Using Dynamic Decision Networks to Make Rational Tutorial Actions

Peter S. Carbonetto
Department of Computer Science
University of British Columbia
Vancouver, British Columbia
pcarbo@cs.ubc.ca

December 4, 2002

1 Introduction

For this project I propose a dynamic decision network for the purpose of modelling the actions of a logic gate tutoring system.

With respect to research on artificial intelligence in ITS, building a tutoring system based on dynamic decision networks holds some advantages over ad-hoc methods because it can be easily extended to apply machine learning techniques in order to build adaptive tutors, and it is a general framework grounded in extensive research that can help rationalize user behaviour. It was initially motivated by the work of [5] but I will specify priors instead of basing them on empirical data sets; my intent is not to design an empirically-specified tutoring system but rather to gain skills that can be applied and extended to a normative framework for ITSs.

We can consider the interaction of a student with the tutor as an ongoing dialogue. In this setting, whenever the student acts on the interface, the tutoring system must consider an appropriate response (note that the best response in some situations could very well be no response at all). The tutor's selection for the best action to execute depends on the objectives of the tutor and the tutor's current belief in the state of the student. In order to obtain a quantitative measure of what a "good" action is, the tutor has a predetermined table of utilities for individual student states. The utility node depends on two things: the expected knowledge of the student and the expected morale of the student.

Dynamic decision networks model situations in which chance nodes change over time. A set of nodes respresenting values at a point in time is called a *slice*. The proposed system takes a greedy approach to action selection by considering only two slices into the future, a similar approach to the ITS of Murray and colleagues [7],[8]. The rationale supporting this assumption is that an immediate increase in utility of the student state leads to a high increase in utility over

time. Realistically, this is not necessarily the case: for example, hinting may result in an immediate increase in knowledge, but over the long term it may hamper the student's ability to self-explain [3]. In summary the objective of the tutor is act to maximize the expected utility of the student's state after she or he performs the most likely action.

Typically, as new slices are added to the network old ones are removed and the influence diagram is recomputed in order to integrate accumulated evidence, as illustrated in the HUGIN system [4]. When the tutor is not making decisions about its future actions, it only keeps track of one state – the last observed state of the user. It does not have to remember past states since the current state can completely determine the future states. When a decision needs to be made in response to a user action, the decision network is dynamically built using the representation of the student's current state. It should be noted that the conditional probability tables in the network remain static throughout a tutoring session – updating them based on evidence would improve the flexibility of the tutor, but it is beyond the scope of this project.

Inference in dynamic decision networks is computationally expensive and it many situations it may not be realistic to make a decision in real time. Some systems (e.g. [8]) use anytime methods to compute approximate results within a specified time frame in order to ensure real-time interaction with the student. This would be desirable for this project, but it is probably not realistic to implement robust stoachastic sampling on bayesian networks in the given time frame for the project. I will assume that the student model is simple enough that all decisions can be resolved in a reasonable amount of time.

The dynamic decision network proposed in this project could potentially be extended to other pedagogical agent tasks, such as problem selection. Since the decision process is very explicit, a reasonably accurate selection involves predicting the student's state upon completion of the problem – in other words, predicting several slices into the future.

2 Implementation Details

2.1 The User Interface

The graphical user interface plays a crucial role in the success of any program. This cannot be overemphasized for intelligent tutor systems since a poorly designed interface can interfere with the learning process. Most publications rarely mention the role of the GUI only because its importance is widely appreciated. The domain of the Logic Gate Tutor is simple enough that the graphical design is straightforward, but design of the GUI still time consuming and it is easy to create a poor user interface.

The graphical user interface was designed using the Fox C++ toolkit library [1]. The authors of the graphics toolkit placed a high importance on platform-independent and as a result they de-emphasized its ease of use. Since it the API

[1] www.fox-toolkit.org

is unintuitive, I do not recommend the FOX graphics library for developers.

