, University of Texas at Arlington, Box 19015 Arlington, Texas 76019-0015, U.S.A.
cook@ cse.uta.edu

Human and software agents exhibit regularities in their activities. We describe our ONASI algorithm, which derives a Markov model from such observed regularities, and dynamically scales the model through merging and splitting of states. ONASI uses this model to predict the agent's next action. We evaluate ONASI's predictive accuracy on a dataset of Wumpus World games, and demonstrate from this approach that the model can correctly predict the agent's next action with adjustable computation and memory resources. These predictions can be used to imitate, assist or obstruct an agent.

Keywords: User Modeling, Machine Learning, Game Strategy, Markov Models

1. Introduction

With the increasing popularity of computers and the Internet, researchers have focused on user modeling as a means of providing tools customized to individual users. In user modeling research, an agent is observed and modeled by drawing inferences about the agent's behavior. Much of the user modeling research has focused on specific applications within which either the details of the application are known ahead of time, or the user's actions have a known meaning and known effects.

Popular user modeling applications abound in which the user's actions are given a known meaning from a set of rules that is incorporated into the model. For example, when using a word processor such as Microsoft Word, if the user inserts a bullet, Word will help the user by inserting the next bullet, predicting that the user most probably will continue with another bullet. Similarly, some operating system environments display recent user commands or applications while hiding others, predicting that the user will probably start up the recently-used user command or application next. Such applications typically keep only the recent information they gathered about the user or rely upon a set of predefined modeling rules. One problem with such approaches is that knowledge of recent activities is not always adequate to learn a user's behavior, because the user's pattern history may influence the behaviors of the user. Another problem is that implementing a set of rules and goals into an application can be a tedious job.

In contrast, we consider an approach to modeling an agent's behavior given no information about the application or significance of actions within the application.

Attempting to model a user's behavior without this information is similar to trying to learn an opponent's chess strategies without having any knowledge about the game or his goals. One approach to model an agent's activities is to build a Markov model of observed actions. Unfortunately, the size of these models can grow exponentially with the number of possible actions. We describe an approach that dynamically adjusts the size of the Markov model in order to optimize modeling performance without imposing excessive storage constraints on the problem. We demonstrate the validity of our ONASI (ON-line Agent Strategy Identification) algorithm by modeling an agent playing the game called *Hunt The Wumpus*. We model the agent's behavior by building a Markov model without incorporating any knowledge of the application's structure or the agent's goals into the model. Such a model can then be used either to assist (cooperative) or obstruct (adversarial) the user.

2. User Modeling Techniques

Today there is a wide variety of techniques, ranging from web browsers to database systems, which attempt to determine user behaviors. To accomplish this task, the community has extensively researched user modeling algorithms. Some researchers have used Bayesian networks to infer user future actions from past behavior [1, 10]. Bayesian networks and influence diagrams are embedded in applications that 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. [10] determine from the users' actions when the user is likely trying to define a custom chart in Excel, and Albrecht et al. [1] determine from the users' actions when the user is likely trying to rescue a teddy bear in a game. Other approaches include applying back-propagation neural networks to the real world problem of sorting e-mail messages based upon the sender's address [7]. This approach uses neural networks to capture user's preferences.

Some researchers have focused on user modeling strategies that do not rely on domain knowledge, and in fact do not build a model at all [5, 13]. In this approach, a probability table stores relative frequencies of pairs of actions occurring in sequence. The probabilities of those action pairs that occurred recently are increased, and the probability of all other sequences are decreased. Predictions are made by selecting the action with the highest probability given the previous action that was executed. This approach makes an implicit Markov assumption, that the last action contains enough information to predict the next action.

Using Bayesian Networks or Neural Networks 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 typically use informed modeling strategies, which means there is some prior knowledge of the model. In most cases, this prior information includes the task goal. There is also a need to consider more than the last action when determining patterns. The purpose of this research is to determine if a scalable Markov model can effectively model an individual's behavior patterns with no

prior information about the application or the goals of the individual. The model is constructed using only a history of the individual's actions.

Other work uses case-based reasoning to recognize a plan and subsequently predict user actions. This work by Kerkez and Cox [11, 12] represent states using first-order predicate calculus. If goals and plans are clearly specified this is a viable approach. This assumption does not hold for many of our problem domains. Also, the approach does not represent external events and changes to the environment, and stores the entire history of actions. We desire a method that overcomes these limitations.

The motivation for our approach is based on the following two related projects. Gorniak's [8] approach is to predict future actions by matching patterns from historical data. In this work, 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 scaled to a large dataset. In contrast, Zukerman's [21] approach is to build a simple Markov model from the collected data and predict the next action based on the current state in the model.

The foundation of our work is similar to the action prediction performed by Zukerman et al. [21], 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 patterns from individual models. In this paper, we start with a first-order Markov model and work towards a sequence-matching approach (order-n Markov model), trying to find a middle ground that will yield an effective solution.

3. Model Setup

Our approach is tested using a game called *Hunt the Wumpus* (which we refer as the *Wumpus World*). The Wumpus World is an early computer game, which contains a square grid of locations. 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 identified by a Cartesian coordinate system, with the agent's initial location being the lower left grid cell (1,1). At each point in the game, 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.

Knowledge of the object locations is determined only by the logical reasoning from the percepts the agent perceives. The Wumpus World game is chosen to validate our approach because it provides an excellent environment for designing and testing intelligent agents. The agent needs to have some kind of logical reasoning ability to be successful in this game. Our goal is to show that our scalable Markov model can successfully model an agent capable of playing a game that requires logical, strategic reasoning such as the Wumpus World.

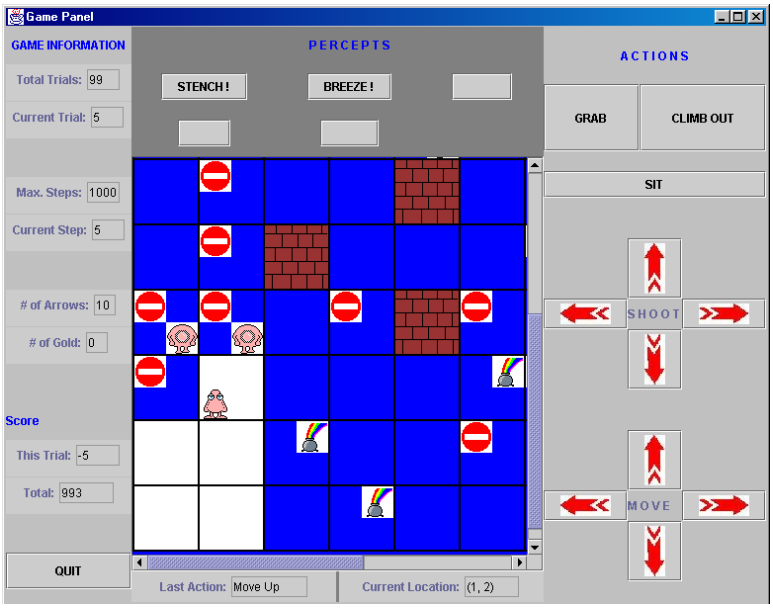


Fig. 1. Wumpus World game.