Upon startup, the user is asked to enter his or her name. If the system recognizes the name from a previous session, that session is loaded from a file. There is a single sessions index file that keeps track of the names of all the previously conducted users sessions. Each user name is associated with an individual sessions file. Each separate file keeps track of the user's session information, including completed exercises and the probabilistic user state. If the user enters a name that has never before been encountered, an entry is added to the sessions index and the user state is initialized to the estimated priors.

Upon starting a new session or resuming an old one, the user is asked to select from a list of exercises. This dialog window also appears whenever the user has completed an exercise. A separate text file stores information about an exercise, and consists of a list of gates, inputs, wires and user inputs. For each gate we specify a label string, the type of gate (AND, OR, NOT or XOR) and how the gates are connected to each other. Similar information is specified for the user inputs of the logic gate exercise.

Each user input consists of a logic table, initially filled with zeros. Clicking on one of the zeros switches the value to one, and clicking on a one switched the value back to zero. Once the student is satisfied with logic table, pressing the "OK" button sends a message to the pedagogical agent to update the state of the user and make a decision based on the current user action and state. Note that the tutoring agent does not make a decision unless it receives a response from the GUI since this is an event-driven system. If the user has entered the correct values in the logic table, the user will not be allowed to re-enter the values. The user must input the correct values for all the logic tables before the question is completed. The student can solve the problem in any order.

There are some buttons along the side of the main window that allow the user to quit, start a new session and start a new exercise (say, the user may be fed up with the current one). There is also a button called "show statistics"; this pops up a new window that displays information on the user state with bars of different colours, where longer bars indicate higher probabilities. Figure 1 shows the working interface.

To aid development the system reads in the probability tables directly from an Excel spreadsheet so small modifications in the conditional probability tables do not necessitate a recompile of the program.

2.2 The Student Model

The objectives of the pedagogical agent can be separated into two possibly contrasting groupings: increase the user's knowledge in the domain of interest and, secondly, maintain a desirable affective state. Through interaction with the graphical user interface, the agent attempts to influence the user's knowledge and affective state. To address these influences we must develop a model that takes into account the student's current system of beliefs and affective state and updates this model over time as the user interacts with the program interface. Accordingly we need a reasonably approximate model of the student's beliefs,

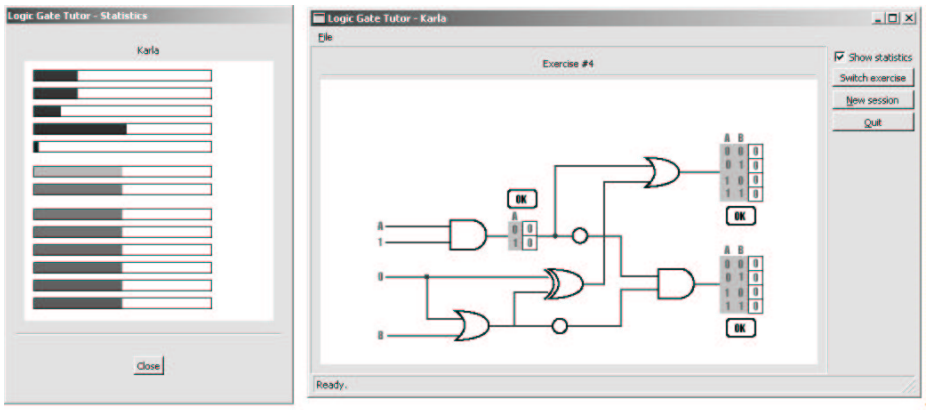

Figure 1: The graphical user interface for the Logic Gate Tutor program. The window on the left displays the probabilities of the user state. Each bar represents a node in the user state. The bottom six bars represent the action utilities for the six different actions. The window on the right is where all the user interaction takes place. A sample exercise is shown.

emotional state and goals and in order to achieve a sound decision-making processes.

The program's internal model of the user is simple, and cannot serve to fully describe the complex learning and tutor-student interaction that takes place during the course of a tutoring session. However it is the hypothesis of the project that the user model described below is useful in supporting the pedagogical agent's interaction decisions.

Keeping track of the user's affective state is not an easy task since there is a great deal of uncertainty between the emotions of the user and the observable effects [2]. This challenge is further exacerbated by the fact that convential computers are not good at acquiring the rich and complex percepts in humans that indicate emotional state. One approach is to limit the space of possible behaviours through constraints in interaction, thereby limiting the set of outcomes and making the problem of predicting the specific emotions tractable. This methodology is used in the ABAIS architecture [6].

In this project, the problem is artificially reduced in complexity by considering only two binary affective states, labeled Independence and Morale. The Independence state is defined as the student's ability to independently solve and overcome problems without the aid of the tutor, and the Morale state is the student's general sense of enthusiasm and confidence with respect to the task at hand.

The knowledge of the student is divided into five states: the student's understanding of the four different gates (AND, OR, NOT and XOR gates) and the student's proficiency at combining the output tables of these gates into a single logic table. The latter skill is emphasized in this educational task since it is the

most general and perhaps more difficult to acquire. Therefore, more weight is placed on this combining proficiency in updating the student's knowledge in the decision network.

The procedural knowledge is considered *flat* for the purposes of this domain, in contrast to the hierarchical model used by Brown and Burton [1]. It does not model the student's subgoals in the overall task. For more robust explanations hierarchical models may be needed, although this would probably have to be coupled with changes to the current interface. One could make the argument that the knowledge in the domain is declarative and not procedural; I assume that such a distinction is not relevant for this project since it does not help in guiding the construction of the proposed decision framework. While the proposed knowledge model is very simple, I believe it is sufficient for the tutor to make non-trivial decisions regarding hinting.

One disadvantage of this inelaborate representation is that the tutor does not keep track of which combinations of gates the student has seen. Ideally, the tutor would gain more confidence of the student's ability to combine gates if the student has seen a variety of combinations instead of the same ones over an over again. Consequently, there is a certain degree of reliance on the exercises being varied enough to justify this simple knowledge representation.

2.3 The User Input Model

The user input model is dependent on the current state of the exercise; the probability nodes are constructed on demand. There are three different types of nodes that directly represent the user's interaction with the tutoring environment.

When the user has entered the binary values directly into the probability table, the "success on input" node prior is set to 1. Otherwise it is set to 0.

The "failure on previous input" node is needed to predict the student's independence and morale. Keeping this extra bit of information reduces the complexity of the entire network. At any one time we only need to look at the current state t in order to predict future states $t + k$. Additionally, construction of the on-demand model of user input is simpler if we store the user's failure on the previous input. By default, the "failure on previous input" is 1 if the user has not yet interacted with a particular interface input.

The last coupled probabilistic state is in fact a set of states, representing the agent's previous actions (*i.e.* hinting). These binary nodes are labeled "gave hint for AND", "gave hint for OR", "gave hint for XOR", "gave hint for NOT" and "gave hint regarding combining gates".

2.4 Updating the User State

Nodes in the network represent fixed values. To represent changes in the user state over time, I use dynamic belief networks [9]. Whenever the agent needs to update the state of the user, it creates a new *slice* representing the state at time $t + 1$. The probabilities are then propagated into the new time slice

based on information about the time t state and the percepts derived from the user interface. This process of evolving the belief in the state of the world over time where $P(state_{t+1}|state_t, percepts_{t+1})$ is called a Markov chain. Kalman filtering is used to update the state. As new slices are added to the network, the Kalman filter accumulates the probabilities from the previous slices and removes the previous slice, so over time the current belief in the system is always intregrated into a single state.