Using our implementation shown in Figure 1, a sample software agent that performs two different strategies is created to play the Wumpus World game. The agent uses the same two strategies for all of the games. A data set is collected that contained 2,000 games to create a Markov model [18]. Without containing any information about the strategy the agent employed or the environment, the data set contains only the information it can gather from the interface, which is the same information a software agent (or even a human player) can obtain from playing the game. The collected data set contains 19,628 data points, representing individual agent moves.

3.1 *Initial model*

The model is built without any knowledge of the environment. In particular, the size of the environment (world) or the environment being deterministic or not (in this case, non-deterministic) is unknown. Due to the ambiguity of the environment, a decision is made to implement a model with each state containing only the information in the agent's local area. Representing too much information would increase the number of states necessary to model the problem, with possibly limited gains in performance.

ONASI builds a first-order Markov model from the collected data set. A node represents a state (the current state) of the agent and an arc (or transition) represents an action the agent performs from the current state that leads to the resulting state. The relative frequencies of agent actions are used to generate transition probabilities. These

probabilities are then used to predict the agent's action from any state in the model. To illustrate how ONASI creates a Markov model from an action history, consider the following action-state sequence example:

$$\{(S_2, a_1), (S_3, a_2), (S_1, a_1), (S_2, a_1), (S_3, a_2), (S_1, a_2), (S_4, a_2), (S_3, a_2), (S_1, a_2)\}$$

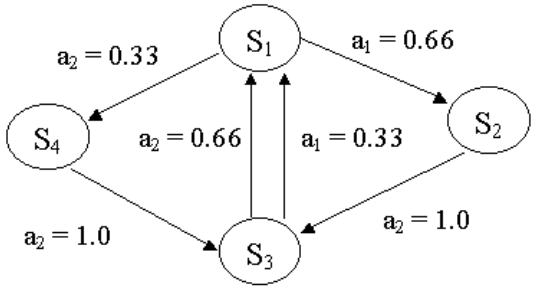


Fig. 2. Initial Markov model.

Each tuple in this sequence represents a state and the previous action leading to that state. Figure 2 describes the initial Markov model that is created for the above data set. The states are connected by edges labeled with actions. Each action from state S_i has an associated weight, $w(S_i, a_j)$ representing the frequency of the corresponding move. For a given state, the probabilities of the possible next actions a_j is computed as follows:

$$\Pr(a_j \mid S_i) = \frac{w(S_i, a_j)}{\sum_{a \in Succ(S_i)} w(S_i, a)}. \quad (1)$$

Let us illustrate the calculation of the probability of a particular action given the model in Figure 2. Assume that the current state is S_1. The probability of each action is calculated using Eq. (1) as follows:

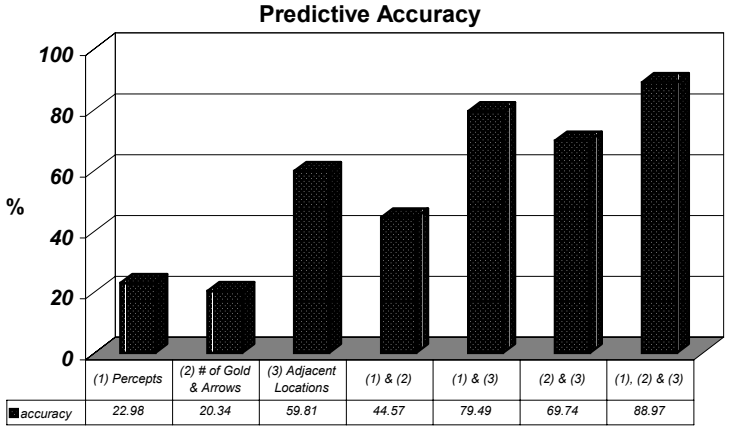
$$\Pr(a_1 \mid S_1) = 2 / (2 + 1) = 0.66$$
$$\Pr(a_2 \mid S_1) = 1 / (2 + 1) = 0.33$$

Agent's next action is predicted by observing the current state. The action with the highest probability from that state is output as the predicted next action. In our example, the predicted next action from the current state S_1 is a_1.

The collected data set is used to build our initial model. Each time a data point is read, the model is incrementally refined and ONASI predicts which of the eleven possible moves the agent will select next. The model is incrementally refined with each data point, and the predictive accuracy is averaged over the entire set of games. To test the influence of state features on the algorithm performance, the predictive accuracy was measured as

the state features were added. As more features were added, the predictive accuracy consistently improved. Inserting all the features that are observable (without any knowledge of the environment), the greatest predictive accuracy achieved was 88.97%. Table 1 displays ONASI's predictive accuracy as a function of the state features.

Table 1. Initial model predictive accuracy.



3.2 *Improved model*

In the initial model, all of the observable features were included. If more details about the application were included (such as the rules of the Wumpus world game), performance would have improved even further. To improve ONASI's performance without utilizing extra knowledge of the application, the agent's previous action is integrated into the state description (making this an order-1 Markov model). Most of the actions one selects are influenced by past actions and states, not just the current state. Therefore, including the previous action into the state description should result in better predictive accuracies. A new and improved model is built by including the previous action into the state description as a feature.

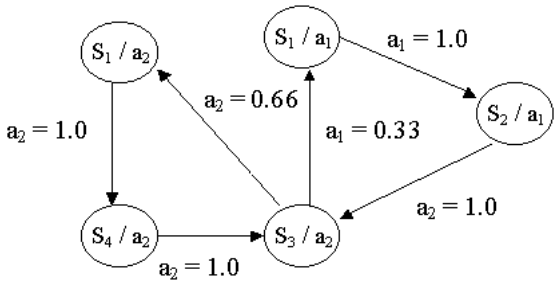


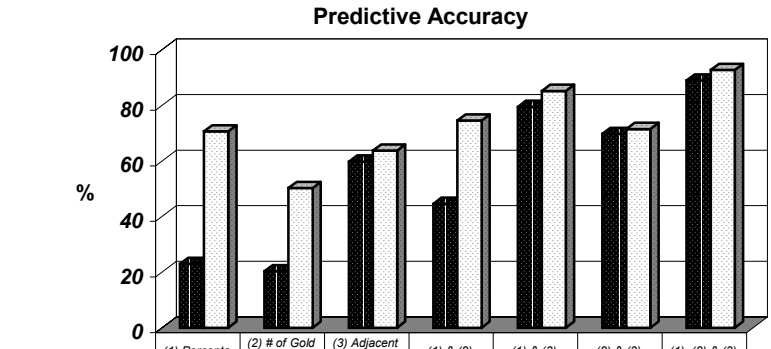
Fig. 3. Improved Markov model.