We can describe the process evolution in more detail by breaking it down into three stages. In the first stage, *prediction*, we begin with a two-slice network, $state_{t-1} \longrightarrow state_t$. We assume that we've already calculated our belief in $state_t$. $P(state_{t+1})$ is given by the following equation:

$$\frac{1}{k}P(Percepts_{t+1}|State_{t+1}) \times \sum_x P(State_{t+1}|State_t = x)P(State_t = x) \quad (1)$$

where k is the normalization constant. Once we have infered the probabilities in the new time slice, the second step is to marginalize the probabilities by removing slice t and now the probabilities in slice $t+1$ are priors. To prepare for the next prediction, the last step consists of adding a slice and the conditional probabilities for slice $t+2$. This entire process is illustrated in Figure 2. The conditional probability tables used in the update process are shown in Tables 2 through 5 in the Appendix. Additionally, the priors for the user state are given in Table 1.

The tutoring system uses the SMILE portable decision network inference engine, developed for the Visual C++ Windows programming environment, to run the kalman filter described above. [2].

2.5 Selecting Actions from the User State

Once the tutoring agent has updated its internal user state representation, it has a variety of actions it can perform: it can provide a hint relevant to the user's focus in the exercise or it can do nothing. In many situations, students may be better without any help from the tutor. There are several reasons for this: for one, it gives an opportunity for students to learn from their own mistakes, and it allows the student to address specific weaknesses in their skills more precisely than a computer tutor. Furthermore, interaction with the tutor may be detrimental to the student's learning process if the hints are provided at an inappropriate time. These points have led towards the interest in pedagogical agents that support self-explanation. A more thorough study is given in [3]. A consideration of the role of self-explanation in learning since it is out of scope of the project. However, we need to consider the appropriateness of carrying out an action for a given situation, represented with a utility node. An implicit assumption of this project is that the appropriateness of an action is solely based on the value of the resultant state.

[2] www.sis.pitt.edu/ genie

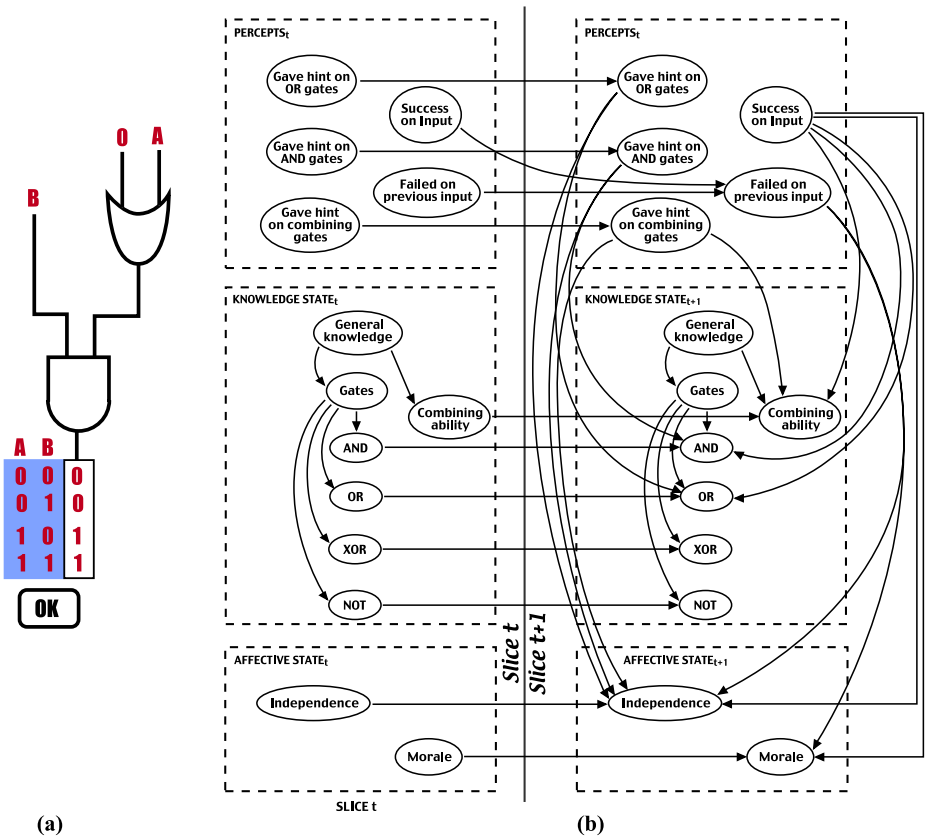

Figure 2: Figure (b) represents the state of the logic gate exercise shown in Figure (a). At this point, the network is ready for the prediction stage since it has the nodes at slices t and $t+1$ and the conditional probabilities between the two slices.