Inserting the previous action into the state makes the state more specific than in the previous model, and increases the size of the model. The model shown in Figure 3 is created from the same data set as the model in Figure 2, by integrating the previous action into the state representation. By including the previous action in the representation, the size of the model increases from four to five states.

Let us illustrate how the prediction differs from the initial model. Assuming the current state is S_1 and the previous action was a_2, the probability of the next action is calculated using Equation 1 as $Pr(a_2 | (S_1 / a_2)) = 1 / (1) = 1.0$.

From current state S_1 action a_2 is predicted, where in the initial model action a_1 was predicted as the next action. Which model yields a better prediction? In the given example data set, there is a recurring pattern of action a_2 leading to state S_3, followed by action a_2 leading to state S_1, followed by action a_2. If the current action sequence matches this pattern, action a_2 may indeed be a better prediction. Given a sufficient amount of training data, the more informed model (the improved model) should outperform the original Markov model. Our assumption is empirically confirmed by the results that are obtained using the Wumpus World games. Tables 2 and 3 illustrate these results.

Table 2. Improved model predictive accuracy.

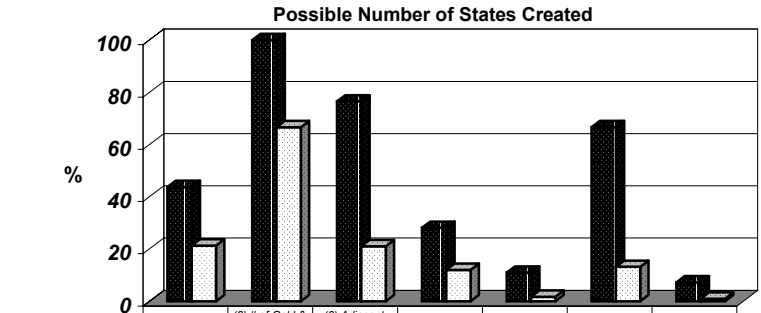


	(1) Percepts	(2) # of Gold & Arrows	(3) Adjacent Locations	(1) & (2)	(1) & (3)	(2) & (3)	(1), (2) & (3)
without previous action	22.98	20.34	59.81	44.57	79.49	69.74	88.97
with previous action	70.69	50.28	63.79	74.59	85.23	71.52	92.77

Table 2 shows the improvement in predictive accuracy resulting from inserting the previous action into the alternative state representations. Table 3 provides the percentage of growth in the number of states created by the model. There is a 92.77% predictive accuracy resulting from the new model, which reflects an improvement of 3.79%. Including the previous action into the state description makes each state more specific, which leads to a 61.93% increase in the number of states the model contains. Although there is an increase in the number of states, the percentage of the total possible number of created states decreases, as can be seen in Table 3. Due to a state being

described by more features, the possible number of states the model can include becomes greater.

Table 3. Number of states created by improved model.



	(1) Percepts	(2) # of Gold & Arrows	(3) Adjacent Locations	(1) & (2)	(1) & (3)	(2) & (3)	(1), (2) & (3)
without previous action	43.75	100.00	76.54	28.13	10.96	66.67	7.20
with previous action	21.35	66.67	21.09	11.98	1.71	13.17	0.97

After testing the improved model on our Wumpus World data set, the achieved results are shown in Figure 4. Due to the software agent being a reflex (rule-based) agent, the agent will not visit all possible states. The performance grows initially and plateaus after a certain amount of data is processed. The plateau starts at the point where there are no more new states for the model to learn.

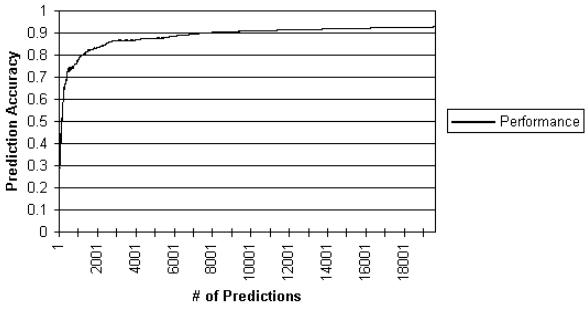


Fig. 4. Performance of the improved ONASI model.

After testing the improved model on our Wumpus World data set, the achieved results are shown in Figure 4. Due to the software agent being a reflex (rule-based) agent, the agent will not visit all possible states. The performance grows initially and plateaus after a certain amount of data is processed. The plateau starts at the point where there are no more new states for the model to learn.

From this experiment, we conclude that the more features we use to describe a state, given sufficient training data, the better the predictive accuracy is. We also conclude that including action history in a state representation increases the predictive accuracy. This leads to the question of how much history should be included in a state. The higher the order of the Markov model is, the more informed would be the prediction, at the expense of creating a larger model. We now describe a method for automatically adjusting the size of the Markov model.

4. Splitting

Although adding features can improve the predictive accuracy of a model, the resulting increase in size will affect storage and computational requirements, as well as increase the amount of needed training data. Determining the ideal size of a model is a problem-specific task that requires substantial effort to balance size with predictive accuracy. We describe a method in which the size of the model can be adjusted automatically, through dynamic merging and splitting of states in the Markov model. Merging states can reduce the size of the model and is performed when accuracy will not be significantly affected. Splitting states, on the other hand, can increase the accuracy at the cost of additional size.

4.1 *Prior splitting methods*

There has been research performed by the community on model splitting to explicitly refine the states in a model. One approach is using an instance-based algorithm with a tree to store instances and to represent states for reinforcement learning. McCallum [15] introduces a Utile Suffix Memory (USM), which is an instance-based algorithm that performs online instance-based state learning, permitting state definitions to be updated quickly based on the latest results of reinforcement learning. USM uses statistical tests to determine the relevance of history information considered for inclusion in state definitions.