The interaction of the tutor with the student can be considered as a two-party conversation, where each one responds in turn to the action of the other. To model the process evolution we need to represent both the action of the tutoring agent and the student. I use a decision node for the agent's action and a probability node for the student, since belief in the student's subsequent action is based on the current state. An entire *cycle* consists three slices, where the first slice is the current user state, the second slice is the result after the tutor's action and the third slice is the outcome after the student has acted on the graphical user interface. Given a measure of utility of the user's final state, decision theory holds that a rational agent should select an action that produces the maximum expected utility. This process is represented abstractly in Figure 3.

Using the Kalman filtering algorithm described in the previous section, we

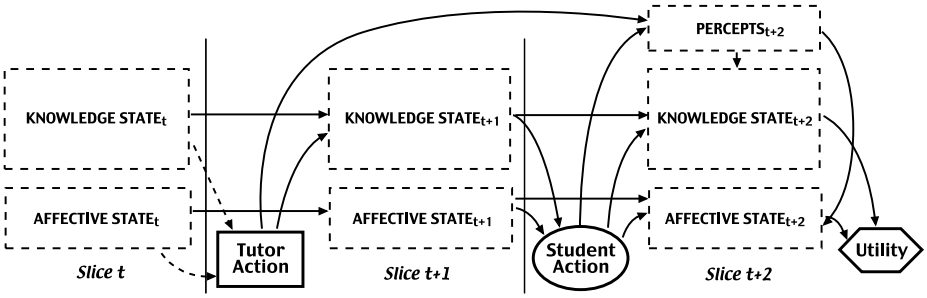

Figure 3: A consise look at the three-state decision cycle. The dotted lines represent implicit dependencies in the network, but are not actually used to calculate the optimal tutor action.

can infer the probabilities of the nodes in the slices. It is a simple extension of this algorithm to compute the expected utility EU of each tutoring agent action using the following formula:

$$EU(A \mid E) = \sum_{result} P(result(A) \mid E) \times U(result(A)) \tag{2}$$

where E is the evidence in the network and A is the action of the tutor. Since all the agents can be done with equal probability, we can assume that $P(result(A) \mid E) = 1$ for all actions. Therefore, the utility of any action is simply equal to the sum of the utilities of the results.

The work described above for updating the user state can be reused with slight alteration. The only addition is the agent action node which alters the probabilities of the user's knowledge state and the "gave hint for input" percept node in slice $t + 1$. The one complication is that we don't want the tutor to perform the same action more than once for a single input in the exercise. My solution is to filter out previously performed actions after the tutor has calculated the expected utilities of the actions. Obviously, "no action" can be performed without limit. A more developed system might have an array of possible hints to provide, in which case the agent may want to select the same action more than once. See Figure 4 for the agent action subnetwork.

Predicting the user's action is not quite as straightforward since the user's performance in the exercise depends on the specific composition of logic gates. For example, knowledge of OR gates will not help very much if the exercise consists solely of AND gates. In addition, we need to have a measure of difficulty of the problem in order to properly determine the student's success. The user's action is represented simply by the "Success on Input" node. The challenging part is defining the dependency of the student's success on input given his or her knowledge and affective state.

The approach used here is grab a snapshot of the state of the exercise and use this as a basis for the conditional probability table. First of all, we consider only the active logic gates – in other words, the gates in which the student has

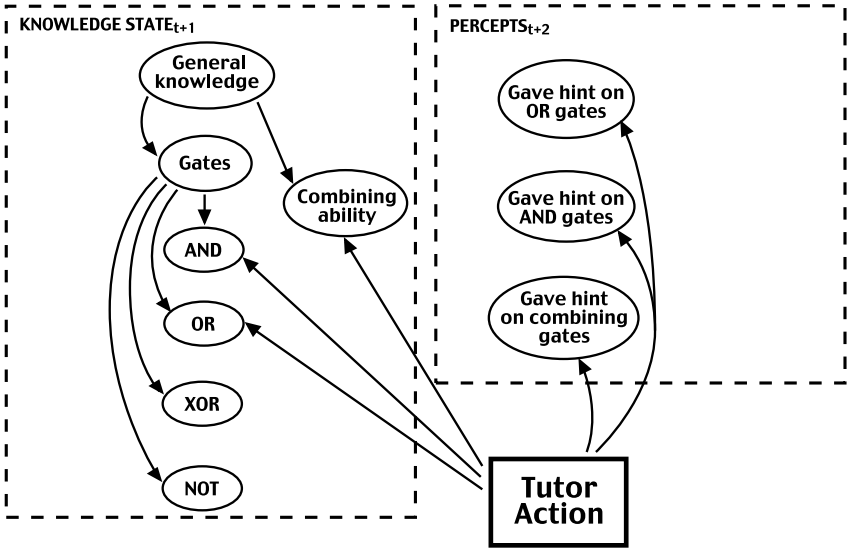

Figure 4: The tutoring agent action subnetwork. Note that this is a continuation of the same example of Figure 2.

not already inputed a correct solution. This is a concern because the logic gates may overlap on several inputs an exercise. Once we have identified the relevant gates, we have to assign a measure of the difficulty to each skill in question. For example, as the number of gates increases the probability of the student providing a correct solution diminishes. This is not a linear relationship since a student that is able to solve a problem with six gates should be able to extend his or her abilities to a problem with seven gates with little difficulty. On the other hand, going from one gate to two gates is much more difficult for the beginner student. A similar relationship holds for the AND, OR, XOR and NOR gate skills. I approximate this non-linear relationship with a piecewise linear function, as shown in Figure 5.

These importances are used to calculate the component probabilities p_i using the following formula:

$$p_i = \frac{1}{\sum_i importance_i} P(X_i)^{importance_i} \tag{3}$$

where X_i is the prior probability of once of the parent nodes (e.g. the $combining\ ability_{t+1}$ node) and $importance_i$ is the assigned importance, using the equations illustrated in Figure *. These component probabilities p_i are then added together to form the conditional distribution of $P(Success\ on\ input)$.

The configuration of the user action with respect to the time $t+1$ slice is shown in Figure 6.

An advantage with this framework is that it allows the tutor to predict and possibly prevent student actions that lead to poor outcomes. For example, it can prevent future errors on the student's part.

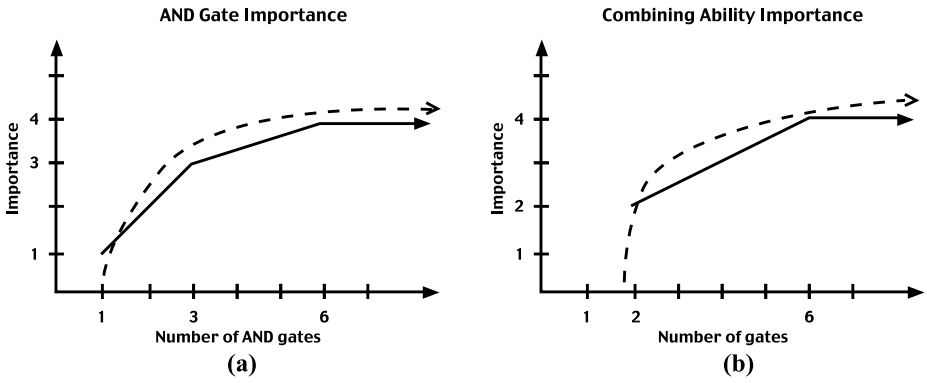

Figure 5: (a) and (b) are the hypothetical graphs of the importance measurements for the AND gate skill and combining ability, respectively. Note that (a) applies to all of the gates, not just AND.