Another approach is to replace states that do not make good Markovian states with states that do, by splitting those states. Gorniak [9] replaces the old state with new states by mining the recorded history and attaching history segments to each new state that yield better predictions. The replacement is accomplished by grouping sequences of actions such that the groups give as much information as possible about what action to take from the current state. Such groupings can be performed given the recorded history of action sequences that precedes the state as well as the actions that follow that state.

Using all of the preceding action sequences to determine new states will end up creating a large number of new states after splitting. Therefore, to overcome this problem, Gorniak [9] dynamically constructs attributes as sets of history segments with only a few values for each attribute. This approach considers only fixed-length history sequences in the recorded history that precede the state. Binary splits are made based on information the split yields about outgoing transitions, or actions. Using a method similar to building

a decision tree, attributes are selected that generate the greatest information gain about the actions that led up to a state. The best split is determined from the information gain thresholds, the sum of the number of action instances, and the sequence length. One major problem in using such an approach is that explicitly retaining the observed history increases resource and memory cost. In addition, the length of the history sequence in the recorded history has to be determined ahead of time.

Instead of explicitly retaining the observed history and attaching the history segments to the new states, we take a much less costly method by storing only the relationships of *previous states* (the states that have a transition to the current state) to *next states* (the states that have a transition from the current state). This avoids retaining all action history. It also avoids trying to determine a length of the history to construct dynamic attributes. Although keeping all of the history might yield a better split, the relationship between the previous states and the next states is sufficient to make good splits without any overhead of resources and memory. Due to the lack of overhead in our model, the model is constructed on-line.

4.2 *Model splitting method*

Our method is somewhat similar to the model splitting method introduced by Brants [4] to derive Markov model structures from text corpora. A selected state is divided into two new states, and the two states adopt the description of the original single state. The transitions leading to the old state is redirected into two transitions leading to the two new states, and distributes the outputs from the old state between the two new states. The transition probabilities are transferred to the new states without changes. The outputs from the new states are the same as the original state but with different probabilities. The probabilities of these outputs are calculated with the additional information that is obtained by the maximum likelihood from the training corpus.

To determine a good split, Brants finds probability distributions resulting from potential splits that maximize the divergence. Due to the exponential growth of splits they use a greedy method to derive a local maximum instead of the global maximum. Out of all the possible states into which a state can be split, they extract two split states that maximize the divergence of their respective probability distribution. Then from these two extracted split states, they build two sets; each containing one of the two extracted split states. For all other states resulting from the split, they add them to whichever of the two sets yields the minimum divergence.

ONASI differs in two ways from this method. The first is, additional information other than what is stored in the state is not used. Secondly, the remaining split states are assigned using minimum entropy, in a manner that minimizes the impurity of the distribution instead of the divergence.

4.3 *ONASI splitting algorithm*

There are two thresholds that need to be met for a state to be selected for splitting. One threshold, V_{min}, represents the minimum number of visits a state must visit, and the other threshold A_{max} represents the maximum probability of the *strongly preferred action* (an outgoing transition or action that has a very high probability compared to the rest of the actions) from that state. If a state has two or more strongly preferred actions, the state is split into more specific states until each split state yields only one strongly preferred action. For example, let {0.05, 0.4, 0.45, 0.1} represent a probability distribution over actions taken from a given state. There are two strongly preferred actions in the above probability frequency of actions, with probabilities 0.4 and 0.45. A state also has to be visited a minimal number of times to be selected for splitting. By doing so, the effect of noisy data on the splitting decision is reduced. Therefore, if the number of visits to that state is greater than V_{min}, and the highest probability for a next action is less than A_{max}, then that state is considered as a candidate to be split.

Each state S that satisfies the thresholds is split into two similar states S_1 and S_2. The split states S_1 and S_2 differ from each other by the unique path they contain from the *previous states* (state with a transition to the state S). By taking such an approach, implicitly history is included into each split state. The previous states are clustered into two sets, where each cluster of states consists of a subset of the previous states for a given split state. The clustering is done such that the two sets of states provide as much information as possible about the next action to take from the state S. A *decision stump* (decision tree of height one) is used to cluster the set of previous states. Measuring of how well a given state feature separates the previous states is done according to their target classifications (next action taken from the current state) by using the *information gain* [19]. The quality of the split is determined by the probability distribution of the next action from the current state. Therefore, the previous states are split according to an attribute (a feature describing the state) that gives the highest information gain. The following equation is used to calculate information gain:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \cdot \quad (2)$$

where

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i \cdot \quad (3)$$

The values of c in Eq. (3) represent the alternative next actions that can be chosen from the current state. The previous states are separated according to the values V_i of the selected attribute. Let States(V_i) be the set of previous states with value V_i. States(V_h) and States(V_{2h}), two States(V_i) sets that have the highest and the second highest *predictability* out of all the States(V_i) are selected. By predictability here we refer to an outgoing action probability distribution that is heavily skewed toward the strongly

preferred action. Starting with these two selected States(V_i), two sets are built, one of which contains previous states yielding distributions closer to States(V_h), the other one contains previous states yielding distributions closer to States(V_{2h}). The rest of the States(V_i) are attached to one of the two sets by finding the combination that yields the minimum entropy (the lowest impurity). Each one of the sets will represent previous states for a split state. Figure 5 shows this process, which is the splitting algorithm for ONASI. After the *BinarySplitState* function in Figure 5 is executed, we check to see if the split states satisfy the two thresholds. If any one of the split states does not satisfy these values, then that state is split into two more states by repeating the above process.

BinaryStateSplit(*State*, V_{min}, A_{max})
State is the state in the model
V_{min} *is the threshold for minimum number of visits a state is visited*
A_{max} *is the threshold for highest probability for a next action*
- If the thresholds V_{min} and A_{max} are met, then
- A ← The attribute from Attributes that yields the highest Information Gain
- For each possible value, V_i of A
- Let States(V_i) be the subset of previous states that have value V_i for A
- States(V_h) ← The States(V_i) that has the highest number of same target attributes
- States(V_{2h}) ← The States(V_i) that has the 2nd highest number of same target attributes
- Let S_1 and S_2 be the split states containing States(V_h) and States(V_{2h}) respectively
- For each possible value, V_i of A (except V_h and V_{2h})
- Add to S_1 or S_2 the States(V_i) that yields the lowest entropy

Fig. 5. Splitting algorithm for ONASI.