3 Preliminary Evaluation and Discussion

Preliminary evaluations hold some promise but deficiencies are apparent. There are some indications that this project is headed in the right direction but it appears that this simple model is in fact too simple to adequately describe the user's state in order to make informed decisions. Therefore, early investigations show that the hypothesis of this project is not valid but there are still some positive things to take away from this work.

The model is good at measuring the student's general confidence abilites through the morale, independence and knowledge nodes and it is good at predicting when the student is floundering. However, the information in the predicted user state is not rich enough to direct the actions of the tutoring agent in a very useful manner. This is where a model using something like path finding may have served better [3]. An alternative conclusion one could make from the preliminary observations is that the nodes in the user state are too tightly coupled, and a different – and perhaps broader – combination of knowledge and affective states may serve better for the decision- making process. For example, the Independence state appears to be useful in guiding decisions but under less common circumstances. The overall persuation is that a greater emphasis should be placed on a precise mapping of the logic gates domain to a probabilistic network before consideration of a more extensive affective model.

Another possible shortfall of the belief network is that it places little emphasis on knowledge states with low probability; it treats all increases in probability with the same measure of utility. Ideally, the tutor should emphasize improving the weakest knowledge states before addressing global increases in knowledge. This could be managed through probability nodes that represent several levels

[3]See the chapter on "Student modeling" in M.C. Polson and J.J. Richardson, *Foundations of Intelligent Tutoring Systems*, pp. 55-78.

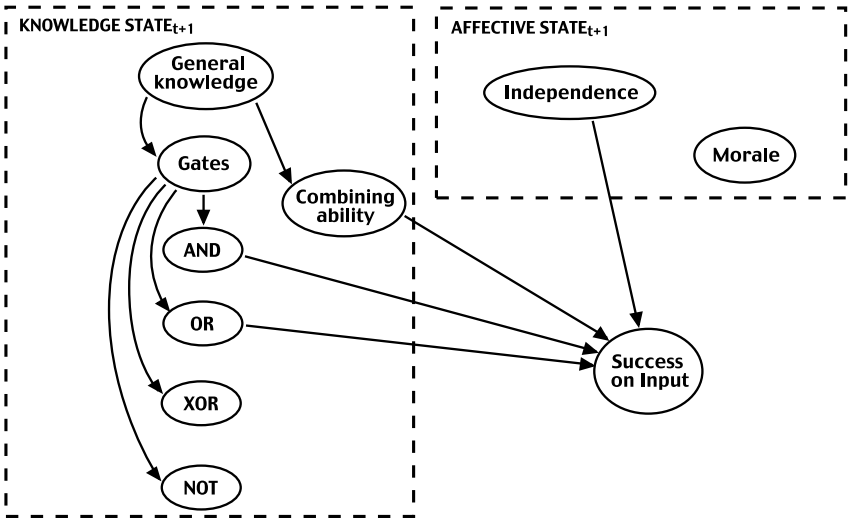

Figure 6: The user action subnetwork. This is a continuation of the example in Figure 2. Note that the "success on input" node represents what we entitle the user action node.

of mastery.

The user's comprehension rate or skill-acquiring rate would be useful to model in such a system, especially with regards to tutor-guided selection of exercises. However, this would certainly involve a more elaborate state updating process since it would it would need to maintain a history of slices.

Having cast the results in a negative light, there are still some positive outcomes. The dynamic decision networks framework is a good framework for optimally selecting tutorial actions, provided of course that it accurately represents the learning process of the domain in question. Bayesian networks are astonishingly insensitive to errors and may in fact compensate for small errors in conditional probability tables. There are still some issues with tractability, but the advantages of decision-theoretic networks, including the ability to make tradeoffs in the selection of discourse actions in the face of multiple objectives, show that this approach holds promise in the domain of intelligent tutoring systems.

References

[1] John S. Brown & Richard R. Burton. Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, Vol. 2, 1978, pp. 155-192.

[2] Cristina Conati. Probabilistic Assessment of User's Emotions in Educational Games. *Journal of Applied Artificial Intelligence*, to appear.

[3] Cristina Conati & Kurt VanLehn. Toward Computer-Based Support of Meta-Cognitive Skills: a Computational Framework to Coach Self-Explanation. *International Journal of Artificial Intelligence in Education*, Vol. 11, 2000, pp.

398-415.

[4] T. Huang, D. Koller, J. Malik, G. Ogasawara, B. Rao, S. Russell and J. Weber. Automatic Symbolic Traffic Scene Analysis Using Belief Networks. *Proceedings of the Twelfth Conference on Artificial Intelligence*, 1994.

[5] Michael Mayo and Antonija Mitrovic. Optimising ITS Behaviour with Bayesian Networks and Decision Theory. *International Journal of Artificial Intelligence in Education*, Vol. 12, 2001, pp. 124-153.

[6] Eva Hudlicka & Michael D. McNeese. Assessment of User Affective and Belief States for Interface Adaptation: Application to an Air Force Pilot Task. *User Modeling and User-Adapted Interaction*, Vol. 12, 2002, pp. 1-47.

[7] R. Charles Murray & Kurt VanLehn. *DTTutor:* A Decision-Theoretic, Dynamic Approach for Optimal Selection of Tutorial Actions. *5th International Conference on Intelligent Tutoring Systems*, 2000.

[8] R. Charles Murray, Kurt VanLehn & Jack Mostow. A Decision-Theoretic Appracoh for Selecting Tutorial Discourse Actions. *Proceedings of the NAACL Workshop on Adaptation in Dialogue Systems*, June 2001.

[9] Stuart J. Russell & Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

Appendix

Table 1: User state priors.

X	$P(X = true)$
General Knowledge	0.01
Independence	0.5
Morale	0.5

Table 2: Probability table for $AND_{t+1} \mid Success\ on\ input, Gave\ hint, AND_t$. Note that this table also applies to the other gates.

$Success\ on\ input$	$Gave\ hint$	AND_t	$P(AND_{t+1} = true)$
T	T	T	0.90
T	T	F	0.10
T	F	T	1.00
T	F	F	0.30
F	T	T	0.50
F	T	F	0.01
F	F	T	0.70
F	F	F	0.02

Table 3: Probability table for $Combining\ ability_{t+1} \mid Success\ on\ input$, $Gave\ hint, Combining\ ability_t$.

$Success\ on\ input$	$Gave\ hint$	$Comb.\ abil._t$	$P(Comb.\ abil._{t+1} = true)$
T	T	T	0.80
T	T	F	0.06
T	F	T	1.00
T	F	F	0.12
F	T	T	0.40
F	T	F	0.01
F	F	T	0.70
F	F	F	0.02

Table 4: Probability table for $Morale_{t+1} \mid Failed\ previous$, $Success\ on\ input, Morale_{t-1}$.

$Morale_t$	$Failed previous$	$Success\ on\ input$	$P(Morale_{t+1} = true)$
T	T	T	1.00
T	T	F	0.70
T	F	T	1.00
T	F	F	0.90
F	T	T	0.30
F	T	F	0.01
F	F	T	0.10
F	F	F	0.01

Table 5: Probability table for $Independence_{t+1} \mid Gave\ hint\ for\ input, Success\ on\ input, Failed\ on\ previous\ input, Independence_t$.

$Gave\ hint$	$Ind._t$	$Failed\ previous$	$Success\ on\ input$	$P(Ind._{t+1} = true)$
T	T	T	T	1.00
T	T	T	F	0.50
T	T	F	T	1.00
T	T	F	F	0.99
T	F	T	T	0.20
T	F	T	F	0.01
T	F	F	T	0.01
T	F	F	F	0.01
F	T	T	T	1.00
F	T	T	F	0.70
F	T	F	T	1.00
F	T	F	F	0.99
F	F	T	T	0.40
F	F	T	F	0.01
F	F	F	T	0.02
F	F	F	F	0.02