4.4 *Example*

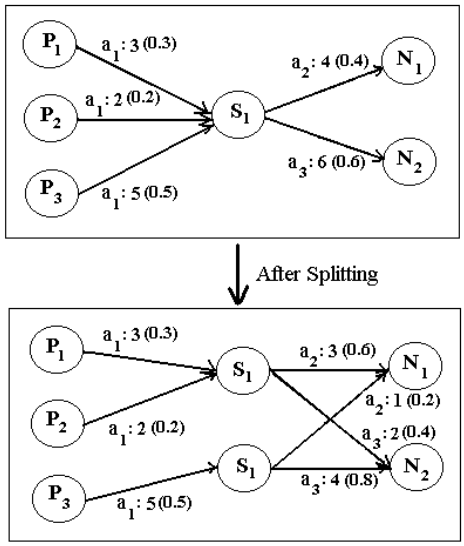


Fig. 6. Splitting example.

Let us look at the example depicted in Figure 6. As shown, the current state S_1 is preceded by previous states P_1, P_2, and P_3, and is followed by states N_1 and N_2. The current state S_1 contains the relationships between the previous and next states shown in Table 4: Column N_1 displays action a_2 taken from the previous states resulting in state N_1 and column N_2 displays action a_3 taken from the previous states resulting in state N_2.

Table 4. Transition Distributions of Action through Current State

		next states	
		N_1	**N_2**
previous states	**P_1**	2	1
	P_2	1	1
	P_3	1	4

For example, if the previous state is P_1, the agent has executed action a_2 twice from the current state resulting in next state (N_1); and if the previous state is P_3, the agent has executed action a_3 four times from the current state resulting in state N_2. In this example, the threshold for the minimum number of visits V_{min} is 10, and the threshold for the maximum next action probability A_{max} is 0.6.

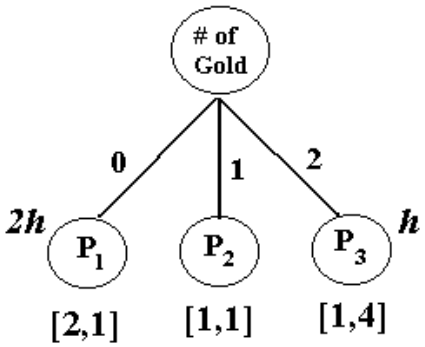


Fig. 7. Decision stump.

Eq. (2) is used to obtain a state feature as the root node for the decision stump. After determining the attribute that yields the highest information gain, and separating the previous states into the attribute values V_i, the decision stump is obtained as shown in Figure 7. The feature 'number of gold' is selected as the root node with the highest information gain, and the previous states P_1, P_2, and P_3 are separated into 3 child nodes according to their respective feature values (0, 1, and 2). Each child node contains a distribution of the number of next actions taken. For example, [2,1] means action a_2 was

executed twice and action a_3 was executed once. From these distributions we determine that previous state P_3 has the highest predictability, and P_1 has the second highest predictability. The two split state clusters are formed, by adding previous state P_2 to either the cluster with state P_1 or P_3, depending on which combination yields the lowest entropy. Combining P_2 with P_1 will yield an entropy of 0.97095 and combining P_2 with P_3 will yield an entropy of 1.069675. Since the former combination yields the lowest impurity, this is chosen.

After the split, the split states look as shown in Figure 6, along with the relationships to the previous and next states. Since the two split states satisfy both thresholds V_{min} and A_{max}, they are not considered for further splits.

4.5 *Splitting algorithm results*

To determine the effect of state splitting on performance and model size, the ONASI splitting algorithm is run on the Wumpus World data. There is a slight increase in the predictive accuracy when the splitting threshold (A_{max}) is between 0.44 and 0.5. The greatest increase is obtained when A_{max} is 0.5. The accuracy decreases as A_{max} increases beyond 0.5. Since there is no need to split a state if A_{max} is greater than 0.5 (which means there is a strongly preferred action with a high probability), the accuracy decreases as A_{max} increases beyond 0.5. The number of states increases with the number of splits and the splitting threshold. Figures 8 and 9 show the effect on the performance and the model size, respectively.

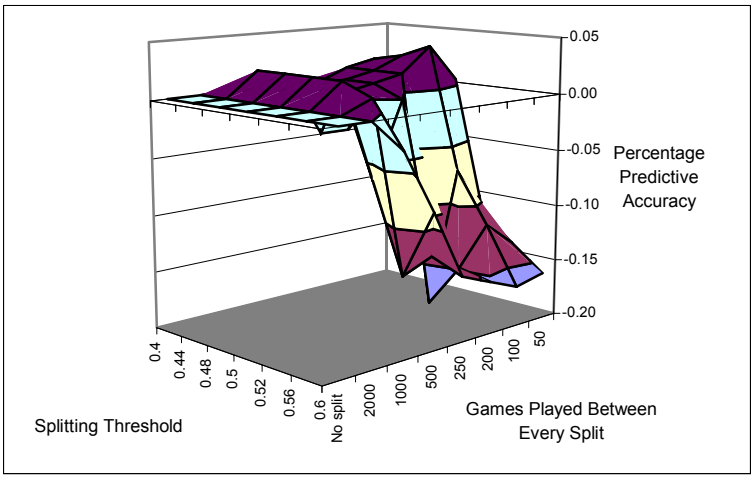


Fig. 8. Predictive accuracy.

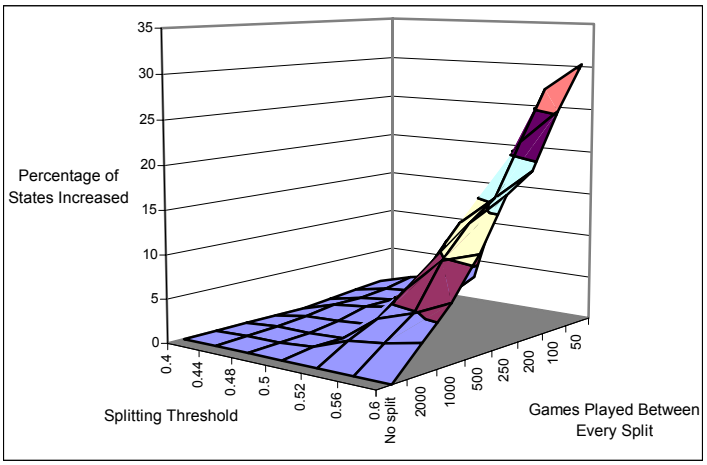


Fig. 9. Increase in model size.

5. Merging

5.1 *The problem*

When building a model of the agent's strategies, every specific detail about the state and action of the agent must be taken into account. This is critical especially if there is only a small amount of data that is available. However, as more data is gathered and states are visited more often, it is not so critical to keep such detailed states. To control the size of the model, states can be merged, by clustering them according to similar state features. This decision operates under the assumption that similar states will yield similar action predictions. Often a Markov model contains several very similar states. In some cases, the action most often selected from similar states is different, due in part to the histories that led to the states. In other cases, however, similar states do yield similar action selections. In such cases, we could generalize states by merging these states on the basis of certain similarities in state features.

With the slight improvement of predictive accuracy that results from splitting states, there was a subsequent increase in the size of the model. If each state is split to take history into account for the purpose of generating greater predictive accuracy, there is a tendency for the model to grow exponentially. The number of possible states is calculated by multiplying all the values each feature contains, as shown in Eq. (4).

$$\textbf{Total \# of states} \equiv \prod_{i=1}^{c} (V_i) \ . \tag{4}$$

where

V_i = number of values for each feature *i*
***c* = total number of features**

To yield better predictive accuracy, a state (or a split state) can be split to a maximum number of N states, where N is the number of actions the agent executes. This will lead the model to grow exponentially. An approach must be implemented to reduce the size of the model. Therefore, to compensate for the cost of the increased number of states, states are selectively chosen and merged. Merging states reduces the resource costs of the approach, and also reduces the amount of training data that is necessary to yield accurate predictions. The goal is to merge states such that the model reduces in size, but does not decrease in the predictive accuracy.

5.2 *Prior work on merging*

There has been research performed by the community on model merging to generalize the states in a model. One approach is called "best-first model merging," which is a technique for dynamically choosing the structure of a related architecture while avoiding over fitting [16]. This technique was also introduced to induce models for regular languages from a few samples [20], and has been adapted to natural language models [2, 3]. The process starts with a large specific model consistent with the training data and successively combines states to build a smaller and more general model. Brants [4] also uses this technique for inducing model parameters for Markov models from a text corpus. Here, Brants groups words from the text corpora into categories using the model merging technique.

The *model merging* technique induces Markov models as follows. There is one path in the model for each utterance in the corpus and each path is used by one utterance only. Each path is assigned the same probability. The states of the model are merged successively by selecting two similar states. These two states are removed from the model and are replaced with a new merged state. The transitions from and to the two similar states are directed to the new state by adjusting the transition probabilities to maximize the likelihood of the corpus. The criteria for merging states is based on the probability of the Markov model generating the training data. That is, out of all the possible merges, the merge that results in the minimal change of the probability is selected. The amount of time needed to use the above method is $O(n^4)$, where n is the length of the training corpus.

The model merging method merges states that are similar due to the redundancy of similar states created by each path in the model. Our model may contain similar states, but only due to ONASI's state splitting algorithm. Merging such states is not going to help; this type of merge will just create a cycle of model adjustments. Instead, we pick states that are similar in their feature description as well as the outgoing action probability distributions. By doing so, the predictive accuracy would be decreased. Also, instead of merging only two states at a time, the states are separated into clusters of states and all the states in the same cluster are merged together. This avoids the large computation cost inherent in the best-first model merging technique. The selection of states to merge is implemented according to the following conditions:

1. Select pairs of states that share some of the feature values.
2. Select pairs of states that contain the same preferred next action.

We execute the merge process every α number of ONASI games.

5.3 *Merging algorithm*

The merging algorithm induces a Markov model in the following way. Each state in the model is assigned a target attribute (classification), which is defined as the action executed from the corresponding state with the highest probability (the strongly preferred action). If the probability of the strongly preferred action is above the merging threshold M_{min}, then the state is selected as a candidate to be merged with another state. The effect that noisy data may have on a state is offset, by selecting a state that has a high probability for the strongly preferred action. Once the states are selected that satisfies M_{min}, these states are clustered using a decision tree learning algorithm. The specifications for the learning algorithm based on the Markov model are as follows:

1. Examples: States that are selected to be merged
2. Attributes: Features comprising the state description
3. Target Attribute: Action that has the highest probability

A decision tree learning algorithm is used because it is one of the most widely used and practical methods for inductive inference. This method for approximating discrete-valued target functions is robust in the presence of noisy data. Since ONASI uses decision tree learning for approximating discrete-valued functions, a high value must be selected for the merging threshold to classify the states correctly. The algorithm is used to create clusters of states instead of outputting the generated decision tree. Keeping a high merging threshold M_{min} minimizes the decrease in the predictive accuracy, which prevents the wait until sufficient amount of data is available to merge states.

By using the features included in the state representation, the attributes for the training data are created. From a list of these attributes and the set of states as training examples, ONASI uses the ID3 algorithm [17] to separate the states into good clusters. Terminating the recursion of the algorithm is based on two decisions: either all the states have the same target attributes, or nodes cannot be further split (all attributes have been used).

ModelMerging(*Model*)
Model is the Markov model that was built by ONASI
- For each state in the model, S_i
- Let $T(S_i)$ be the target attribute for S_i, which is the *strongly preferred action*
- If $T(S_i)$ is greater than the merging threshold M_{min}
- Add S_i to the *MergeList*
- Create an *AttributeList* from the given state features
- MergeStates(*MergeList, Target Attribute, AttributeList)*
- Add the new merged states into the model with the appropriate arcs from and to the states

Fig. 10. ONASI's model merging algorithm.

When merging a set of states, a new state is created from the set of states. The new state is defined from the attribute values of the cluster as follows. If all the states have the same attribute value for a given attribute, this value is passed on to the new state. If all the states have different attribute values for a given attribute, then the new state attribute value for that attribute is the union of all the values. Figures 10 and 11 describe the model merging algorithm for ONASI.

MergeStates(*MergeList, Target Attribute, AttributeList)*
MergeList contains the states that satisfy the merging threshold that is going to be merged.
Target Attribute is the attribute whose values is to be predicted by the tree, which is the next action from the current state.
AttributeList is a list of other features of a state that may be tested by the learned decision tree.

- If the target attributes of all the states are the same,
- Merge all the states together
- Else
- **A** ← The attribute from Attributes that has the highest information gain
- Subtract attribute **A** from the attribute list
- For each possible value, V_i , of **A**
- Let *Examples*(V_i) be the subset of *Examples* that have value V_i for **A**
- If *Examples*(V_i) is not empty
- Repeat MergeStates for each subset of *Examples*

Fig. 11. ONASI's merge states algorithm.

5.4 *Example*

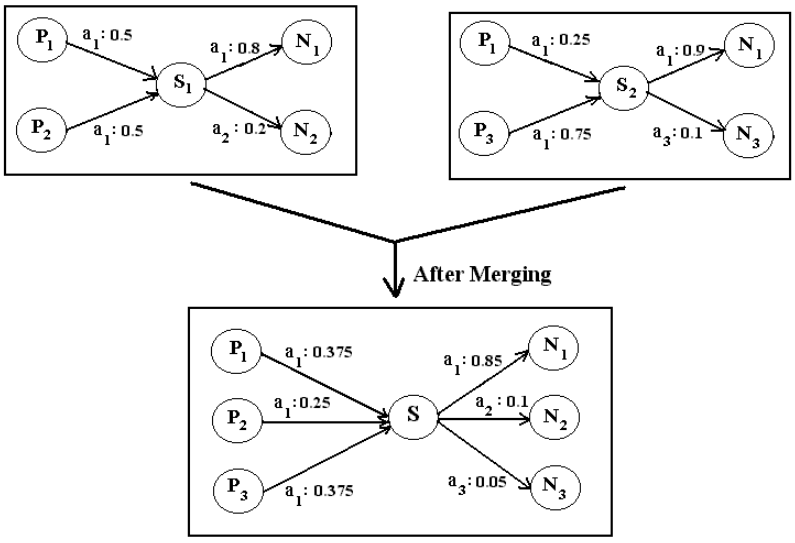


Fig. 12. Merging example.

Let us look at an example depicted in Figure 12. In this example, M_{min} is equal to 0.75, which means states S_1 and S_2 satisfy the *merging threshold*. The target attribute for both states is action a_1, the action with the highest probability. As displayed in Figure 12, the merging of states S_1 and S_2 is performed as follows. The transitions from and to the current states S_1 and S_2 are redirected to the merged state S with the probability frequency recalculated accordingly.

5.5 *Experimental results*

The ONASI algorithm with state merging on the Wumpus World data is run to determine the effect of the refinements on performance. The accuracy of prediction stays mostly unchanged with slight changes in some cases. The slight changes were due to the merging point α that was selected. As shown, there is a fluctuation in the accuracy and the number of states, when the merging threshold M_{min} is 0.5. When merging states, we need to consider states that have a high probability for their strongly preferred action. When M_{min} is 0.5, states may be considered for merging that do not have a strongly preferred action with a large corresponding probability (for example, states with an action probability of 0.51). As a result, if the merging threshold is too low, ONASI may generate predictions based on noise.

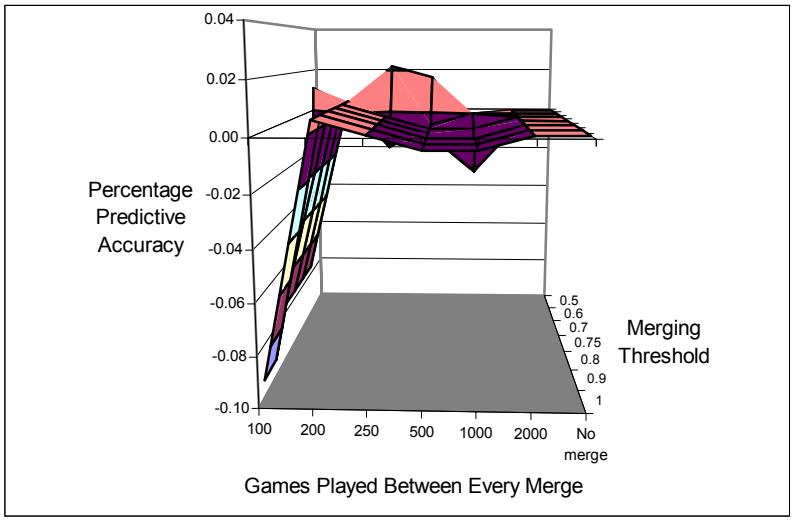


Fig. 13. Predictive accuracy.

There is a drastic reduction in the number of states in the model. There is approximately a 50% reduction in the number of states in the model. Figures 13 and 14 display the predictive accuracy and the increase in model size, respectively.

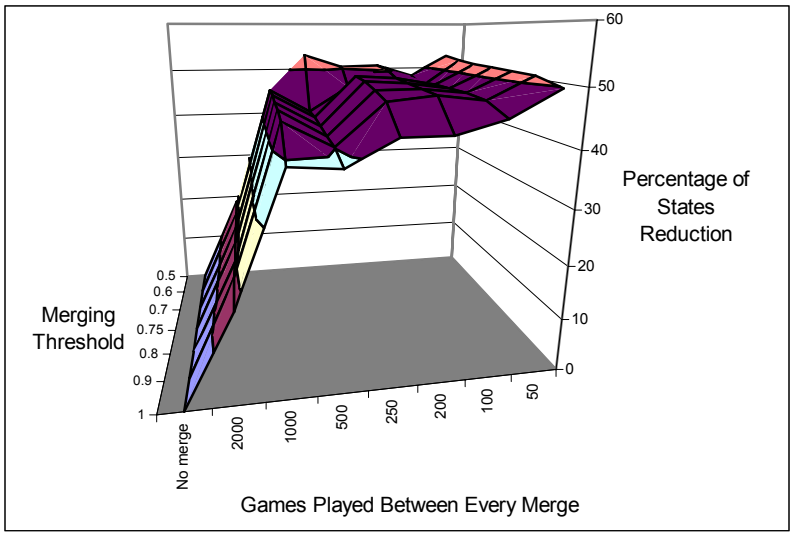


Fig. 14. Increase in model size.

5.6 *Splitting and merging combined*

We combine the splitting algorithm and merging algorithm and use the final ONASI algorithm to determine the performance that can result from these enhancements. By combining both, the best results are obtained, where splitting states provides a slightly better predictive accuracy than the original model, and merging of states decreases the number of states.

The drawbacks of each splitting and merging are resolved by combining them. In splitting states the draw back was the increase in the number of states. By merging states, we address this problem. From the experimental results we obtain the best model when the splitting threshold A_{min} is 0.5 and the merging threshold M_{min} is 0.9.

We wish to determine if the difference in performance between the original model and refined model used by ONASI is statistically significant. A one sided, upper tailed, matched-pairs T-test is selected to perform this determination.

To show that the improved model yields a better predictive accuracy, we consider the relative performance of these two models averaged over all the training sets of size 1,900 games that might be drawn from the data set. Using the data set of 2,000 games to compare the predictive accuracy of the two models, this data set D is divided into a training set D_{train} (1,900 games) and a disjoint test set D_{test} (100 games). The training set is used to train both models, and the test data is used to compare the two accuracies.

Using the matched-pairs T-test and n-fold cross validation with n=20, our null hypothesis is described as the mean of the difference between the initial model and the improved model. A sample mean of 3.99 and a standard deviation of 0.672 are obtained.

The test statistic value that was obtained was 26.6 with 19 degrees of freedom. The probability of this result, assuming the null hypothesis, is less than 0.0001. Therefore we reject the null hypothesis and conclude that we have empirical evidence to support our research hypothesis that the improved model has significant predictive accuracies with respect to the initial model, with a risk of a Type-I error that is less than 0.0001.

6. Conclusions

In this research, we have concerned ourselves with deriving a scalable Markov model of the agent's behavior only from the observations, without incorporating the application details or the agent's goals into the model. Our approach was tested by creating a software agent to play a game called the Wumpus world. Our ONASI algorithm is designed to build a model using just observed agent actions. The ONASI algorithm was very successful in building the initial model in the Wumpus world domain. The initial model was very fast to learn with a predictive accuracy of 88.97%. We conclude that as we include more features, the better our model becomes. To improve the model, the previous action the agent executed was included into the current state of the model. Doing so, there was an increase of the prediction accuracy to 92.77%, but it increased the size of the model as well.

ONASI is enhanced once more to split states to include history when making predictions from a state. We show that only the relationship between the previous states and the next states is necessary in making good splits. This avoids unnecessary resource overheads incurred by retaining the complete action history. Although we expected a large increase in the predictive accuracy, there was only a slight increase. This was due to the domain and the strategies the software agent performed. Implementing a software agent that uses a lengthy history in making decisions would further increase the predictive accuracy.

Including the previous action in the state representation and adding split states resulted in an increase in the model size. Therefore, ONASI selectively chooses states to merge. Clustering states with similarities performs the selecting of states. A drastic drop in the model size was observed, keeping the predictive accuracy mostly unchanged.

Something important to notice about the ONASI algorithm is when building the initial model there should be a facility to extract local information from the observation in such a domain. For example, we took the adjacent four locations (plus the current location) of the agent as the local environment, where in some domains the adjacent eight locations define the local environment. There is a need to establish the meaning of the local environment especially when the global environment is not visible. Another important note is that in specific domains, other applications might do better than ONASI due to the knowledge of the domain incorporated into the application. However, in domains where the specific details are not completely known, ONASI algorithm would perform well. Overall, we conclude that ONASI is an effective and scalable tool for modeling an agent's behavior.

We hope to apply the ONASI algorithm to detect behavior patterns in humans, not only in the Wumpus world domain, but also in other domains. There remains also some analysis left to do in the comparison of ONASI to other models. In the future, we hope ONASI will help user modeling in areas such as profiling user behaviors, as well as in military areas where behavior of the enemy can be modeled in an unknown battle terrain.

References

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

[2] T. Brants, "Estimating HMM Topologies," *Tbilisi Symposium on Language, Logic, and Computation,* Human Communication Research Center, Edinburgh, 1995.

[3] T. Brants, "Better Language Models with Model Merging," *Proc. Conf. on Empirical Methods in Natural Language Processing*, Philadelphia, PA, 1996.

[4] T. Brants, "Estimating Markov Model Structures," *Proc. 4th Conf. on Spoken Language Processing*, 1996.

[5] B. D. Davison and H. Hirsh, "Predicting Sequences of User Actions," Technical Report, Rutgers, The State University of New York, 1998.

[6] T. G. Dietterich, "Proper Statistical Tests for Comparing Supervised Classification Learning Algorithms," Technical Report, Department of Computer Science, Oregon State University, Corvallis, OR, 1996.

[7] P. J. Gorniak, "MailMind - A Connectionist E-Mail Sorting Agent," Honors Thesis, University of British Columbia, 1998.

[8] P. J. Gorniak and D. Poole, "Predicting Future User Actions by Observing Unmodified Applications," *Proc. of the National Conference on Artificial Intelligence*, 2000.

[9] P. J. Gorniak, "Keyhole State Space Construction with Applications to User Modeling," Masters Thesis, University of British Columbia, 2000.

[10] E. Horvitz, J. Breese, D. Heckerman, D. Hovel and K. Rommelse, "The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users," *Proc. 14th Conf. on Uncertainty in Artificial Intelligence*, 1998.

[11] B. Kerkez and M. T. Cox, "Local predictions for case-based plan recognition," *Advances in case-based reasoning: Proc. 6th Conf. on ECCBR*, eds. S. Craw & A. Preece, 2002.

[12] B. Kerkez and M. T. Cox, "Incremental case-based plan recognition using state indices," *Proc. of the 4th Int. Conf. on Case-Based Reasoning*, 2001.

[13] B. Korvemaker and R. Greiner, "Predicting UNIX Command Lines: Adjusting to User Patterns," *Proc. of the National Conf. on Artificial Intelligence*, 2000.

[14] T. Mitchell, Machine Learning, McGraw-Hill, 1997.

[15] R. A. McCallum, "Instance-Based Utile Distinctions for Reinforcement Learning," *Proc. 12th Int. Machine Learning Conference*, 1995.

[16] S. M. Omohundro, "Best-First Model Merging for Dynamic Learning and Recognition," Technical Report*,* International Computer Science Institute, Berkeley, CA, 1992.

[17] J. Quinlan, "Induction of Decision Trees," Machine Learning 1(1), Boston: Kluwer, 1986.

[18] P. Sandanayake and D. J. Cook, "Imitating Agent Game Strategies Using a Scalable Markov Model," *Proc. 15th Florida Artificial Intelligence Research Symp.*, Pensacola, FL, 2002.

[19] C. Shannon and W. Weaver, "The Mathematical Theory of Communication," University of Illinois Press, Urbana, 1949.

[20] A. Stolcke and S. M. Omohundro, "Best-First Model Merging for Hidden Markov Model Induction," Technical Report, International Computer Science Institute, Berkeley, CA, 1994.

[21] I. Zukerman, D. W. Albrecht and A. E. Nicholson, "Predicting Users' Requests on the WWW," *User Modeling: Proc. 7th Int. Conf.*, 1999.

Priyath Sandanayake received the B.S. and M.S. degrees in Computer Science and Engineering from The University of Texas at Arlington, Texas, U.S.A. in 1998 and 2002 respectively.

Diane Cook is currently a Professor in the Computer Science and Engineering Department at the University of Texas at Arlington. Her research interests include artificial intelligence, machine learning, data mining, robotics, and parallel algorithms for artificial intelligence. She has published over 120 papers in these areas. Dr. Cook received her B.S. from Wheaton College in 1985, and her M.S. and Ph.D. from the University of Illinois in 1987 and 1990, respectively.